Deep Learning in Kernel Machines

Submitted in partial fulfillment of the requirements
for the award of the degree of
DOCTOR OF PHILOSOPHY

by

AFZAL A. L.
Reg.No : 4856

under the supervision of

Dr. ASHARAF S
Associate Professor
Indian Institute of Information Technology and Management - Kerala (IIITM-K)
Thiruvananthapuram, Kerala

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)
KOCHI, KERALA, INDIA- 682 022

Conducted by

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT - KERALA (IIITM-K)
THIRUVANANTHAPURAM, KERALA - 695581

September 2018

Deep Learning in Kernel Machines

Ph.D Thesis under The Faculty of Technology

Author :

AFZAL A. L.
Ph.D Research Scholar
Indian Institute of Information Technology and Management - Kerala (IIITM-K)
Thiruvananthapuram, Kerala, India – 695581

Supervising Guide :

D. ASHARAF S.
Associate Professor
Indian Institute of Information Technology and Management - Kerala (IIITM-K)
Thiruvananthapuram, Kerala, India – 695581

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)
KOCHI, KERALA, INDIA- 682 022

Conducted by

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT - KERALA (IIITM-K)
THIRUVANANTHAPURAM, KERALA - 695581

Dedicated to my family

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT - KERALA (IIITM-K)
THIRUVANANTHAPURAM, KERALA - 695581

Certificate

Certified that the work presented in this thesis entitled ***"Deep Learning in Kernel Machines "*** *is based on the authentic record of research done by* ***Mr. AFZAL A. L.*** *towards the partial fulfillment of the requirements for the award of degree of* ***Doctor of Philosophy*** *of the Cochin University of Science and Technology, under my guidance and supervision and that this work has not been submitted elsewhere for the award of any degree.*

Signed: _____

Dr. ASHARAF S
(Supervising Guide)
Associate Professor
Indian Institute of Information Technology and Management - Kerala (IIITM-K)
Thiruvananthapuram, Kerala

Date: _____

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT - KERALA (IIITM-K)

THIRUVANANTHAPURAM, KERALA - 695581

Certificate

Certified that the work presented in this thesis entitled ***"Deep Learning in Kernel Machines"*** *submitted to Cochin University of Science and Technology by* ***Mr. Afzal A.L.*** *for the award of degree of Doctor of Philosophy under the faculty of Technology, contains all the relevant corrections and modifications suggested by the audience during the pre-synopsis seminar and recommended by the Doctoral Committee.*

Signed: _____

Dr. ASHARAF S
(Supervising Guide)
Associate Professor
Indian Institute of Information Technology and Management - Kerala (IIITM-K)
Thiruvananthapuram, Kerala

Date: _____

Declaration

I hereby declare that, the work presented in this thesis entitled ***"Deep Learning in Kernel Machines"*** *is based on the original research work carried out by me under the guidance and supervision of Dr. Asharaf S., Associate Professor, Indian Institute of Information Technology and Management - Kerala (IIITM-K), Thiruvananathapuram – 695581 in partial fulfillment of the requirements for the award of the Degree of Doctor of Philosophy. I further declare that no part of the work reported in this thesis has been presented for the award of any degree from any other institution.*

Signed: _____

AFZAL A. L.
Ph.D Research Scholar
Indian Institute of Information Technology and Management - Kerala (IIITM-K)
Thiruvananthapuram, Kerala

Date: _____

Acknowledgments

Firstly, I would like to express my sincere gratitude and admiration to my advisor Dr. Asharaf S. for his motivation, valuable scientific guidance, constructive feedback and the continuous support of my Ph.D study and related research. I thank him for his patience and fairness to share his immense knowledge. His guidance helped me during the entire course of research and writing of this thesis. I consider myself fortunate to have worked under his guidance.

I am also grateful to our Director prof.(Dr.) Saji Gopinath for his unconditional support and encouragement.

Besides my advisor and director, I would like to thank Dr. Tony Thomas, member of Doctoral Committee for his insightful comments which encouraged me to widen my research from various perspectives.

I am thankful to all the members of Research committee and other faculty members of IIITMK for their valuable comments and support.

My sincere thanks also goes to Dr. Rajasree M.S. , the former director, who provided me an opportunity to join and pursue my research work in this institution.

I also would like to thank my colleagues and friends at Data Engineering Lab for all their constructive views and discussions.

I am grateful to the concerned authorities of my parent Institution, CAPE, College of Engineering Perumon, for sanctioning of leave for my PhD studies.

Last but not the least, I would like to thank my family: my parents, wife and sons for supporting me spiritually throughout writing this thesis and my life in general.

Sincerely

Afzal A.L.

Abstract

The attempt to build algorithms to solve cognitive tasks such as visual object or pattern recognition, speech perception, language understanding etc. have attracted the attention of many machine learning researchers in the recent past. The theoretical and biological arguments in this context strongly suggest that building such systems requires deep learning architectures that involve many layers of nonlinear information processing. Deep learning approach has originally emerged and been widely used in the area of neural networks. The techniques developed from deep learning research have already been impacting a wide range of signal and information processing applications. In the recent past, excited by the startling performance that deep learning approaches have to offer, there are many attempts to embrace deep learning techniques in other machine learning paradigms, particularly in kernel machines. Convex optimization, structural risk minimization, margin maximization, etc. are the some of the elegant features that makes kernel machines popular among the researchers. With the advent of recently developed multi layered kernel called arc-cosine kernel, the multilayer computations is made possible in kernel machines. The multi-layered feature learning perceptiveness of deep learning architecture have been re-created in kernel machines through the model called Multilayer Kernel Machines(MKMs). Support vector machines were often used as the classifier in these models. These deep models have been widely used in many applications that involves small-size datasets. However the scalability, multilayer multiple kernel learning, unsupervised feature learning etc. were untouched in the context of kernel machines. This research explored above problems and developed three deep kernel learning models viz; (i) Deep kernel learning in core vector machine that analyze the behavior of arc-cosine kernel and modeled a scalable deep kernel machine by incorporating arc-cosine kernel in core vector machines. (ii) Deep multiple multilayer kernel learning in core vector machines modeled a scalable deep learning architecture with unsupervised feature extraction. Each feature extraction layer in this model exploit multiple kernel learning framework that involves both single layer and multilayer kernel computations. (iii) Deep kernel based extreme learning machine combines the multilayer kernel computation of arc-cosine kernel and fast, non-iterative learning mechanism of Extreme Learning Machines. The theoretical and empirical analysis of the proposed methods show promising results.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

BPTT Backpropagation Through Time

CNN Convolutional Neural Networks

CVM Core Vector Machine

DBM Deep Boltzmann Machines

DBN Deep Belief Network

DSVM Deep Support Vector Machines

ELM Extreme Learning Machine

GRU Gated Recurrent Unit

KPCA Kernel based Principal Component Analysis

KPCA Kernel based Principle Component Analysis

LSTM Long Short-Term Memory

MEB Minimum Enclosing Ball

MKL Multiple Kernel Learning

MKM Multilayer Kernel Machine

PCA Principle Component Analysis

QP Quadratic Programming

RNN Recurrent Neural Networks

SLFN Single Layer Feed-forward Networks

SVM Support Vector Machines

SVR Support Vector Regression

UMKL Unsupervised Multiple Kernel Learning

CHAPTER 1

Introduction

THE idea of creating an intelligent machine is as old as modern computing. In 1950, Alan Turing devised the mechanism to qualify the intelligent conversation capability of smart computing machines. It attracted the attention of leading scientists like Marvin Minsky and John McCarthy and started the most thrilling field of computer science, Artificial Intelligence (AI). In recent past, a prominent enabler of modern artificial intelligence, machine learning has offered many fascinating solutions in a wide variety of real world applications. The predominant characteristic of machine learning algorithms is its ability to improve the generalization capability in solving problems autonomously by learning from data. It involves learning a hypothesis from examples in such a way that it can be further generalized to unseen data. The parameter turning mechanisms adopted in machine learning algorithms to fit the data often misinterprets this scientific discipline as an extension of parameter optimization problems. This may cause two detracting scenarios such as over-fitting and under-fitting, which should be avoided. Over-fitting is the situation in which the learning models are unnecessarily complex compared to training dataset and it causes the model to fit irrelevant features of data. Under-fitting is the scenario in which models are too simple and not capable enough to cope up with complexness present in large volumes of data. Balancing these two scenarios is the key challenge in designing machine learning algorithms. Machine learning algorithms are extensively used in diverse domains of real world applications employing pattern identification, object recognition, prediction, classification, dimensionality reduction etc. Machine learning algorithms may be broadly categorized as follows [1]:

Supervised learning: In this approach, a supervisor (often represented using labels in a sample dataset) guide the process of training. It necessitates the availability of labelled dataset to infer the appropriate hypothesis. Each member of the dataset, referred as data sample, is represented as a pair of an object and its associated label. The associated label often act as the supervisor in the training process. The training process involve the evaluation of model capturing the approximation between estimated label and the desired label to make appropriate model corrections. When the learned model is capable of producing / classifying the labels of unseen examples (test dataset) correctly, the training is said to have accomplished its goal. This algorithm is often used in data analysis tasks such as classification where the model is used to provide categorical labels to unseen data points and in prediction where the model helps to find future trends.

Unsupervised learning: Unsupervised learning algorithms are used to model applications that involve dataset consisting of unlabelled data samples. In a sense, an unsupervised learning algorithm receives inputs (data samples) without any associated labels or rewards. The training process involved in these algorithms evaluates the similarities or dissimilarities present in the dataset. These algorithms exhibit the potential to discover and leverage the hidden structures / patterns / regularities in the dataset. The unearthed pattern can be used for further decision making. These algorithms are commonly used in tasks like clustering, dimensionality reduction, etc.

Semi-supervised learning: Many popular real world applications leverage a blend of both supervised and unsupervised learning approaches called semi-supervised learning. In this type of learning algorithm, the training dataset encompasses both labelled and unlabelled data samples. The training process starts with labelled data and then trained on unlabeled data to define models.

Reinforcement learning: This approach involve an agent which learn how to interact with the environment by executing an action and evaluating the corresponding reward / penalty received to updates its state. Agent utilizes these rewards to learn the best course of action sequence to achieves a task. In contrast to supervised

learning where the algorithm is trained on the given dataset, reinforcement learning involves a sequential decision making in which the next action depends on the current state of the agent and the observation made by the agent at the current time period. The potential to identify the ideal behavior within a specified context / environment make it a very good candidate in many real world applications.

The invention of Rosenblatt's single layer perceptron, an early initiative in artificial neural networks with supervised learning capability, was a milestone in the development of modern machine learning algorithms [2]. The non-linear learning issues encountered in those models were later addressed by multi-layer neural works. The famous gradient based back propagation algorithm was the primary driving force behind the astounding performance of multi layer neural networks in machine learning applications [3]. In a parallel track, exploration on incorporating statistical approaches in learning algorithms gave rise to a new learning paradigm called statistical learning theory [4]. The prime members of this family are kernel machines which utilizes the mathematical notion of kernel functions to achieve a computationally efficient learning approach called Kernel Trick [5, 6, 7, 8, 9, 10]. Support Vector Machines (SVM) [11, 12, 4, 13, 7] is the most popular learning model among the kernel machines.

In the conventional machine learning algorithms the learning task is accomplished by extracting a pre-designed set of features relevant for the task and then using an appropriate machine learning algorithm. These learning algorithms use only a couple of feature extraction layers and they are often tagged as shallow learning algorithms. Support vector machines [11, 12, 4, 13, 7], Kernel regression [14, 15, 16], maximal entropy models [17, 18], conditional random fields [19, 20], k-Nearest Neighbors [21, 22] and Hidden Markov Models [23, 24] are some of the celebrated shallow learning techniques. The shallow learning techniques have been showing promising results in many real world problems. However, most of the modern intelligent application also demand feature learning capabilities involving the identification and utilization of implicit complex structures with corresponding rich representation of data. The handcrafted feature extraction in shallow learning algorithms necessitate prior knowledge and human intervention, requiring deep domain knowledge to identify relevant features. This manual

process of selecting features seems to be tedious and also requires considerable effort of experts. This process also may lead to inappropriate selection of features. The difficulty in shallow learning algorithms to obtain the appropriate data representation restrict their use in many real world applications that involve natural signals such as human speech, natural sound, images, visual scenes, natural languages and many more. Building such real world applications necessitate data representation mechanisms that cope up with highly complicated and varying functional aspects of such real world applications. It is also stated that the well abstracted, task specific data representation can play a vital role in the generalization performance of many learning models [25].

A pragmatic approach for building a solution in many real world application is to discover, represent and leverage appropriate data abstraction amenable for the given tasks. In this context, the representation learning refers to a class of machine learning algorithms that discover suitable internal representation from the available data. Many recent success stories in machine learning precisely speak the vital role of representation learning, to extract useful information (learned features) from the data samples [26, 27]. Other advantage of representation learning is that it can extract different aspects of internal representations from available raw data [28]. Autoencoder is a well known example of representation learning algorithm. However, single layer of feature / representation learning has its own limitation in addressing many real world problems. To widen the scope and applicability of machine learning, it would be highly desirable to conceive more abstracted representation of data by subsuming multiple layers of feature learning. It is also observed that the use of abstracted and task specific features learned as a hierarchical representation can be useful in many real world application [29]. Further, the biological and theoretical arguments in the context of highly complicated and varying cognitive tasks like visual object recognition, pattern recognition, speech perception, language understanding etc. also suggest the necessity of multiple layers of feature extraction. This multiple layers of representation is often referred as hierarchical representation learning that infer top-level features from observed low-level features with increasing levels of abstractions.

There are several attempts in solving real world problems that employ the power of multilayer representation learning in conjunction with traditional machine learning

algorithms. Deep learning is an emerging trend in this direction that exploits hierarchical representation learning from available data. Deep learning accelerates the process of building more complex concepts out of simpler concepts, through multiple layers of non-linear feature transformations. Each stage in this layered representation involve a kind of trainable feature transformation [30]. For example, deep learning approach resolves the complicated feature mapping from pixel to object identification through multiple layers of feature learning (extractions) as follows. The first hidden layer is defined in such a way that it learn the edges from the set of pixel values (input data), the next layer learns the curves and contours out of learned edges , the third layer is responsible for learning object parts out of these learned curves and contours and finally this abstracted features (object part details) are fed to the classifier for identifying the objects. A fine tuning mechanism is then employed on the entire structure to improve the overall generalization performance of the machines [31].

In general, deep learning is a form of representation learning that attempts to build high-level abstractions from available data using a learning model composed of multiple non-linear transformations [25, 32]. The fast layer wise learning algorithm for Deep Belief Network (DBN) by Hinton [33] was a breakthrough in deep learning approaches. The other contributions towards deep neural network learning such as greedy layer wise training [34], sparse representation with energy-based model [35], Deep Boltzmann Machines (DBM) [36] sparse representation for deep belief model [37], Convolutional Neural Networks [28],Recurrent Neural Networks (RNN) [38] etc. have taken machine learning to greater heights in terms of pragmatic use in real world applications. In addition to the multilayer feature extraction, deep learning approaches exhibit the capability of combining both supervised and unsupervised paradigms. These elegant factors manifest the prominence of deep learning in divergent machine learning application arena such as acoustic modeling [39, 40, 41], sentence modeling [42], face recognition [43], action recognition [44], image classification [45], etc.

Even though the deep learning approaches have been primarily pursuing in the context of neural networks, there are several attempts to adapt deep learning capabilities in other learning paradigms, particularly in kernel machines. Kernel machines are class of machine learning algorithm that rely on a concept called 'kernel trick' [5, 6, 7, 8, 9, 10].

Kernel trick enables machine learning algorithms to attain computations in an implicitly defined high dimensional feature space for learning non linearities without an explicit mapping of data samples. It uses a special function called kernel function(s) that takes data samples in the input space as inputs and computes their inner product in the high dimensional feature space. Any machine learning algorithm that rely only on the dot product between data samples can be kernelized by choosing an appropriate kernel that compute the inner product of data samples in an implicitly defined feature space [10]. Kernel machines have the capability to learn complex decision boundaries by transforming the data representation into the high dimensional feature space called Reproducing Kernel Hilbert Space (RKHS), with a limited number of training samples[9]. Mercers theorem provides the mathematical grounding for qualifying functions as kernel functions [4]. The elegant property of these feature mapping is that it could even lead to computations enabled in an infinite-dimensional feature space. Support Vector Machines (SVM) [11, 12, 4, 13, 7], Support Vector Regression (SVR) [46], Core Vector Machines (CVM) [47], Kernel based Principle Component Analysis (KPCA) [48], etc. are some of the predominant members in the family of kernel machines. Kernel machines often succeeded in attaining attention because of their convex loss functions that eliminates local optima and guarantees global optimum. However, the kernel machines typically involve only a single layer of kernel computation making them shallow architectures. Single layer kernel computation is apparently unequipped to discover the rich internal representations of data and seems to be effective only in modeling simple and well constructed data problems. It necessitates multiple layers of feature extraction (multilayer kernel computation) and scalable mechanisms to widen the scope and applicability of kernel machines.

Recently, there were several attempts to impart deep learning capabilities into kernel machines to empower them with multiple layers of feature extraction capabilities[49, 50, 51]. A breakthrough in this context is the emergence of arc-cosine kernels [49, 52, 53] that mimics the computations in a multilayer neural network. Arc-cosine kernel has the ability to exhibit different behaviors at different layers, which are governed by the activation value or degree at that layer. Multilayer arc-cosine kernels have been widely used in conjunction with SVMs and exhibit the potential to build many real world applications

that involve relatively small sized datasets. Multilayer Kernel Machines (MKMs) was another milestone in this journey of deep kernel machines [49] which enables multiple layers of feature extraction. In this context, this research study identified the need for scalability, the possibility of developing multiple multilayer kernel learning algorithms, the potential for developing a method for unsupervised feature extraction by exploiting multiple kernel learning, etc. are some of the potential explorable opportunities.

1.1 Research Motivation

In recent past, there were several attempts to extent the application domains of machine learning to areas such as speech perception, visual object or pattern recognition, natural language understanding etc. The startling performance in these attempts are obliged to a recent advancement in machine learning paradigm called deep learning. This approach conceived an abstracted representation of data by embracing multiple hierarchical layers of feature / representation learning. Further, the deep learning approaches often offer the capability to utilize unlabelled data, which are plenty in nature, for initializing the network and feature extraction tasks. Deep learning approaches originated and have mainly been pursued in the area of neural networks. Even though deep neural network learning approaches are moving to greater heights by the invention of innovative combination of novel feature learning and traditional machine learning techniques may often realized locally optimal solutions due to the gradient based, non-convex optimization techniques used. These approaches seem to be effective only on problems having enormous amount of training data.

On the other hand kernel machines typically exploit convex optimization strategies which eliminate local optima and provide globally optimal solutions. Kernel machines also exhibit the potential to learn a complex decision boundary by transforming the data into the high dimensional feature space, with limited training samples. The kernel machines reduce the complexity of explicit mapping of every data samples into the feature space by exploiting kernel function(s) that facilitate the implicit computation of similarity between each pair of samples in the feature space. Kernel machines also manifest elegant properties like structural risk minimization and maximum marginal

classification. Support vector machine and its scalable counter part core vector machines, extreme learning machines, kernel based principal component analysis etc. are some of the popular kernel machines. Even though the kernel machines exhibit strong theoretical grounding, its shallow architecture resulting from the single layer kernel computation (feature extraction) limited its applicability to only problems with well defined data representations.

Recently a few attempts have been reported in the machine learning literature to impart deep learning capabilities to kernel machines. The recent uplift in this context is the invention of arc-cosine kernels to impart deep learning capabilities in kernel machines. This kernel enables multiple layers of non-linear transformation that mimics the computations in multi layer neural network. The capability of having different activation values (degree) in different layers of arc-cosine kernel offers qualitatively different geometric properties making it a suitable candidate for layered feature extraction. There are different avenues where the arc-cosine kernel proved its efficiency, particularly in conjunction with SVMs, called Deep Support Vector Machines (DSVM). However, the quadratic formulation of SVMs and multiple layers of computation in arc-cosine kernel impose a high computational cost and it restricts the application domains of DSVM[1]. This is identified as an interesting research avenue to explore the possibility of scaling up DSVM to large data problems. In this direction, the CVM, the scalable alternative for SVMs, is identified as a suitable candidate to build scalable deep kernel machines using arc-cosine kernels.

Multilayer Kernel Machine (MKM) introduced the process of multiple layers of feature extraction in kernel machines. It exploited unsupervised Kernel based Principal Component Analysis (KPCA) and arc-cosine kernel in its feature extraction layers. It encounters fixed kernel computations and scalability issues. Enhancing the MKM framework to overcome the fixed kernel computation and scalability issues is identified as an another potentially explorable opportunity.

The prominent learning approaches such as kernel machines particularly SVMs, and Neural Networks necessitate an iterative training procedure to adjust their learning

[1]DSVM : Support Vector Machines with arc-cosine kernel.

parameters. In machine learning literature, there was a strong belief that all the parameters in each layer of learning models need to be adjusted for an efficient generalization. However, Extreme Learning Machine (ELM) break this assumption and tend to be an effective learning model without any iterative parameter turning. ELM accomplish this by exploiting a simple generalized matrix inverse operations to compute the output weights and a random computation for its input weights and biases. ELMs also exhibit the potential to achieve the smallest norm of output weights in addition to minimizing the training error. The original formulation of ELM as a fast learning algorithm for Single Layer Feedforward Networks(SLFN) is then remodeled with universal approximation and classification capabilities. The unified learning method of ELM facilitates divergent form of feature mappings such as random feature mappings and kernel methods. The exploration towards the enhancement of shallow kernel ELM to build a deep kernel ELM is identified as another research avenue.
The above said facts motivated to formulates the research problem as detailed below.

Problem Statement

Explore the possibility of building kernel machines with deep learning characteristics. The potential dimensions explored are (i) Scalable deep learning in SVM like kernel machines. (ii) Scalable deep kernel machines with multiple layers of unsupervised feature extraction. (iii) Deep kernel learning in non-iterative learning approaches like extreme learning machines.

Based on the above problem statement, this thesis proposed three kernel machines with deep learning capabilities. The first exploration in this research “Deep Kernel Learning in Core Vector Machine” modeled a scalable deep kernel machines by combining arc-cosine kernel and Core Vector Machine. The Second contribution, “Deep Multiple Multilayer Kernel Learning in Core Vector Machines” was an attempt to bring out multilayer unsupervised feature extraction in scalable kernel machines by exploiting multiple kernel learning framework. Multiple kernel learning frame work in this model can combine both multilayer and single layer kernels. The third model, “Deep Kernel based Extreme Learning Machine” exploited multilayer arc-cosine kernel and fast,

non-iterative extreme learning machine algorithms to model a deep kernel based extreme learning machine. Experiments show that all the proposed methods consistently improve the generalization performances of the conventional shallow kernel machine approaches. The rest of the thesis is organized as detailed in next section.

1.2 Thesis Outline

Chapter 1 This chapter presents the background and motivations of research and provides the thesis outline.

Chapter 2 This chapter provides a comprehensive description on related methods and methodologies used in this research.

Chapter 3 The main focus of this chapter is on scalable deep learning in SVM like kernel machines. It describes deep kernel computation in core vector machines.

Chapter 4 The main theme of this chapter is deep kernel learning in non-iterative methods. It evaluate the feasibility of combining arc-cosine kernel and extreme learning machines.

Chapter 5 This chapter gives the details of building scalable deep kernel machines with multiple layers of unsupervised feature extraction.

Chapter 6 Conclusions and future works are mentioned in this chapter

All the papers published in various journals from the above works and references are presented at the end of thesis.

CHAPTER 2

Literature Survey

Core dream of Artificial Intelligence (AI), the most exciting branch of computer science, is to make the machines to think, to reason, to perceive, to speak, to communicate and to imitate many other human talents. AI has gone through several stirs of technical evolutions from first order logic to expert systems to the early waves of machine learning to today's deep learning revolution. Deep learning is one of the hottest discussion among the machine learning researchers which emphasized on both data representation and traditional classification/regression methods and producing state of the art results in many highly varying and complex pattern recognition tasks.

Shallow based learning models, prior to deep learning model, involve only one or two layers of hand-crafted feature extraction. These models do not seem to be good enough to discover the rich internal representation of data. These models were often fitted with supervised methods that enforces the requirement of large amount of labelled data. On the other hand, deep learning models are equipped with both supervised and unsupervised methods. These models can process on huge amount of data (both labelled and unlabelled) and counterbalance the extensive dependency of human intuition and prior knowledge which is needed to define feature representation in shallow models. In addition to combining both supervised and unsupervised learning paradigms, deep learning architectures also possess hierarchical representation of data, greedy layer wise training and many more desirable characteristics.

Deep learning concepts originates and has been mainly pursuing in the context of neural networks. Main challenge in modeling deep neural networks is its dependence on gradient based non-linear optimization techniques which are non-convex in nature. This non-convex optimization problems do not guarantee global optimization. On other hand, the kernel machines, particularly Support Vector Machines (SVMs), were attracted by many of the researchers because of its convex optimization and other interesting properties such as structural risk minimization and maximal marginal classification. However, the single layer kernel computation in SVMs are seemingly unequipped to discover the rich internal representations of data and it also seems to be good enough only in applications that involve comparatively small amount of data. It demands the requirement of both multilayer kernel computation and the scalability to cope with modern real world applications that involve huge amount of data. This thesis explored in this direction to enhance the kernel machines with scalability and multiple layers of feature extractions (kernel computations). In this context it worth to walk through the main concepts and method that we have adopted from various machine learning literatures. The purpose of this chapter is to provide a brief introduction to the research work conducted in the area of deep learning and kernel machines which have direct or indirect relevance in this research study. This literature survey begins with a brief introduction on deep learning in neural network followed by deep learning attempts in kernel machines. Rest of the chapter is organized as follows. Deep learning strategies in neural work are discussed in Section 2.1. The basic concepts of kernel machines and commonly used kernel machines are included in Section 2.2. Multiple kernel learning strategies are then discussed in Section 2.3. Deep kernel computation and different deep kernel learning architectures are given in Section 2.4.

2.1 Deep Learning

In the ever-changing ecosystem of machine intelligence, it may be often required to design new learning paradigms to keep up with highly complicated and varying modern real-world applications. Traditional shallow based machine learning algorithms work with predominantly hand-crafted representations of data. It often necessitates a lot of

human intervention and prior knowledge about the task being modeled. Moreover, the explored features may not be well suited for the learning task and it may de-escalate the generalization performance of the learning algorithms. Representation learning seems to be an efficient approach to explore learned representation which often result in much better performance when compared to hand crafted representation. However, the representation learning task does not address variability in the observed data, the factors of variation that explain the observed data. It requires more abstract, high level description of observed data. The hierarchical representation of data facilitates more abstract representation which involves high level representation of data in terms of simple low level representations. The abstract representation of data pave the way for identifying rich variability in observed data. The theoretical and biological arguments in the context of building more complicated real-world applications that involves natural signals such as human speech, sound, language, natural image, visual scenes etc. also suggest the necessity of multilayer data abstraction.

New trend in machine learning is to combine hierarchical data representation learning with traditional learning algorithms. One such trending learning paradigm that expedite the process of modeling highly complicated and varying modern real-world applications is 'deep learning'. Deep learning approaches attempt to model the multilayer learning capability of human brain which transforms high dimensional sensory data to abstract representation by passing through distinct layers of neurons in a hierarchical manner. Unlike shallow learning architectures that involve at most one layer of data representation, deep learning architectures encompass multiple processing layers to extract more abstract representation from large quantities of both labelled and unlabelled data. This recent entrant in machine learning builds complex concepts in terms of simple low level concepts in a hierarchical fashion. ' *Deep learning is a particular kind of representation learning that achieves great power and flexibility by learning to represent the raw data as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones*' [31]. The relationship between machine learning , representation learning and deep learning is clearly illustrated in Figure 2.1. In representation learning the handcrafted feature selection approach in machine learning is replaced with learned features. Deep

learning involve multiple layers (hierarchy)of such learned features from simple ones to more abstracted features.

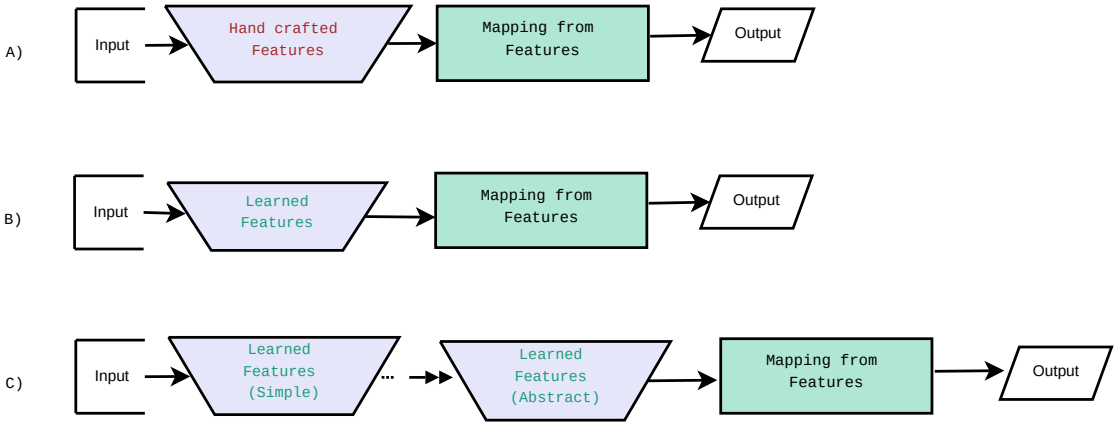

FIGURE 2.1: An architectural comparison between classical machine learning, representation learning and deep learning approaches. A) Traditional machine learning B) Representation learning C) Deep learning

In general, deep learning is a form of representation learning that attempts to build high-level abstractions from low-level descriptions of data through multiple non-linear transformations (representation) in a hierarchical manner [25, 32]. Each successive layer in this hierarchical model process the output from the previous layer. An algorithm is deep means the input is passed through a hierarchy of non-linear transformations. Each stage in that hierarchy is considered as trainable feature transformation. It has been clearly depicted in Figure 2.2.

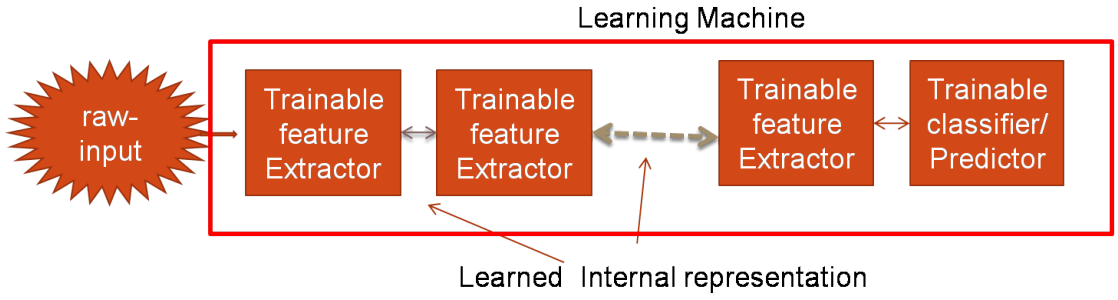

FIGURE 2.2: Block diagram illustrating the hierarchical feature learning in deep learning architectures

The motivating factor that favor deep learning is that it gives due importance in both data representation as well as traditional classification methods. Deep learning has been making tremendous waves at many highly complicated task domains like

speech perception, visual object recognition, pattern recognition, language understanding etc. The recent upturn in the development of general-purpose graphical processing units(GPUs), availability of immense amount of data and recent advancement in optimization techniques proliferate the popularity of deep learning. The capability to learn from both labelled and unlabelled data, which is plenty in nature, enable deep learning models to learn prior knowledge from input data itself and it also reduces human invention in a greater extent. Deep learning concepts originate and it has been mainly pursuing in the area of neural networks. Following sections walk through different milestone in the development of todays most exciting practitioner of machine learning, deep learning.

2.1.1 Historical Background

In around 1960, the famous researcher Frank Rosenblatt developed an algorithm based on how biological neurons learn from stimuli. This first generation artificial neural network so called 'perceptron' consists of one input layer, an output layer and a fixed set of hand crafted features [2]. A small random weights are applied to the inputs, and the resulting weighted sum of inputs passed to a function (threshold) that produces the output. Marvin Minsky and Seymor Papert highlighted various shortcomings of this model in their book 'Perceptrons: an introduction to computational geometry' and it dampened the interest on perceptron. They noticed the incapability of perceptron in learning non-linear functions like XOR.

Non-linearity in artificial neural networks was a long term research in machine learning community. It came to reality when Mr. Geoffrey Hinton collaborated with his colleagues David E. Rumelhart, & Ronald J. Williams and invented a new simple learning algorithm to train the neural network with many hidden layers [3]. It was considered as the second birth of artificial intelligence. Their back-propagation (BP) algorithm, the enabler of multilayer neural networks with the capability for no-linear computation, well addressed the limitations of single layer perceptron and endowed the neural networks with the ability to learn non-linear functions. This algorithm works by computing the model output in the forward pass and then back-propagating the

errors to update network weights by exploiting the derivatives of loss function. Such multilayer networks exhibit the potential to learn any function [54, 55]. This gradient based learning approach made tremendous change in the quality of machine learning and paved the way for building many real-world applications. However, BP algorithm struggled to find good models having reasonable generalization performance in neural networks with more than a few number of hidden layers. Real world learning problem often involves non-convex objective functions and it get often trapped into local optima.

A resurgence in neural networks (Deep learning) began in around 2006 when Geoffrey Hinton proposed the idea of unsupervised pre-training and a novel class of generative learning models called deep belief nets (DBN) [33, 56]. In the same time, other pioneers in deep learning research, Yoshua Bengio and Ranzato introduced non generative, non probabilistic, unsupervised learning models such as stacked auto-encoders [34] and an energy-based model for training sparse representation of data [35]. These two models also used a greedy-layer wise training similar to DBN. These three models defined the pillars of modern multilayer neural network popularly called as deep learning. A picturization of the interesting journey from earlier electronics brain to todays deep neural networks is shown in Figure 2.3 [57].

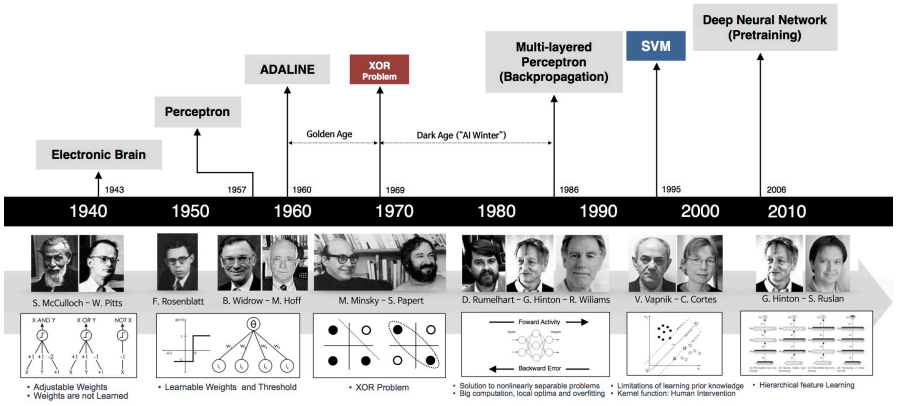

FIGURE 2.3: Milestones in the journey of deep learning

2.1.2 Deep Learning Architectures

In machine learning literature, the term deep learning is popularly used to refer a wide range of learning algorithms and architectures involving multiple layers of hierarchical

data representation. Most of the works in this context are broadly classified into three categories : Generative deep architectures, Discriminative deep architectures and Hybrid deep architectures [30]. Generative deep models focus on unsupervised learning to capture abstract, high-level description of the data (representation learning). Discriminative models are intended to learn classifiers using labelled data by capturing the posterior distributions of classes. Hybrid deep architectures constitute the combination of both generative and discriminative deep learning architectures. The following section discuss three interesting deep learning architectures viz; Deep Belief Network(DBN), Convolutional Neural Network(CNN) and Recurrent Neural Network(RNN).

Deep Belief Network : The era of deep learning architectures started when Mr.Geoffrey Hinton, explored a new multilayer neural network called a Deep Belief Network (DBN)[33]. Emergence of DBN as a stack of Restricted Boltzmann Machines (RBM) contributed an effective way of optimizing network weights in a multilayer neural network with a greedy, layer wise approach . Each layer-wise unsupervised pre-training utilizes a single-layer representation learning models such as RBM. It exploits a contrastive divergence approximation of the log-likelihood gradient to train every layer of DBN. This approach often achieves a time complexity linear to depth and size of networks. The inputs to the network are fed into the first layer RBM and the final abstracted representation is extracted from the hidden layer of the last RBM. The deep learning models built with DBN configured pre-training followed by a traditional neural network (with back-propagation) outperformed Multi Layer Perceptron (MLP) with random initialization [58, 40]. This model exhibited the potential to utilize unlabelled data in its pre-training process. This pre-training process effectively alleviated the under-fitting and over-fitting problems which are common in deep neural networks.

An alternative application of layer-wise greedy unsupervised pre-training principle, on auto-encoder instead of RBM, was introduced by Yoshua Bengio [34]. Auto-encoders attempts to reconstruct the input in the output layer from the encoded intermediate representations in the hidden layer. Thus, target output tends to be the input itself. It often exploited the transpose of input-hidden layer weight matrix as

hidden-output layer weight matrix. Their paper also utilized a simple fix based on partial supervision that achieves better improvements. It was an attempt to enhance DBNs to handle continuous-valued inputs. In another paper [29] Hinton suggested the combination of three great ideas for effectively building multiple layers of representations. Layer wise representation learning by exploiting RBM like machine was his next idea. It helps to decompose complex learning task into multiple simpler procedures and to eliminate the inference problems. His third thought was the use of separate fine tuning mechanism to improve the generalization performance of the composite learning model.

Convolutional Neural Networks : Other prominent deep learning architecture which has been widely used in image processing is Convolutional Neural Networks (CNN)[59]. This discriminative deep architecture comprises a stack of convolutional layers and pooling layers. CNN proposed three ideas such as shared weights, local receptive fields and sub-sampling, to handle shift and distortion invariance commonly encountered in image processing tasks. The local receptive fields facilitates the neurons to identify the visual features like oriented edges, corners and so on. The variation in silent features can be made by the distortion or shift of inputs. The set of neurons corresponding to the receptive fields at different locations of the image often share identical weights. The convolution process involves a digital filter which is convoluted with a local receptive field in the image (in first layer there after feature map) and then a bias value is added. This process is then extend through the entire portion of input, in horizontal and vertical direction. Any number of filters can be used for convolution and the stacked output of all those filters contributes the convolution layer and delivers divergent feature maps. Each learned filters can capture different aspects of an image [60]. It is a common practice to include pooling layers after convolution, to constitute another level of translation invariance. The pooling layer performs local averaging and sub-sampling, to reduce the resolution of feature map. It causes the reduction in responsiveness of output against shifts and distortions. Pooling facilitates reduction in spatial size of the representation which in effect minimize the risk of over-fitting by reducing the number of parameters. At the end of deep network, this layered structure is attached to a fully-connected layers. The CNN model is depicted in Figure 2.4. CNN has been

proved to solve many real world problems involving image recognition and computer vision tasks [61, 62].

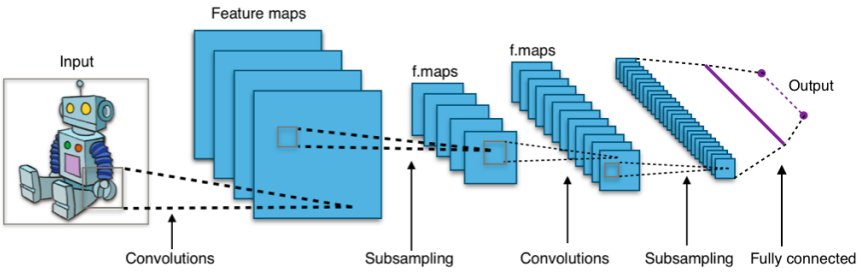

FIGURE 2.4: A typical architecture of Convolutional Neural Network

Recurrent Neural Networks: It is an interesting deep learning architecture intended to model learning problems involving sequential data [38]. In contrast to traditional feedforward network, the cyclic connection involved in RNN enables them as a powerful choice to model sequential problems. The beauty of RNNs lies in its memory which remember the information about previous inputs they have received and thus the potential to maintain long-term dependencies in sequential data. RNN uses same parameters across all the timestamps, which resembles as the same task is repeated in a layered fashion, hence the name recurrent. Like other deep learning architectures, RNNs also used a variant of backpropagation algorithm called Backpropagation Through Time (BPTT). It back-propagated the errors from last to first timestamps as unfolded through layers positioned in a temporal fashion. A basic model of RNN has been shown in Figure 2.5. In this figure U, V, W represent input to hidden state, hidden state to hidden states

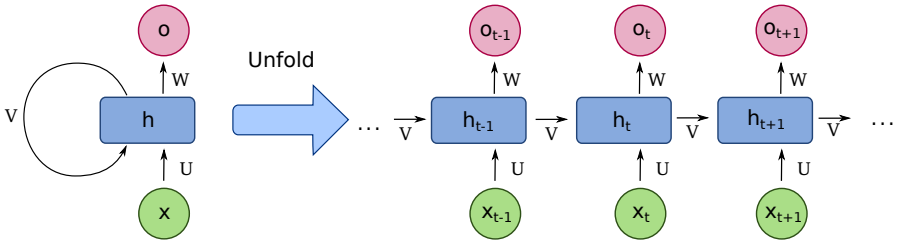

FIGURE 2.5: A Symbolic modeling of RNN and its unfolded representation

and hidden states to output weight matrices. Now, the hidden states and output can be

computed with hidden activation function $\mathcal{F}_h$ and output activation function $\mathcal{F}_\mathcal{O}$ as :

$$\begin{aligned} h_t &= \mathcal{F}_h(Ux_t + Vh_{t-1}) \\ O_t &= \mathcal{F}_\mathcal{O}(Wh_t) \end{aligned} \tag{2.1}$$

The basic RNN models encounter the well-known gradient vanishing problem while training to learn long term patterns. The invention of Long Short-Term Memory (LSTM) and Gated Recurrent Unit(GRU) addresses gradient vanishing problems in some extends. The following paragraphs discuss LSTM and GRU in detail. The remembering capability of RNNs have been extended to a long period of time by exploiting Long Short-Term Memory (LSTM) networks, RNN layers are build upon LSTM units [63]. LSTM exhibit a gated cell approach which has the potential to learn which information and how long it to be remembered. It has been achieved through three gates named as forget, input and output gates. The gates are formulated with sigmoid activation and point wise multiplication operations. The architecture of RNN with a single LSTM unit has been shown in Figure 2.6. In this figure F_t, I_t, O_t represents the

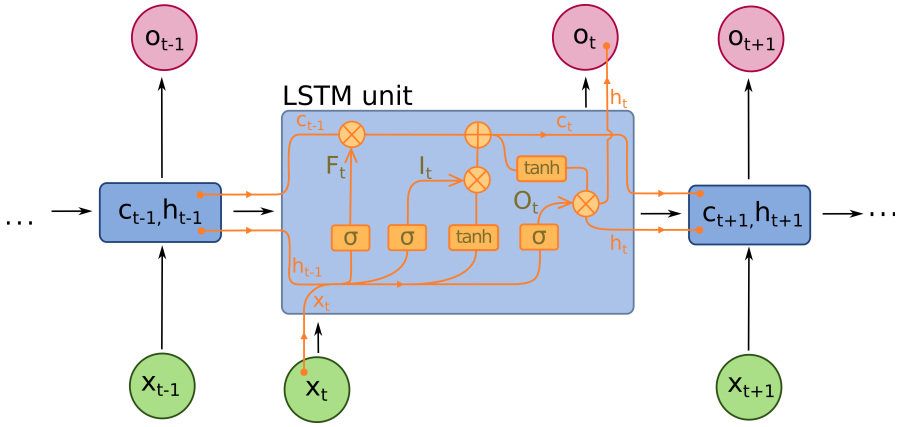

FIGURE 2.6: Architecture of Recurrent Neural Network with Long-Short Term Memory

forget gate, input gate and output gate respectively. Forget gate determines the information to be get rid of the cell state and the output gate is responsible for the portion of information delivered from the cell sate. Updating new information into the cell state is carried out by determining the portion of new information to be updated (input gate layer) and then updating the cell state with filtered information. These operations can be summarized as follows.

$F_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$	*Forget gate - return a value between 0 and 1*
$F_t * C_{t-1}$	*Forgetting the decided information*
$I_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$	*Input gate determines the information to be updated*
$\hat{C}_t = \tanh(W_c.[h_{t-1}, x_t] + b_c)$	*The candiate vector to be added to the states*
$C_t = F_t * C_{t-1} + I_t * \hat{C}_t$	*Updating the state with new value after forgetting from previous states*
$O_t = \sigma(W_o.[h_{t-1}, x_t] + b_0)$	*Output gate decided the portion of information to be outputed*
$h_t = O_t * \tanh(C_t)$	*part of information outputed from LSTM*

Another contribution towards long-term dependency problem in RNN is Gated Recurrent Unit(GRU) [64, 27]. It combines forget gate and input gate in LSTM as update gate and also involves a reset gate. This architecture is further simplified by combining cell state and hidden state. Operations in GRU can be summarized as follows.

$$p_t = \sigma(W_p x_t + U_p h_{t-1} + b_p) \quad \textit{Update gate}$$

$$q_t = \tanh(W_q x_t + U_q h_{t-1} + b_q) \quad \textit{Reset gate}$$

$$h_t = (1 - p_t) \circ h_{t-1} + p_t \circ \sigma_h(W_h x_t + U_h(q_t \circ h_{t-1}) + b_h)$$

These enhancements resolve gradient vanishing problem and other optimization bottlenecks in some extends and makes RNN more popular. The high dimensional hidden state and capability to learn and remember previous informations over a sequence makes RNN a good candidate in many real world applications [65, 66, 67]. Other appealing deep layered neural networks and its main achievements over the last decade are listed in Table 2.1 and Table 2.2 respectively.

TABLE 2.1: A summary of popular deep learning architectures

Year	Author(s)	Description of work
1986	Michael I. Jordan	Explored Recurrent Neural Networks, where connections between the neural nodes for directed cycles. These models are capable for handling sequential information.
1990	Yann LeCun	Explored LeNet, expressed the possibility of deep neural networks in practical applications.
1997	Schuster and Paliwal	Explored Bidirectional Recurrent Neural Networks, where output depends on both the previous and next elements in the sequence.
1997	Hochreiter and Schmidhuber	Explored Long Short Term Memory networks and resolved vanishing gradient complications.
2006	Geoffrey Hinton	Explored Deep Belief Networks and layered pretraining approach, which opened the present deep learning era.
2009	Salakhutdinov and Hinton	Explored Deep Boltzmann Machines, where hidden units are arranged in a deeply layered pattern. This neural network model exhibits connection only in the adjacent layers and lacks hidden-hidden or visible-visible connections within the same layer.
2012	Geoffrey Hinton	Explored Dropout, an effective mechanism for training deep neural networks.
2014	Ian Goodfellow	Explored Generative Adversarial Networks, which has the capability to mimic any data distribution.

TABLE 2.2: Main achievements of deep neural network learning

Year	Autors	Description
2011	John Markoff	Watson : A question - answering model from IBM beats humans in a Jeopardy! competition by exploiting natural language processing and information retrieval techniques.
2012	Andrew Y. Ng et.al.	Google Brain , the machine learning group lead by Andrew Ng, identifies cats from the unlabelled image frames of videos.
2014	Yaniv Taigman et.al	Deep face: A remarkable achievement from Facebook which learn the neural work to identifies faces. This model yielded an accuracy of 97.35% which is more than 27% accuracy over its immediate predecessor.
	Alex Woodie	SIbly: Another initiative from Google, provides a platform for massive parallel learning. It facilitates human behaviour prediction and recommendations in a greater extent.
2016	David Silver et.al.	AlphaGo: Yet another Google's innovation the first Computer Go program to beat an unhandicapped professional human player. It exploited tree search techniques in machine learning approach.
2017	Alex Woodie	AlphaGo Zero & AlphaZero : Improved versions of AlphaGo which generalized to Chess and more two-player games.

From the above analysis, the noticeable characteristics and issues of deep neural network learning approaches are listed as follows:

- Deep learning approaches give due respect in both data representation and classification or prediction tasks.
- Multiple layers of feature extraction reduces the risk of feature engineering.
- Greedy, layer-wise training method reduces complexity of training.
- These members have best-in-class performance on problems involving unstructured media like text, sound and images.
- They exhibit the capability to handles both labelled and unlabelled data effectively.
- Deep learning approaches have the potential to combine both supervised and unsupervised learning paradigms.
- Non-convexity in optimization problem involved in the learning process of these architectures do not guarantee globally optimal solutions.
- In these models, the structural complexity required for solving any given problem is heuristically decided and empirically verified - thus do not support the notion of structural risk minimization.

2.2 Kernel Machines

In machine learning literature, linear models are seem to be effective in learning tasks that involve linear decision boundaries which are rare in most of the real world applications. As per Cover's theorem, problems with non-linear decision boundaries can be addressed by transforming data samples into a high-dimensional space (feature space) where they may become linearly separable [68]. This transformation facilitates linear operations in higher dimensional spaces which are equivalent to non-linear functions in input space. Let the original input vector space is $\Re^d$ and $\mathcal{F}$ be the high dimensional feature space. Then the non-linear mapping ϕ can be expressed as :

$$\phi : \Re^d \mapsto \mathcal{F}$$

For the given training set $T = \{(x_i, y_i)\}_{i=1}^{N}$ where $x_i \in \Re^d, y_i \in \Re$, the nonlinear mapping of data samples into the feature space $\mathcal{F}$ can be expressed as :

$$\phi : x \in \Re^d \mapsto \phi(x) \in \mathcal{F},\ \mathcal{F} \in \Re^D$$

The data sample in the feature space then takes the form $(\phi(x_i), y_i)$. The class labels y_i remain unchanged in the feature space. It ease the process of finding a linear decision boundary in the high dimensional feature space $\mathcal{F}$ that separate the data samples $((\phi(x_1), y_1), ..(\phi(x_i), y_i)..(\phi(x_N), y_N))$. It is clearly depicted in Figure 2.7. However, it

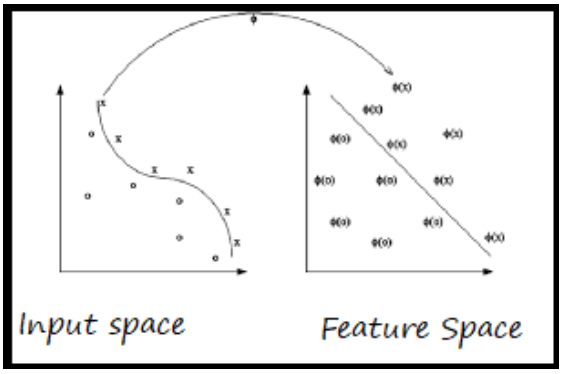

FIGURE 2.7: An illustration of non-linear transformation to the high dimensional feature space

seems to be highly expensive to apply a non-linear function on every instance of data sample and explicitly transform them into the high dimensional feature space.

An effective approach in this context is the implicit computation of similarity between training samples in feature space without explicitly projecting them onto the feature space. It can be accomplished by a special mechanism called kernel trick which involves kernel function. Kernel functions possess the potentiality to handle non-linearity problems by implicitly mapping data in the input space - where data is linearly inseparable, to a new high-dimensional space - where data is linearly separable. A kernel facilitates the computation of inner product between all the samples x_i, x_j in the feature space as a direct function of the data samples in the original space, without explicitly applying non-linear mapping ϕ on every data sample [9]. A kernel function $\mathcal{K} : d \times d \mapsto \Re$ is perceived as an inner products between data samples mapped in the

feature space through a non-linear mapping, ϕ and it can be expressed as :

$$\mathcal{K}(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_F$$

This kernel trick facilitates the implicit no-linear transformations in many applications that can be modeled in terms of inner product between the data samples. Any machine learning algorithm rely on the inner product between the instances can be transformed into kernel machines (kernalized) by replacing all instances of $< x_i, x_j >$ with an appropriate reproducing kernel function $k(x_i, x_j)$ [10]. In the dual representation of optimization problem, the replacement of inner product with an appropriate kernel function facilitates the non-linear transformation to the higher dimensional feature space without increasing the tunable parameters. Kernel representation increases the computational power of linear machines by projecting data into high dimensional feature space. The learning models that leverage this kernel trick is popularly cited as 'kernel machines'. It can be exemplified with a kernel function $\mathcal{K}(x_i, x_j) = \langle x_i^T x_j \rangle^2$ as follows. Let $\{(x_i, y_i)\}_{i=1}^N, x \in \Re^2$ be the dataset. Then

$$\begin{aligned}
\mathcal{K}(x_i, x_j) &= \mathcal{K}\left(\begin{pmatrix} x_i^1 \\ x_i^2 \end{pmatrix} \begin{pmatrix} x_j^1 \\ x_j^2 \end{pmatrix}\right) \\
&= \left\langle \begin{pmatrix} x_i^1 \\ x_i^2 \end{pmatrix} \begin{pmatrix} x_j^1 \\ x_j^2 \end{pmatrix} \right\rangle^2 \\
&= \left(x_i^1 x_j^1 + x_i^2 x_j^2\right)^2 \\
&= \left(x_i^1 x_j^1\right)^2 + \left(x_i^2 x_j^2\right)^2 + 2 x_i^1 x_j^1 x_i^2 x_j^2 \\
&= \underbrace{\left((x_i^1)^2 \sqrt{2} x_i^1 x_i^2)(x_i^2)^2\right)^T}_{\phi(x_i)} \underbrace{\begin{pmatrix} (x_j^1)^2 \\ \sqrt{2} x_j^1 x_j^2 \\ (x_j^2)^2 \end{pmatrix}}_{\phi(x_j)}
\end{aligned}$$

Commonly used kernel functions are listed in table 2.3.

TABLE 2.3: Commonly used kernel functions

Kernel	Mathematical Expression	Description
Linear Kernel	$k(x_1, x_2) = x_1^T x_2 + C$	It is simply the inner product between the training samples with an additional constant.
Polynomial Kernel	$k(x_1, x_2) = (\alpha x_1^T x_2 + C)^d$	A non-stationary kernel which is applicable only on normalized training samples.
Gaussian Kernel	$k(x_1, x_2) = EXP\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$	It is a well known example of Radial Basis Function (RBF) kernel. The performance of this kernel is highly influenced by the turning of its parameter σ. If σ is over estimated, the non-linear power will be diminished and its underestimation will badly affect the regularization
Laplacian Kernel	$k(x_1, x_2) = EXP\left(-\frac{\|x_1 - x_2\|}{\sigma}\right)$	Laplacian kernel is equivalent to Gaussian kernel. However it is very less sensitive to the parameter σ
Sigmoid Kernel	$k(x_1, x_2) = tanh\left(\alpha x_1^T x_2 + C\right)$	It comes from neural networks. It is also known as multi-layer perceptron kernel
Circular Kernel	$k(x_1, x_2) = \frac{2}{\pi} arcco\left(-\frac{\|x_1 - x_2\|}{\sigma}\right) - \frac{\|x_1 - x_2\|}{\sigma}\sqrt{1 - \left(\frac{\|x_1 - x_2\|}{\sigma}\right)^2}$	An isotropic, positive definite kernel in R^2. commonly used in geostatic applications
Spherical Kernel	$k(x_1, x_2) = 1 - \frac{3}{2}\frac{\|x_1 - x_2\|}{\sigma} + \frac{1}{2}\left(\frac{\|x_1 - x_2\|}{\sigma}\right)^3$	Same as that of circular kernel but positive definite in R^3
Chi-Square kernel	$k(x_1, x_2) = 1 - \sum_{i=1}^{n} \frac{(x_{1i} - x_{2i})}{\frac{1}{2}(x_{1i} - x_{2i})}$	Inspired from Chi-Square distribution

Kernel function ($\mathcal{K}$) computes an $N \times N$ matrix representing the similarity between all the pairs of data samples in the feature space. This kernel matrix is also known as Gram matrix and it can be represented as :

$$K\,[i,j] = \langle \phi(x_i), \phi(x_j) \rangle \qquad \forall_{i,j}$$

According to Mercer theorem, a kernel ($\mathcal{K}$) to be a valid kernel, it is necessary and sufficient that the corresponding kernel matrix K must be symmetric ($K = K^T$) and positive-semi definite ($CKC^T \geq 0, \quad \forall_C$).

Re-engineering is another interesting capability of kernel functions. It promote the construction of complex kernels by combining relatively simple kernels. Some commonly used approaches to re-engineer the kernels are shown in Table 2.4. Unlike neural

TABLE 2.4: Basic rules for kernel re-engineering

Rule	**Expression**
Linear Combination	$k(x_i, x_j) = \sum_k \beta_k K_k(x_i, x_j)$
Positive Offset 2	$k(x_i, x_j) = k_1(x_i, x_j) + c$
Product	$k(x_i, x_j) = \prod_k k_k(x_i, x_j)$
Exponential	$k(x_i, x_j) = \exp(\frac{1}{\sigma^2} k_k(x_i, x_j))$
Normalization	$k(x_i, x_j) = \frac{k_k(x_i, x_j)}{\sqrt{k_k(x_i, x_i) k_k(x_j, x_j)}}$

networks, the kernel machines exploit convex optimization strategy which eliminates local minimum and guarantees global optimization [69]. The modularity of kernel functions facilitate the independent algorithmic design and analysis from kernel functions [70]. These elegant features and very strong mathematical slants makes the kernel machines rather popular in machine learning literature. The interesting features and issues of kernel machines can be summarized as follows.

- Independent design of data representation and learning algorithms.
- Guaranteed global optimization through its convex optimization.
- Kernel function can be used in diverse forms of data such as sequences, vectors, sets, graphs, etc. (versatility)
- Applicable in both supervised and unsupervised scenarios.

- Learning capacity can be controlled by regularization function
- Strong theoretical frameworks.
- Some of the kernel machines employ structural risk minimization approach
- Selection of an inappropriate kernel function causes the degradation of generalization performance of the learning machine.

Following subsections reviews some of the popular learning models that exploit kernel trick for pattern analysis and classification.

2.2.1 Support Vector Machines

A support vector machine (SVM) is a well-known kernel machine, initially conceived of by Cortes and Vapnik [11]. This discriminative classifier is conceptualized by exploiting Maximum Marginal Hyperplane (MMH). A separating hyperplane that constitutes maximum margin from any training observations is the Maximum Marginal Hyperplane(MMH). The margin between each class and the hyperplane involves distance computation between hyperplane and the support vectors, the data points close to hyperplane, of the respective class. The concepts of hyperplane, support vectors and margin are clearly depicted in Figure 2.8. Initially, SVM was formulated as a supervised

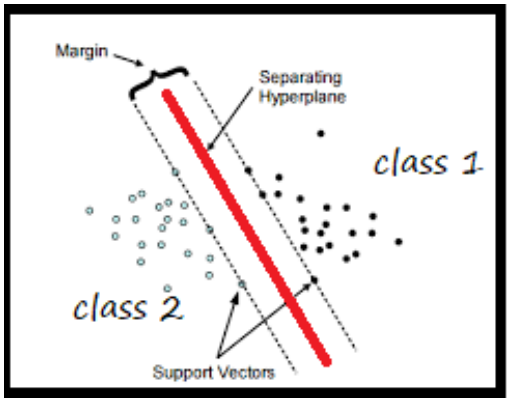

FIGURE 2.8: A geometric illustration of Hyperplane, margin and support vectors

linear classifier on binary class problems. Later it has been enhanced for regression [46], multi-class classification [71], etc. The non-linearity in support vector machines have

been accomplished by exploiting kernel trick [5, 6, 7, 8, 9, 10]. Support vector machines also exhibit the capability to tackle unsupervised scenarios [72].

Generally, SVMs are formulated as quadratic programing problem. SVMs attempt to minimizing an upper bound of the generalization error through maximizing the margin between the separating hyperplane and the data. The formulation of well structured optimization problem of SVM can be detailed as follows [11, 12, 4, 13, 7].

SVM formulation : Let $f_{<w,b>}(x) = w.x + b$ be the candidate hyperplane on dataset $\{(x_i, y_i)\}_{i=1}^{N}$ where $x_i \in \Re^d$ and $y_i \in \{-1, +1\}$. SVMs tend to obtain the optimal solution by maximizing the margin between the candidate hyperplane and canonical hyperplanes ($f_{<w,b>}(x) = +1$ and $f_{<w,b>}(x) = -1$), the hyperplanes passed through support vectors of both the classes. By realizing a functional margin of 1 to the candidate hyperplane $f(w, b) = 0$, the training samples belongs to positive and negative classes can be described as:

$$x_i w + b \geq +1 \qquad \text{for y} = +1$$

$$x_i w + b \leq -1 \qquad \text{for y} = \text{-1}$$

These two hyperplanes can be combined as :

$$y_i(x_i w + b \geq -1), \forall_i$$

It has been clearly shown in Figure 2.9. It has been observed that the geometric margin between the canonical hyperplane is $\frac{2}{||w||}$ and the quadratic programming optimization problem of SVM (primal form) can be expressed as :

$$\underset{w,b}{min} \frac{1}{2} \|w\|^2 \quad \text{s.t.} \quad y_i(< w, x_i + b) \geq 1, \quad \forall i \tag{2.2}$$

The constrains can be cater into minimization problem by exploiting Lagrange multipliers α.

$$\begin{aligned} L(w, b) &= \frac{1}{2} \|w\| - \sum_{i=1}^{N} \alpha_i [y_i(x.w + b - 1)] \\ &= \frac{1}{2}(w.w) - \sum_{i=1}^{N} \alpha_i y_i (w.x_i) - \sum_{i=1}^{N} \alpha_i y_i b + \sum_{i=1}^{N} \alpha_i \end{aligned} \tag{2.3}$$

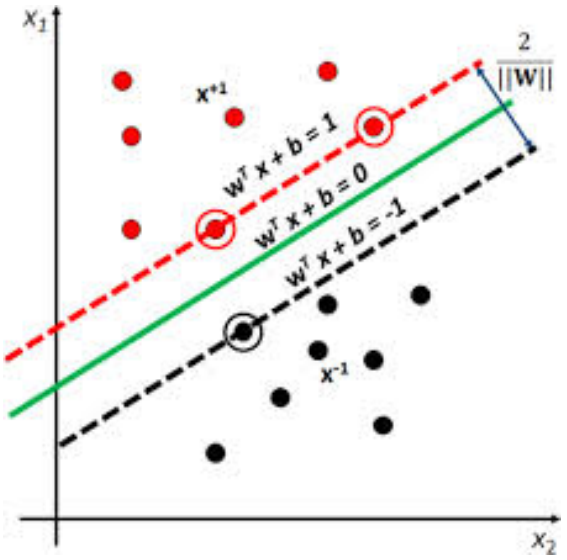

FIGURE 2.9: A geometric illustration of candidate hyperplane, canonical hyperplanes and margin

Equating the derivatives of Equation 2.3 w.r.t w, b to zero derives $w = \sum_{i=1}^{N} \alpha_i y_i x_i$ and eliminate $\sum_{i=1}^{N} \alpha_i y_i b$ respectively. Substituting these factors in Equation 2.3 offers a new formulation which depends on Lagrange multipliers α. This Wolfe dual form of SVM optimization problem can be expressed as:

$$\begin{aligned} L(\alpha) = & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i y_i \alpha_j y_j (x_i . x_j) \\ & \text{subject to} \quad \alpha_i \geq 0, \forall_i \quad \textit{and} \quad \sum_{i=1}^{N} \alpha_i y_i = 0. \end{aligned} \tag{2.4}$$

By considering the matrix $M = \sum_{i,j} y_i y_j (x_i \cdot x_j)$, the above expression can be written as:

$$\begin{aligned} L(\alpha) = & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{M} \boldsymbol{\alpha} \\ & \text{subject to} \quad \alpha_i \geq 0, \forall_i \quad \textit{and} \quad \sum_{i=1}^{N} \alpha_i y_i = 0. \end{aligned} \tag{2.5}$$

This linear formulation of SVMs involve only dot product between the training samples. This observation alleviated the process of building non-linear SVM, by replacing the inner product between the data samples with an appropriate kernel function $k(x_i, x_j)$ which compute the inner product $< \phi(x_i), \phi(x_j) >$ in the feature space. Now, the

non-linear SVM can be expressed as [8] :

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i y_i \alpha_j y_j k(x_i.x_j) \tag{2.6}$$

For the past few years, there have been significant developments in the theoretical understanding of SVM and its application in real world problems such as text categorization [73], face detection [74], generic object categorization [75] etc. SVM methods have also been addressed the problems involving structured outputs like tree, sets and sequences [76]. The structured prediction is commonly used in natural language processing. Support vector machines exhibit better generalization performance from limited data samples by exploiting the structural risk minimization principle [4, 6].

2.2.2 Core Vector Machines

The classical SVM formulation, both binary [11, 77] and multi-class [78], involves Quadratic programing (QP). In this formulation, the training process enforces a time complexity in the order of $O(n^3)$[1] and a space complexity of $O(n^2)$, for 'n' training samples. It does not seems to be a feasible solution for many real world applications that involve very large size datasets. Many attempts have been made to solve this scalability problem in kernel machines, particularly in support vector machines. The recent development in this context, the Core Vector Machine (CVM), exhibit the potential to scale up support vector machines. The main characteristics of CVM is the reformulation of QP problem of SVM as an equivalent Minimum Enclosing Ball (MEB) problem. It is then solved by a fast $(1+\epsilon)$ algorithm by utilizing the concepts of core sets. The minimum enclosing ball formulation of SVM and its approximation using core set is further explained as in the following paragraphs.

Minimum Enclosing Ball Problem : Let $A = \{a_1, a_2, ..a_N\}, \quad a_i \in \Re^d$ be the finite set of data points in a d dimensional space. The minimum enclosing ball on A ($MEB(A)$) represents the smallest unique ball that encloses all the vectors in A. It can be expressed

[1]Solving the quadratic problem and choosing the support vectors involves the order of n^2 , while solving the quadratic problem directly involves inverting the kernel matrix, which has complexity on the order of n^3

with center $c \in \Re^d$ and radius $r \in \Re$ as $B_A(c, r)$, which computes the minimum radius of a ball that covers all the data points in A. The primal and its equivalent dual form of minimum enclosing ball scenario can be expressed in the following two equations[79] :

$$(c, r) = \min_{c,r} \; r^2 : \left\|c - a^i\right\|^2 \leq r^2; \qquad \forall i \tag{2.7}$$

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{N} \alpha_i \left\langle a_i, a_i \right\rangle - \sum_{i,j=1}^{N} \alpha_i \alpha_j \left\langle a_i, a_j \right\rangle \\ &s.t. \quad \sum_{i=1}^{N} \alpha_i = 1, \quad \alpha_i \geq 0; \quad \forall i \end{aligned} \tag{2.8}$$

Any quadratic programming problem of this form can be considered as MEB problem. Traditional approaches to solve minimum enclosing ball problem [80, 81, 82] suffers from scalability, depends on both dimensionality d and size N of data points. A well-known $(1 + \epsilon)$ approximation algorithm [83, 84, 85, 86] has the capability for solving this minimum enclosing ball scenario. This algorithm is accomplished on a subset of data points called core set. Core set is nothing but a subset of dataset on which the optimization problem can be applied to accomplish a good approximation to the original inputs.

For the given $\epsilon \geq 0$, the $(1 + \epsilon)$ approximation of the ball $B_A(c, r)$ can be expressed as $B_A(c, (1 + \epsilon)R)$, where $R \leq r$ and $A \subseteq B_A(c, (1 + \epsilon)R)$. The data points enclosed by $B_A(c, R)$, a subset of A, is called the core set of data points A. More formally, a subset $S \subset A$ signifies as core set of A if and only if the expansion of MEB(S) by a multiplicative factor of $(1+\epsilon)$ covers all the points in A. It has been clearly depicted in Figure 2.10. An approximation strategy proposed in [84] which employed an iterative method to achieve $(1 + \epsilon)$ optimal(approximate) solution. This scheme iteratively expands the current ball,$B_A(c_i, r_i)$, by including the furthest point away from the ball $B_A(c_i, (1+\epsilon)r_i)$, until $B_A(c_i, (1+\epsilon)r_i)$ covers all the point in A. The distinctive characteristics of this strategy is that the number of iteration and the size of core set depends only on multiplicative factor ϵ, not on dimensionality and size of dataset. This remarkable advantage of $(1+\epsilon)$ algorithm facilitates its applications on kernel methods.

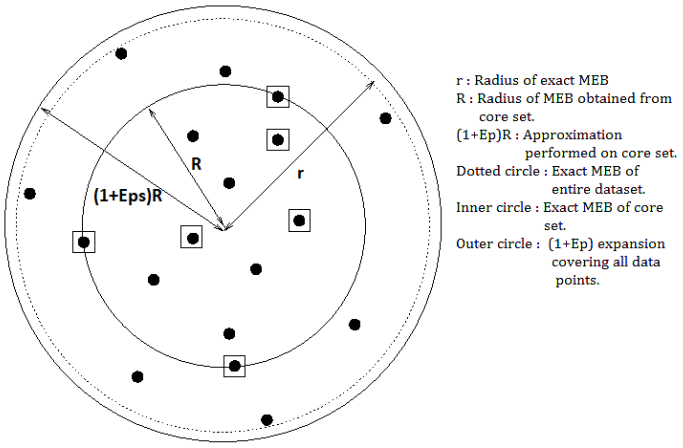

FIGURE 2.10: A geometric illustration of (Approximate) Minimum Enclosing Ball problem

Kernel induced MEB problem: The minimum enclosing ball (MEB) problem exhibit close relation to techniques such as hard-margin support vector data description [87, 88, 86] (SVDD), binary class and unary class support vector machine paradigms. Let $B_A(c, r)$ be the MEB in the kernel induced feature space on dataset $A = \{(x_i, y_i)\}_{i=1}^{N}$. Now the primal problem in SVDD can be expressed as [47]:

$$(c, r) = \min_{c,r} \quad r^2 : \|c - \phi(x_i)\|^2 \leq r^2; \qquad \forall i \tag{2.9}$$

where ϕ symbolize the feature map. The equivalent dual problem with an appropriate kernel $k(x_i, x_j) =< \phi(x_i), \phi(x_j) >$ is :

$$\begin{aligned} &\max_{\alpha_i} \sum_{i=1}^{N} \alpha_i k_{ii} - \sum_{i,j=1}^{N} \alpha_i \alpha_j k_{ij} \\ &s.t. \quad \alpha_i \geq 0; \text{ for i= 1 .. N}, \quad \sum_{i=1}^{N} \alpha_i = 1 \end{aligned} \tag{2.10}$$

The matrix equivalent of 2.10, as expressed in [47], is :

$$\max_{\alpha} \quad \alpha^{'} diag(K) - \alpha^{'} K \alpha \quad \text{s.t} \quad \alpha \geq O, \quad \alpha^{'} I = 1 \tag{2.11}$$

Here O, I, α represents vectors of zeros, ones and Lagrange multipliers respectively. Equation 2.10 is further simplified in the situation where $k_{ii} = z$, a constant as :

$$\begin{aligned} &\max_{\alpha_i} -\sum_{i,j=1}^{N} \alpha_i \alpha_j k_{ij} \\ &s.t. \quad \alpha_i \geq 0; \text{ for i= 1 .. N}, \quad \sum_{i=1}^{N} \alpha_i = 1 \end{aligned} \tag{2.12}$$

It is observed that, the isotropic kernel[2] (e.g., Gaussian kernel), dot product kernel - $k(x, y) = k_l(x^{'} y)$ (e.g.,polynomial kernel with normalized inputs;) and all the normalized kernel satisfies the condition $k_{ii} = z$, a constant. Any kernel machine that involves kernel function fulfilling the condition $k_{ii} = z$, a constant can be re-formulated as an equivalent MEB problem [79].

SVM as MEB problem: Consider the training set of input-output pairs $\{(x_i, y_i)\}_{i=1}^{N}$ where $x_i \in \Re^d$ and $y_i \in \{-1, +1\}$. Now the two class SVM can be reformulated as an MEB problem and its Wolfe dual form can be expressed as [89, 90, 79]:

$$\begin{aligned} &\min_{\alpha} \sum_{i,j=1}^{N} \alpha_i \alpha_j (y_i y_j k(x_i, x_j) + y_i y_j + \frac{\delta_{ij}}{C}) \\ &s.t. \sum_{i=1}^{N} \alpha_i = 1 \quad , \alpha_i \geq 0 \quad \forall i \end{aligned} \tag{2.13}$$

where α is Lagrangian multiplier, δ_{ij} is the kronecker delta function[3] and $C > 0$ is an user defined parameter that controls the trade-off between the margin and training error. By using the transformed kernel $\hat{k}_{ij} = y_i y_j (k_{ij} + 1) + \frac{\delta_{ij}}{C}$, the above equation is then reformulated as:

$$\min_{\alpha} \sum_{i,j=1}^{N} \alpha_i \alpha_j \hat{k}_{ij} \quad s.t. \sum_{i=1}^{N} \alpha_i = 1 \quad \alpha_i \geq 0 \quad \forall i \tag{2.14}$$

[2]When a kernel depends only on the norm of the lag vector between two examples, and not on the direction, then the kernel is said to be isotropic: $k(x, z) = k_l(\|x - y\|)$

[3]$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

Whenever the condition $k_{ii} = z$, a constant is satisfied, then the transformed kernel $\hat{k}$ also satisfies the criteria $\hat{k}_{ii} = z^{'}$, some constant value. Hence, the two class support vector machine problem also tend to be a minimum-enclosing ball (MEB) scenario. After formulating the QP problem of binary class SVM as MEB problem (Equation 2.13), the $(1 + \epsilon)$ approximation algorithm [84, 86] can be exploited to obtain the optimum solution on core set. This would attain a reduced time complexity as linear and a space complexity which is independent of both dimension and size of input data sample. The entire process can be elucidated in Algorithm 1.

Algorithm 1: Core Vector Machine

CVM (A, ϵ)

Inputs : T Dataset with N data points ; ϵ Approximation factor; k Kernel function;

Outputs: S Set of core vectors;

begin

$S_0 \leftarrow$ {Initial core vectors};

$c \leftarrow c\_MEB(S_0)$; `/* Center of current MEB */`

$r \leftarrow r\_MEB(S_0)$; `/* Radius of current MEB */`

$S \leftarrow S_0$;

while *(There exist $P \in T$ Such that $\hat{\phi}(P$ falls outside of the ball $B(c, (1 + \epsilon)r))$* **do**

Find a data sample P such that $\hat{\phi}(P$ is furthest away from c;

Update $S = S \cup \{P\}$;

Find new $MEB(S)$;

Update $c = c\_MEB(S)$;

Update $r = r\_MEB(S)$;

return S;

In this algorithm T $= \{(x_i, y_i)\}_{i=1}^{N}$ represent training dataset. The center c of the ball can be computed by the function $c\_MEB(.)$ as: $c = \sum_{i=1}^{m} \alpha_i \hat{\phi}(x_i)$ where $\alpha = [\alpha_1, \alpha_2 ... \alpha_m]^{'}$ is the Lagrange multipliers, m is the number of data samples in the current core set and $\hat{\phi}$ is the feature mapping corresponding to the transformed kernel $\hat{k}$. Similarly, the function $r\_MEB(.)$ computes radius r of current ball as : $r = \sqrt{\alpha' diag(\hat{k}) - \alpha' \hat{k} \alpha}$. This algorithm start with the initialization of core set (s_0) by including one data sample from each of the classes [89]. In each iteration, a data sample $P \in T$ in the feature space $\hat{\phi}(P)$ which is outside of the ball $(c, r(1 + \epsilon))$ and furthest from c is taken as core vector and it is added to the core set S. The process of fetching new core vector necessitate the distance computation between the center c of current

MEB and $\hat{\phi}(P_j), \forall P_j \in T$. It can be computed by the equation :

$$\left\| c - \hat{\phi}(P_j) \right\|^2 = \sum_{P_m, P_N \in S} \alpha_m \alpha_N \hat{k}(P_m, P_N) - 2 \sum_{P_i \in S} \alpha_i \hat{k}(P_i, P_j) + \hat{k}(P_j, P_j) \qquad (2.15)$$

Even though the CVM achieves a faster execution on large dataset it can only be used with certain kernel functions and methods. This limitation has been solved in Generalized Core vector Machines[91] by extending the CVM implementation with the center-constrained MEB problem. It still preserve the time complexity as linear and space complexity that is independent of both dimension and size of input dataset. Ball Vector Machines (BVM) is another implementation which exploits the fixed radius characteristic of Enclosing ball(EB) problem [90]. They also proposed $(1+\epsilon)$ approximation algorithms to obtain the core set, without using any numerical solver. Another major contribution towards the performance of CVM is Cluster Based Core Vector Machine[92]. It improves the performance of CVM by using Gaussian Kernel function irrespective of the orderings of pattern belonging to different classes with in the dataset. According to the authors their method employs a selective sampling based training of CVM using a novel based scalable hierarchical clustering algorithm. A powerful approach to scale up SVM for multi-class classification is Multi-class Core Vector Machine(MCVM)[89] that uses CVM techniques to solve the quadratic multi-class problem defining an SVM with vector valued output.

2.2.3 Kernel Based Extreme Learning Machines

Artificial neural networks (ANN) and kernel machines have already proved their potential in many machine learning applications. Gradient based learning algorithms are extensively used in the context of artificial neural networks. These methods possess interdependencies between the parameters in different layers of neural networks. Quadratic equations with tuning parameters are extensively utilized in kernel machines. These factors enforce iterative training procedures to adjust the learning parameters in both artificial neural networks and kernel machines. Extreme Learning Machines (ELM)tend to be a non-iterative learning paradigm which keep off these tiresome repeated parameter tuning in its training procedure. ELM was first proposed as a fast learning algorithm

for Single Layer Feed-forward Networks (SLFN). It exploits simple generalized matrix inverse operations to compute the output weights and facilitates random computation of input weights and biases [93, 94]. Unlike traditional learning algorithms, ELM tends to achieve smallest norm of output weights in addition to minimizing the training error. The generalization performance of feed-forward neural networks reaching minimal error improves as the norm of output weights decreases [95]. These elegant learning features are the crux of ELM to fulfill many real world applications including semantic conception detection for videos [96], face recognition [97, 98], XML document classification [99], nontechnical loss of electricity analysis [100] etc. ELM can also be utilized to train the threshold neural networks directly instead of approximating them with sigmoid functions [101]. Robotic execution failure prediction can be accomplished by Kernel based extreme learning machine [102].

There are many brilliant works that enhances the learning power of extreme learning machines. The Error minimized extreme learning machine (EM-ELM) [103] is another remarkable work in ELM. This approach automatically performs adding of hidden nodes in the generalized SLFNs, either one by one or group by group manner, which need not be neural alike. ELM based on-line sequential learning method has been proposed in [104], which learns the data one-by-one or chunk-by-chunk with fixed or varying chunk size. This on-line model is further extended with fuzzy logic for function approximation and classification problem [105]. Tensor decomposition based Extreme Learning Machine reflects a significant work in the realm of machine learning. This approach utilizes TUCKER and PARAFAC based tensor techniques for building a scalable ELM [106]. Multiple kernel learning with different combinations of kernels is also approached in the extreme learning machines [107]. Distributed ELM with kernels has the potential to handle the scalability aspects of kernel based ELM by utilizing parallel computing framework such as Map Reduce [108].

2.2.4 Kernel Principle Component Analysis

Principle Component Analysis (PCA) tend to be an unsupervised feature reduction approach. It attempts to reduce the dimensionality of dataset with minimal loss of information. In PCA, linear transformation methods are exploited to find the informative features that maximize the variance in the given dataset. PCA facilitates dimensionality reduction by projecting $d-$ dimensional dataset onto a $f-$ dimensional dataset where $f \leq d$. The entire process in PCA can be summarized as in Algorithm 2.

Algorithm 2: Principle Component Analysis

PCA (A, f)

Inputs : $A = \{(a_1, a_2, ..a_N)\}$ where $a_i \in \Re^d$: Dataset ; $f \leq d$: Subspace dimension
Output: $A^{'}$: Principal component
begin
$C_M \leftarrow \frac{1}{N} \sum_{i=1}^{N} a_i a_i^T$; `/* Covarience Matrix */`
$C_M E = \lambda E$; `/* Computing eigenvector (E) and eigenvalue (λ) */`
$w \leftarrow [e_1, e_2, ...e_f]$; `/* Select f eigenvectors corresponding to the highest eigenvalues */`
$A^{'} \leftarrow w^T A$; `/* Projecting Dataset A onto f dimensional subspace. */`
Return $A^{'}$

Principal component analysis seems to be effective only in linear dimensionality reduction. Recent innovation in unsupervised learning paradigm, the Kernel Principal Component Analysis (KPCA) facilitates nonlinear feature reduction [109]. KPCA tend to be an enhanced PCA in the higher dimensional feature space which involves kernel function to extract the non-linear features in the higher dimensional space [110, 109]. This is accomplished by extracting the principal components (eigenvectors) from the covariance matrix in the higher dimensional feature space, which is same as nonlinear PCA in the original space. Here, kernel function facilitate a nonlinear bridge from the original space to the feature space. The process of modeling KPCA can be detailed as follows. Let $\phi : \Re^d \rightarrow F$ be the nonlinear mapping. Then, the covariance matrix in the induced feature space can be expressed as [111, 109] :

$$C_{vm}^F = \frac{1}{N} \sum_{i=1}^{N} \phi(a_i)\phi(a_i)^T \tag{2.16}$$

As in the case of linear PCA, the eigenvalue problem facilitates the computation of principal components in the feature space. It can be expressed with eigenvectors E in feature space F and eigenvalues λ as :

$$\begin{aligned} & C_{vm}^{F} E = \lambda E \\ \Rightarrow \quad & \left(\frac{1}{N} \sum_{i=1}^{N} \phi(a_i) \phi(a_i)^T \right) E = \lambda E \\ \Rightarrow \quad & \frac{1}{N} \sum_{i=1}^{N} \langle \phi(a_i), E \rangle \, \phi(a_i) = \lambda E \end{aligned} \tag{2.17}$$

A noticeable observation in this expression is that E with $\lambda \geq 0$ span over $\sum_{i=1}^{N} \phi(a_i)$. It implies that :

$$\left\langle \phi(a_k), C_{vm}^{F} E \right\rangle = \lambda \left\langle \phi(a_k), E \right\rangle \quad \forall k = 1, ..N \tag{2.18}$$

Another interesting property of eigenvector is that it can be expressed as the linear combination of $\phi(a_i)$. It can be expressed as:

$$E = \sum_{i=1}^{N} \alpha_i \phi(a_i) \tag{2.19}$$

Exploiting these properties, Equation 2.17 can be further expanded as:

$$\begin{aligned} & \left\langle \phi(a_k), \frac{1}{N} \sum_{i=1}^{N} \langle \phi(a_i), E \rangle \, \phi(a_i) \right\rangle = \lambda \left\langle \phi(a_k), E \right\rangle \quad \forall_{k=1,..n} \\ \Rightarrow \quad & \left\langle \phi(a_k), \frac{1}{N} \sum_{i=1}^{N} \left\langle \phi(a_i), \sum_{j=1}^{N} \alpha_j \phi(a_j) \right\rangle \phi(a_i) \right\rangle = \lambda \left\langle \phi(a_k), \sum_{j=1}^{N} \alpha_j \phi(a_j) \right\rangle \\ \Rightarrow \quad & \frac{1}{N} \sum_{i=1}^{N} \alpha_i \left\langle \phi(a_k), \sum_{j=1}^{N} \phi(a_j) \right\rangle \langle \phi(a_j), \phi(a_i) \rangle = \lambda \sum_{i=1}^{N} \alpha_i \left\langle \phi(a_k), \phi(a_i) \right\rangle \end{aligned} \tag{2.20}$$

In matrix form it can be expressed as $\lambda N K \alpha = K^2 \alpha$. Where K is an $N \times N$ matrix : $K[i,j] = \langle \phi(a_i), \phi(a_j) \rangle$. It can be solved through the eigenvalue problem:

$$\lambda n \alpha = K \alpha \tag{2.21}$$

Now, the first f principal components of data sample $a \in A$ can be computed by projecting $\phi(a)$ on to the eigenvectors $(e_1, e_2..e_f)$ in the feature space F as :

$$\begin{aligned} a^{'} &= \langle e_k, \phi(a) \rangle \,, \quad \text{for p= 1,..f} \\ &= \sum_{i=1}^{n} \alpha_i^p \, \langle \phi(a_i), \phi(a) \rangle \end{aligned} \tag{2.22}$$

In this scenario, kernel function $k(a_i, a_j) = \langle \phi(a_i), \phi(a_j) \rangle$ can play a vital role in the implicit computation of sample similarities in the feature space. The Equation 2.21 assumed that the data samples were centered (zero mean) in the feature space F. In other cases, K is replace with Gram matrix $K^{'} = K - 1_N K - K 1_N + 1_N K 1_N$, where 1_N is a $N \times N$ matrix with all elements as $\frac{1}{N}$.

Kernelization in PCA confer an elegant feature extraction architecture in unsupervised learning paradigm. Kernel PCA has been extensively utilized as the primary means of feature reduction in many real world applications . Novelty detection [112], improved ECG-derived respiratory signals detection algorithm [113], face recognition [110], etc. are some prominent examples in this context. The utilization of Kernel PCA frameworks as an efficient feature extraction tool for building non-linear machines has a predominant role in the domain of kernel machines.

2.3 Multiple Kernel Learning

Conventional kernel based learning frameworks encompasses solitary kernel for its feature transformation. The generalization performance of such machines strongly depends on the choice of kernel function being used. The selection of such designated kernel function necessitate the domain knowledge of the task being modeled. Usually, the selection of optimal kernel among a set of kernel functions is accomplished by the cross-validation procedure. This laborious process of kernel selection quell the major effort of researchers while modeling the machine intelligent frameworks. Unintelligent selection of such solitary kernel can be addressed by the predominant learning approach called Multiple Kernel Learning (MKL). This approach tend to obtain the optimal kernel by letting the learner to choose or combine the base kernels [114]. Let $\{(a_i, b_i)\}_{i=1}^{N}$;

$a_i \in \Re^d, b_i \in \Re$ be the set of training samples and $K = k_1, ...k_m$ is the set of m kernels. The mathematical representation of optimal kernel in MKL takes the form :

$$\mathcal{K}(a_i, a_j) = F\left(\left\{k_p(a_i^p, a_j^p)\right\}_{p=1}^{m} | \mu\right) \tag{2.23}$$

where $F(.)$ is the combination function that map $\Re^m \rightarrow \Re$ and it can be linear, non-linear or data dependent. Kernel functions k_p take m feature representations of data instances $a_i = \{a_i^p\}_{p=1}^m$ and $a_j = \left\{a_j^p\right\}_{p=1}^m$; where $a_i^p, a_j^p \in \Re^{d_p}$. d_p is the dimensionality of the p^{th} feature representation. μ parameterizes the combination function. Here the kernel functions and its parameters are known in advance.

It is also possible to integrate the combination parameter μ onto the kernel functions itself such that it can be optimized during training and it may expressed as:

$$\mathcal{K}(a_i, a_j) = F\left(\left\{k_p(a_i^p, a_j^p | \mu)\right\}_{p=1}^{m}\right) \tag{2.24}$$

The popular one among these kernel combinations, the linear combination takes the advantages of using linear methods to parameterize the combination function, expressed as:

$$\mathcal{K}(a_i, a_j) = \sum_{p=1}^{m} \mu_p k_p(a_i^p, a_j^p) \tag{2.25}$$

Here the combination parameter μ (kernel weights) is restricted to $\mu \in \Re^m$. The constraint $\mu \in \Re_+^m$ makes the above combination as conic where as the convex combination puts the limit as $\mu \in \Re_+^m$ and $\sum_{p=1}^m \mu_p = 1$. The decision function of the MKL machine corresponding to the liner combination (Equation 2.25) can be expressed as :

$$\begin{aligned} F(a) &= \sum_{i=1}^{N} \alpha_i \mathcal{K} \\ &= \sum_{i=1}^{N} \sum_{p=1}^{m} \alpha_i \mu_p k_p(a_i^p, a^p) \end{aligned} \tag{2.26}$$

In this basic formulation, the learner parameter α and the kernel combination parameter μ can be optimized either in combined (single-pass) or an alternative fixing (two-pass) method. In combined method both the combination parameters and the learner parameters are tuned up simultaneously. However, in the case of alternative strategy, the

combination parameters are tuned separately while fixing learner parameters and vice versa [115].

In contrast to the linear and non-linear (non-linear combination function), data dependent combination has the potential to learn the kernel combination criteria for each region by identifying the local distribution in the data sample. It is accomplished by assigning separate kernel weights for each pair of data samples. The mathematical expression for data dependent kernel combination is of the form :

$$\mathcal{K}(a_i, a_j) = \sum_{p=1}^{m} \mu_p k_p(a_i^p, a_j^p) \tag{2.27}$$

where $\mu_p = g(a_i, a_j)$, that compute a weight corresponding to (a_i^p, a_j^p).

Many approaches were proposed to introduce multiple kernel learning in kernel machines. The conic combination of per-defined kernels for the eminent kernel machine, the support vector machines were attempted in [116]. In this paper they have conceived a mechanism to learn the kernel combination and support vector machines simultaneously. Their convex optimization strategy is popularly known as quadratically-constrained quadratic program (QCQP). This approach applies sequential minimization strategies on non-smooth convex optimization paradigm. The semi-definite programming approaches facilitate learning of kernel matrices from the data. The limitation in number of kernels and the data points in this model were addressed in [114]. This paper exploited Moreau-Yosida regularization and sequential minimal optimization (SMO) algorithms to rebuilt the QCQP problem as a second-order cone programming. A learning model in conjunction with semi-infinite formulation of QCQP and support vector machine was proposed in [117]. They have also showed that, their approach can be generalized to one-class classification and regression. The multi-class multiple kernel learning approach exploits joint feature-maps for learning different kernels on the output spaces [118]. This approach uses a common feature-space for entire divergent classes. The requisite for specificity of the kernel in identifying different classes, delimits the utilization of common feature-space. This multi-class multiple kernel learning approach can be incorporated in deep learning paradigm. The localized multiple kernel learning approach utilizes a gating strategy for the selection of appropriate kernels [119]. The

coupling of localized gating strategy, kernel based classifier and thereby jointly performing optimization tend to be the key features of this approach. This framework also have the potential in combining multiple duplicates of identical kernels, which is localized in divergent parts. Later, Jose et al. [120] provided a generalization to localized multiple kernel learning approach for learning sparse, deep and high-dimensional tree-based primal feature-embedding. This primal-based classification facilitates decoupling of prediction-costs from amount of support vectors. This approach can ensure scalability of non-linear support vector machines on massive data.

The traditional MKL algorithms are obliged to the class labels of training data (supervised) for learning the kernels. However, there are many avenues where the kernel learning task has to be accomplished without class labels, the class labels are not available. Unsupervised Multiple Kernel Learning (UMKL) approach proposed by Zhuang et al. tend to be a successful attempt that facilitates multiple kernel learning from unlabelled data points. In their paper [121], they have formulated the UMKL by combining the following two intuitions on the optimal kernel $\mathcal{K}$.

- $\mathcal{K}$ has the capability to reconstruct the original training data a_i from its localized bases ($\sum_j a_j$)weighted by the kernel matrix; I.e $\left\| a_i - \sum_j k_{ij} a_j \right\|^2$ will be minimum, where $k_{ij} = \mathcal{K}(a_i, a_j)$.
- $\mathcal{K}$ has the potential to minimize the distortion $\sum_{i,j} k_{ij} \left\| a_i - a_j \right\|^2$ over all training data.

The authors have also maintained a list of local bases ($\mathcal{B}_i$) for each data sample $a_i \in A$, for exploiting the concepts of locality preservation.

The optimization problem for the unsupervised multiple kernel learning has been achieved by combining the above intuitions as :

$$\min_{k \in \mathcal{K}, B} \frac{1}{2} \sum_{i=1}^{N} \left\| a_i - \sum_{a_j \in \mathcal{B}_i} k_{ij} a_j \right\|^2 + \gamma_1 \sum_{i=1}^{N} \sum_{a_j \in \mathcal{B}_i} k_{ij} \left\| a_i - a_j \right\|^2 + \gamma_2 \sum_i |\mathcal{B}_i| \tag{2.28}$$

Nowadays, there have been wide spread adoption of MKL in real world applications. Object identification [122], bio-informatics [123], image annotation and computer vision [124, 125] , etc. are some of the prominent applications in this context.

2.4 Deep Kernel Machines

Kernel machines exhibit the potentiality to decouple data representation from the designing of learning algorithms. The crux in these models, the kernel functions facilitate more expressive representation by means of similarity between the samples in the high dimensional feature space. The capability of building more complex kernels upon simple ones also empowers the kernel machines to model many real world applications. However, conventional shallow based kernel learning approaches constitute single-layered kernel computation (non-linear feature transformations). These single layered data representation models reflect incapability in modeling highly complex and varying applications which require the extraction of complex-type structures and concrete internal representations from raw input. In order to amplify the scope and ease of applicability of machine learning, it would be highly desirable to build a learning model that should exploit multiple layers of feature learning for the deep representation of data. Recent works in kernel machines highlight the necessity of layered data representations upon which a general classifier /predictor can be placed.

Multiple layers of feature representation has been effectively conceived by deep learning approaches. The deep layered learning architectures offer openness in building richer representation from raw data, through multiple layers of non-linear transformation. In addition to this, deep learning models exhibit the potentiality to combine both supervised and unsupervised learning paradigms. Deep learning methods originate and have been maintaining an astounding performance in the context of neural networks. However,they involve highly nonlinear optimizations and many heuristics for gradient-based learning, which do not guarantee global optimal solutions and proliferates the local optimization with increased network depth [32, 30]. Deep neural networks are seem to be succeeded only in applications that involves huge amount of data.

Unlike neural network, kernel machines ensure global optima by virtue of its convex optimization strategies. In addition to this, kernel approaches utilized only a few parameters to learn the decision boundary by projecting the data onto the high dimensional feature space (reproducing kernel Hilbert space). It enables the classifiers to succeeded on applications involving different sample sizes [126]. Kernel machines also exhibit the features like structural risk minimization, maximal marginal classification, etc. The wide spread acceptance of deep learning approaches in the context of neural network and the elegant properties of kernel machines motivated some researchers to explore the possibilities of leveraging deep learning capabilities in kernel machines. However, only a few attempts in this context were reported in machine learning literature.

2.4.1 Deep Kernel Computation

Inspiring from the success of deep neural network, there have been many attempts to build kernels that mimicking the multilayer computation in deep neural networks. Utilization of deep learning capabilities as an efficient abstraction tool has a prominent role in building deep kernel machines. Let the training sample of input-output pairs $\{(x_i, y_i)\}_{i=1}^{N}$ where $x_i \in \Re^d$, $y_i \in \{-1, +1\}$ and $\mathcal{K}(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{F}}$ be the kernel function. In general, multilayer (l layered) kernel computation is accomplished by applying the non-linear mapping (ϕ) recursively on data samples and then exercise inner product between the samples in the feature space. It can be expressed as :

$$k^l(x_i, x_j) = \langle \phi^l(\phi^{l-1}(...\phi^1(x_i))), \phi^l(\phi^{l-1}(...\phi^1(x_j))) \rangle \tag{2.29}$$

Here the intuition is that, if the single layer network computation can be represented by the base kernel $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ then the $l-$ successive applications of the mapping $\phi(.)$ in the Equation 2.29 can imitate multilayer computation in neural networks.

This concept can be further illustrated with popular kernels such as polynomial kernel and RBF kernel. In the case of homogeneous polynomial kernel, $\mathcal{K}_{POLY}(x_i, x_j) =$

$(x_i.x_j)^d$, the two layer kernel composition can be constructed as [49]:

$$\begin{aligned}\mathcal{K}^{(2)}_{POLY}(x_i, x_j) &= \phi(\phi(x_i)).\phi(\phi(x_j)) \\ &= (\phi(x_i).\phi(x_j)^d \\ &= ((x_i.x_j)^d)^d \\ &= (x_i.x_j)^{d^2}\end{aligned}$$

Similarly, for RBF kernel $\mathcal{K}_{RBF}(x_i, x_j) = e^{-\lambda\|x_i - x_j\|^2}$, the two-fold composition of RBF kernel, $\mathcal{K}^{(2)}_{RBF}(x_i, x_j)$ can be derived as [49]:

$$\begin{aligned}\mathcal{K}^{(2)}_{RBF}(x_i, x_j) &= \phi(\phi(x_i)).\phi(\phi(x_j)) \\ &= e^{-\lambda\|\phi(x_i) - \phi(x_j)\|^2} \\ &= e^{-2\lambda(1 - \mathcal{K}_{RBF}(x_i, x_j))}\end{aligned}$$

In these multilayer kernel computation, it is hard to quantify the qualitatively different representation than the base kernels. Two-fold polynomial kernel is geometrically same as polynomial kernel with higher degree and the two layer RBF kernel is nothing more than another RBF kernel with different variance. A breakthrough in this context was the invention of arc-cosine kernel [49, 127]. Unlike polynomial and RBF kernels, arc-cosine kernel encode their inputs in the feature space that mimics the representations of single-layer threshold networks. Arc-cosine kernels were originally defined by exploring the activation function, the Heaviside step functions used in neural network. Subsequently, the activation functions such as one-sided polynomial functions, shifted Heaviside step functions, and sigmoidal functions were exploited to derive three more versions of this kernel. These different behavior of arc-cosine kernel can be accomplished by tuning its activation value (degree). The composition of arc-cosine kernel exhibit different geometric properties (behaviors). The multilayer arc-cosine kernel in modeling a kernel machine mimics the computations in multilayer neural network. It expedite the process of leveraging deep learning capabilities in kernel machines. The following section detailing arc-cosine kernel and its multilayer compositions.

The n^{th} order kernel in the family of arc-cosine kernel with Heaviside step functions $\Theta(z) = \frac{1}{2}(1 + sign(z))$ can be expressed as [49] :

$$\mathcal{K}_n(x_i, x_j) = \quad 2 \int dw \frac{e^{-\frac{\|w\|^2}{2}}}{(2\pi)^{\frac{d}{2}}} \Theta(w.x_i)\Theta(w.x_j)(w.x_i)^n(w.x_j)^n \tag{2.30}$$

This integral expression computed by arc-cosine kernel exhibit a close relationship to the computation of multilayer threshold networks. Let $f(x) = G(W \cdot x)$ be the non-linear mapping in single layer neural network that transform the input x to the output space $f(x)$. The parameter W represent the weight matrix that connects input units to the output units. The non-linearity G is described by the network activation function, the one-sided polynomial activation function as $G_n(z) = \Theta(z)z^n$. Here $\Theta(z) = \frac{1}{2}(1 + sign(z))$. For the two data samples x_i and x_j, the inner product of their corresponding output is

$$f(x_i) \cdot f(x_j) = \sum_{k=1}^{p} \Theta(w_k \cdot x_j)\Theta(w_k \cdot x_j)(w_k \cdot x_i)^n(w_k \cdot x_j)^n$$

where p is the number of output neurons. With the assumption that, the weight matrix W has a Gaussian distribution with zero mean and unit variance and the network has an infinite number of output units it can be expressed as:

$$lim_{p\to\infty} \frac{2}{p} f(x_i) \cdot f(x_j) = \mathcal{K}_n(x_i, x_j)$$

where

$$\mathcal{K}_n(x_i, x_j) = 2 \int dw \frac{e^{-\frac{\|w\|^2}{2}}}{(2\pi)^{d/2}} \Theta(w \cdot x_i)\Theta(w \cdot x_i)(w \cdot x_i)^n(w \cdot x_j)^n$$

This expression is similar to one expressed in Equation 2.30. In this integral representation of arc-cosine kernel, it has been observed that arc-cosine kernel exhibit a trivial dependency on the magnitudes of the data samples x_i, x_j and a complex dependency on the angle between them [127]. The angle θ between inputs x_i, x_j can be expressed as:

$$\theta = \cos^{-1}\left(\frac{(x_i.x_j)}{\|x_i\|\|x_j\|}\right) \tag{2.31}$$

By conquering all the angular dependencies into a single function $AD(n, \theta)$, the integral expression in Equation 2.30 is simplified further as :

$$\mathcal{K}(x, y|n) = \frac{1}{\pi} \|x\|^n \|y\|^n AD(n, \theta) \tag{2.32}$$

where the angular dependency

$$AD(n, \theta) = (-1)^n (sin\theta^{2n+1})(\frac{1}{sin\theta}\frac{\partial}{\partial\theta})^n(\frac{\pi - \theta}{sin\theta}), \quad \text{for } \forall n = 1, 2, 3.. \tag{2.33}$$

For $n = 0$, the angular dependency $AD(0, \theta) = \pi - \theta$, simply an element of angle between the inputs, the Equation 2.32 takes the form as :

$$\begin{aligned} \mathcal{K}(x, y|0) &= 1 - \frac{\theta}{\pi} \\ &= 1 - \frac{1}{\pi} \cos^{-1}\left(\frac{(x_i.x_j)}{\|x_i\|\|x_j\|}\right) \end{aligned} \tag{2.34}$$

The other expressions of angular dependency with $n = 1$ and $n = 2$ are :

- $AD(1, \theta) = sin\theta + (\pi - \theta)cos\theta$
- $AD(2, \theta) = 3sin\theta cos\theta + (\pi - \theta)(1 + 2cos^2\theta)$

The parameter n involved in these expression usually referred as the activation value or degree which determines the behavior of arc-cosine kernel. The arc-cosine kernel exhibit qualitatively different geometric properties with different activation functions [127]. For the case $n = 0$, the kernel $\mathcal{K}(x_i, x_i|0)$ takes the value 1 and it facilitates the arc-cosine kernel to maps input $x_i \in \Re^d$ to the unit hypersphere in feature space and it behaves like Radial Basis Function (RBF). For the case $n = 1$, $\mathcal{K}(x_i, x_i)$ takes the expression $\mathcal{K}(x_i, x_i|1) = \|x\|^2$ and it exploited the norm of input which is similar to linear kernel. In the case of $n \geq 1$, it expand the dynamic range of the input, with $\mathcal{K}(x_i, x_j|n) = \|x\|^{2n}$ and it is similar to polynomial kernel.

These arc-cosine kernels possess the capability to intensifying the learning models with multilayer kernel computation [52, 53]. The multi-layered, deep arc-cosine kernel can be formed by combining single layered arc-cosine kernel (Equation 2.32) in an iterative manner. Each instance of this iterative execution of multilayer arc-cosine kernel may

have different activation value and hence different behavior . This interesting property of multilayer arc-cosine kernel contributes a deep neural network computation in kernel machines. The Equation 2.32 represent the computation of single layer arc-cosine kernel having an activation value n. In the case of multilayer arc-cosine kernel the value of n may be different in different layers. Let $n1, n2..nl$ be the activation values corresponding to the layers $1, 2, ..l$ respectively. Then the l layered arc-cosine kernel can be represented as:

$$\mathcal{K}^{(L)}_{(n1,n2..nl)}(x_i, x_j) = \langle \phi_{nl}(\phi_{nl-1}(...\phi_1(x_i))), \phi_{nl}(\phi_{nl-1}(...\phi_1(x_j)))\rangle \tag{2.35}$$

It can be illustrated for 2-fold arc-cosine kernel with activation value ($n = 0$) as follows.

$$\begin{aligned}
\mathcal{K}^{(2)}_0(x_i, x_j) &= \langle \phi^1_0(\phi^2_0(x_i)), \phi^1_0(\phi^2_0(x_j))\rangle \\
&= 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\langle \phi^1(x_i|0), \phi^1(x_j|0)\rangle}{\|\phi^1(x_i|0)\| \quad \|\phi^1(x_j|0)\|}\right) \\
&= 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\mathcal{K}^1(x_i, x_j|0)}{\sqrt{\mathcal{K}^1(x_j, x_j|0) \quad \mathcal{K}^1(x_i, x_i|0)}}\right) \\
&= 1 - \frac{1}{\pi}\cos^{-1}\left(1 - \frac{\theta}{\pi}\right)
\end{aligned}$$

In general, the recursive formulation of $l-$ fold arc-cosine kernel can be expressed as :

$$\mathcal{K}^{(l+1)}(x_i, x_j|n) = \frac{1}{\pi}\left[\mathcal{K}^{(l)}(x_i, x_i|n)\ k^{(l)}(x_j, x_j|n)\right]^{\frac{n}{2}} AD(n, \theta^{(l)}_n) \tag{2.36}$$

where $\theta^{(l)}_n$ is the angle between the images of x_i and x_j in the feature space induced by the l-fold composition.

$$\theta^{(l)}_n = cos^{-1}\left(\frac{\mathcal{K}^{(l)}(x_i, x_j|n)}{\sqrt{\mathcal{K}^{(l)}(x_i, x_i|n) \quad \mathcal{K}^{(l)}(x_j, x_j|n)}}\right) \tag{2.37}$$

Above expression is build up on same activation value n at every layers of arc-cosine kernel. It also enables to use different activation values at different layers of recursion. The iterative procedure of arc-cosine kernel that mimics multilayer computation in neural network has been summarized in Algorithm 3. This algorithm takes the list of activation value as parameter, in addition to data samples x_i and x_j. In this activation list, the activation value of layers are arranged in chronological order. The length of

activation list determines the depth of arc-cosine kernel. The layered kernel learning process start with initial kernels $\mathcal{K}_{ii} = \langle x_i, x_i \rangle$, $\mathcal{K}_{jj} = \langle x_j, x_j \rangle$ and $\mathcal{K}_{ij} = \langle x_i, x_j \rangle$. In this algorithm, the sub function *Angular_depen* (Algorithm 4) is used to compute the angular dependency *Ad*.

Algorithm 3: Computing Multi layered Arc-cosine kernel

MLArccosine $(x_i, x_j, [act\_list])$
- **inputs** : Two data samples x_i and x_j; The list of activation or degree in each layer $[act\_list]$
- **output**: The kernel value denoted by $\mathcal{K}^l_{x_i,x_j}$
- **begin**
  - $\mathcal{K}_{ii} \leftarrow$ ***Inner-product***(x_i, x_i)**;**
  - $\mathcal{K}_{jj} \leftarrow$ ***Inner-product***(x_j, x_j)**;**
  - $\mathcal{K}_{ij} \leftarrow$ ***Inner-product***(x_i, x_j)**;**
  - **foreach** $n \in [act\_list]$ **do**
    - $\theta \leftarrow \cos^{-1}\left(\frac{\mathcal{K}_{ij}}{\|\mathcal{K}_{ii}\|\|\mathcal{K}_{jj}\|}\right)$**;**
    - $Ad \leftarrow$ ***Angular_depen***(n, θ)**;**
    - $\mathcal{K}_{ij} = \frac{1}{\pi} \times (\mathcal{K}_{ij})^{\frac{n}{2}} \times Ad$**;**
    - $\mathcal{K}_{ii} = \frac{1}{\pi} \times (\mathcal{K}_{ii})^{n} \times Ad$**;**
    - $\mathcal{K}_{jj} = \frac{1}{\pi} \times (\mathcal{K}_{jj})^{n} \times Ad$**;**
  - $\mathcal{K}^l_{x_i,x_j} \leftarrow \mathcal{K}_{i,j}$**;**
- **return** $\mathcal{K}^l_{x_i,x_j}$;

Algorithm 4: Algorithm for computing Angular dependency

Angular_depen (n, θ)
- **inputs** : Degree or activation value :n; Angle between input vectors θ
- **output**: The angular dependency :Ad
- **begin**
  - **switch** n **do**
    - **case** *0*
      - $Ad = \pi - \theta$
    - **case** *1*
      - $Ad = sin(\theta) + (\pi - \theta) \times cos(\theta)$
    - **case** *2*
      - $Ad = 3 \times sin(\theta) \times cos(\theta) + (\pi - \theta) \times (1 + 2 \times cos(\theta)^2)$
- **return** Ad;

Optimization of multilayer arc-cosine kernel opens up a new insight for deepening kernel machine with deep kernel computations. The deep kernel learning frameworks can be modeled efficiently by plugging this multilayer arc-cosine kernel in any kernel machines. One such well known example in this context is the deep support vector machines

(DSVMs) [49]. Even though DSVM shows better performance on many machine learning problems it encounters bottleneck, due to its high computational cost, on many real world application that involve massive data.

Other noticeable work in this context of deep kernel learning is Laplacian deep kernel learning [128]. In this paper, the authors devised a semi-supervised deep kernel network learning model, in conjunction with Laplacian SVMs. This model combines the high generalization capabilities of Laplacian SVMs and the strength of deep neural networks. The effectiveness of this deep kernel classifier was demonstrated through image annotation experiments. A kernel method that encode information of deep infinite layer neural network was devised in [129]. In this paper, the authors derived stochastic kernels from Gaussian process and showed the alignment of infinite neural network with Gaussian process and kernel method. This kernel framework involves generalization bound and regularization method to eliminate overfitting. Adopting from convolutional neural network, these network exploited localities and non-linearities. This framework is demonstrated with only two layers, beyond that it failed to address non-linearities.

2.4.2 Deep Kernel Learning Architecture

Recently there are several attempt to build deep kernel learning architecture by the hierarchical organization of shallow based kernel machines. In the context of image prediction task, three divergent interpretations of deep kernel learning using multiple kernels are proposed in the paper [130]. This work modeled a breast cancer classification algorithm by exploiting the shallow based SVM with multiple kernel learning. This model also employed the span bound optimization over the dual objective function. Three different deep kernel learning architectures proposed in this paper are shown in Figure 2.11. In their first model they explored three-layered deep kernel learning architecture with two kernel combinations in every layer. The first layer constitute two elementary kernels and the remaining layers involve the combined kernels from its previous layer. The final kernel is computed as a linear combination of learned kernels from three layers. Second model is a two-layered deep learning architecture. In this model, the first layer kernels and first kernel in the second layer are elementary kernels

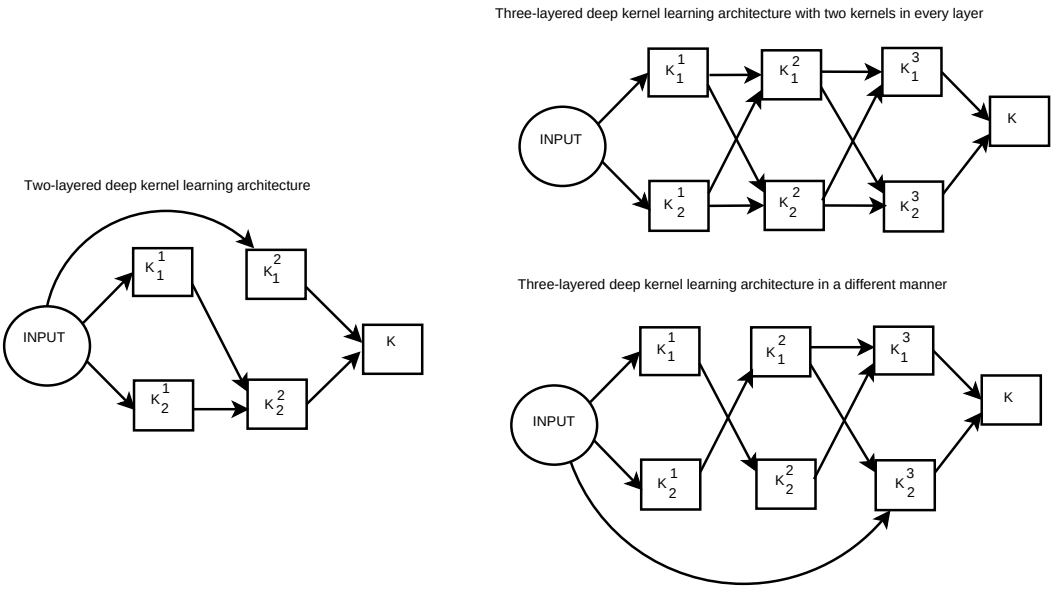

FIGURE 2.11: Different interpretations of deep kernel learning architecture

whereas the second kernel in second layer is learned from kernels in the first layer. The outermost kernel computation involves the linear combination of kernels in the second layer. Third model encompasses three layers of kernel computations. The base kernels which map the original data are presented in the first layer. In second layer, the first and second kernels are mapped to second and first kernel in the first layer respectively. In third layer, the first kernel involves the combination of both the kernels in second layer where as the second kernel maps the combination of original data and the output of first kernel in the second layer. Obviously, the linear combination of third layer kernels outputs the final kernel.

Other noticeable works in the context of deep layered multiple kernel learning approaches is a "two-layer multiple kernel learning" (2LMKL) approach [131]. Unlike traditional single layer MKL approach, this work explored a multiple kernel learning in a hierarchical manner. In this work the authors devised an algorithm for two layer multiple kernel learning. The authors have also exploited the idea of infinite kernel learning to generate base kernels in iterative manner. This approach exhibits better flexibility in comparison with regular multiple kernel learning approaches for exploring optimum kernels in vast applications. Also, alternating-optimization algorithm is utilized for learning base kernel weights and decision functions simultaneously. Only a solitary Gaussian radial basis function is applied at the second layer. This paper empirically showed that

2LMKL and 2LMKL^{Inf} techniques outperformed other techniques such as support vector machines (SVM), convex multiple kernel learning (MKL) and multiple kernel learning with Lp norm regularization (LpMKL). However this approach is attempted only on two-layered strategy and required further enhancements for building deep architecture.

A back-propagation based deep multilayer multiple kernel approach [132] was another attempt to leverage deep learning capabilities in kernel machines. This approach utilizes gradient-ascent strategy instead of leave-one-out error or dual objective-function estimation for learning the optimal kernel combinations. The empirical result shows that this approach outperforms two-layer multiple kernel learning (2LMKL) [131] and deep multi-layer multiple kernel learning (DMKL) [126] approaches.

Based on the literature survey as discussed above, it is learned that the proposed problem statement is highly relevant in the context of advancing science in the area of building deep learning capabilities in kernel machines.

CHAPTER 3

Deep Learning in Core Vector Machines

THE recent advancement in machine learning, the deep learning approach has been moving towards greater heights in pragmatic applications by giving due importance to both data representation and classification methods. It facilitated remarkable success in many real world intelligent applications involving complex, highly varying, higher dimensional, large volume data set. Even though deep learning has been investing its effort primarily in the context of neural networks, there are several recent explorations of deep learning techniques for other machine learning paradigms such as kernel machines.

Recently developed multilayer arc-cosine kernel [49] tend to be the bedrock for extending deep learning capabilities in kernel machines. Arc-cosine kernel represent a family of kernels capable of expressing different behaviors with different activation functions. This multilayer arc-cosine kernel is extensively utilized to build deep kernel learning in conjunction with many well-known kernel machines. A celebrated learning model in this context is Deep Support Vector Machine (DSVM) [49] which combines multilayer arc-cosine kernel and Support Vector Machines (SVM). Quadratic Programming (QP) problem involved in the implementation of SVM impose a time complexity in the order of $O(n^3)$. In addition to this, the use of multilayer arc-cosine kernel increases the time complexity of DSVM further. Even though this deep model has been widely used in many applications that involve small datasets, it fails to scale in problems with large datasets.

Core Vector Machine (CVM) tend to be a scalable SVM formulation leveraging a Minimum Enclosing Ball (MEB) approximation algorithm [47]. In CVM, the QP problem involved in SVM is reformulated as an equivalent MEB problem [81, 133, 134, 135, 136] and then solved by using a subset of training samples (Core Set) obtained by a faster $(1+\epsilon)$ approximation algorithm [83, 84, 86]. CVM achieves a time complexity that is linear in training sample size (m) and a space complexity that is independent of training sample size [92]. In this context, CVM is identified as a potential candidate to build a scalable deep SVM algorithm. The main challenge in building deep core vector machine is the restriction on the kernels that can be used in CVM. It support certain kernels that satisfying the condition $k(x,x) = z$, a constant. This chapter diligently discuss the prominent characteristics of multilayer arc-cosine kernel and the scalable deep kernel machine modeled by combining arc-cosine kernel and core vector machines. This chapter is organized as follows. Section 3.1 explores deep kernel learning in core vector machines. Experimental details and performance measures are discussed in Section 3.2. Section 3.3 conveys conclusions.

3.1 Deep Kernel Learning in Core Vector Machines

Core Vector Machine (CVM) seems to be an efficient mechanism to address the scalability challenge in SVM. It rely on the concept of equivalent Minimum Enclosing Ball problem. CVM tend to obtain approximately optimal solution by reformulating the QP problem of SVM as an equivalent MEB problem and then solved by using an efficient MEB based approximation algorithm by utilizing the idea of core set (a subset of sample set). As discussed in Chapter 2 (Section 2.2.2), any kernel machine using the kernel function $\mathcal{K}$ satisfying the condition $k_{ii} = z$ can easily be converted into an equivalent MEB problem. Based on this concept, the wolfe duel formulation of two class SVM (explained in Chapter 2) can be expressed as [89, 91]:

$$\begin{aligned} &\min_{\alpha} \sum_{i,j=1}^{N} \alpha_i \alpha_j (y_i y_j k(x_i, x_j) + y_i y_j + \frac{\delta_{ij}}{C}) \\ &s.t. \sum_{i=1}^{N} \alpha_i = 1 \quad , \alpha_i \geq 0 \quad \forall i \end{aligned} \tag{3.1}$$

where $x_i \in \Re^d$, $y_i \in \{+1, -1\}$, α is Lagrangian multiplier, $C > 0$ is an user defined parameter that controls the trade-off between the margin and training error and δ_{ij} is the kronecker delta function[1]. By using the transformed kernel $\hat{k}_{ij} = y_i y_j(k_{ij} + 1) + \frac{\delta_{ij}}{C}$, the above equation can be reformulated as:

$$\min_{\alpha} \sum_{i,j=1}^{N} \alpha_i \alpha_j \hat{k}_{ij} \quad s.t. \sum_{i=1}^{N} \alpha_i = 1 \;\; \alpha_i \geq 0 \;\; \forall i \tag{3.2}$$

The interesting observation in this formulation is that when the kernel $k(x_i, x_i) = z$, a constant is satisfied, then the transformed kernel ($\hat{\mathcal{K}}$) also satisfies the condition $\hat{k}(x_i, x_i) = z^{'}$, some constant. It is observed that, the isotropic kernel: $k(x_i, x_j) = \|x_i - x_j\|$ (e.g., Gaussian kernel), dot product kernel: $k(x_i, x_j) = x_i^{'} x_j$ (e.g.,polynomial kernel with normalized inputs) and all the normalized kernels are satisfying the condition $k_{ii} = z$ and hence the kernel machine involving these type of kernel can easily be converted to an MEB problem.

The eminent arc-cosine kernel tend to be the vital element in building deep kernel machines. The multilayer computation in this arc-cosine kernel enables the process of designing kernel machines with deep learning capabilities. This kernel is widely used in conjunction with SVM and produces elegant results in many machine learning applications that involve only small datasets. The following section analyzes the characteristics of multilayer arc-cosine kernel to explore the possibilities of utilizing this kernel in CVM, to build a scalable deep kernel machine.

3.1.1 Analysis of Arc-cosine Kernel

Introducing arc-cosine kernel in SVMs was the first attempt in incorporating the concepts of deep learning in kernel machines. Each layer in this multilayer arc-cosine kernel is defined with an activation value, referred as its degree. The performance of each layer in this hierarchy is highly influenced by the activation value defined for that layer. The ability to utilize divergent activation functions makes the arc-cosine kernel to express different behaviors and hence represents a family of kernels. The arc-cosine kernel with

[1] $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$

different activation functions have qualitatively distinct geometric properties [127]. The arc-cosine kernel and its different behaviors are clearly detailed in Chapter 2. It is summarized as follows. As discussed in Chapter 2,the simplified form of n^{th} order arc-cosine kernel (Equation 2.32) can be expressed as :

$$k_n(x,y) = \frac{1}{\pi} \|x\|^n \|y\|^n AD(n,\theta)$$

Here, the angular dependency $AD(n,\theta)$ (Equation 2.33) depends on two factors such as degree or activation value n and the angle θ between data samples, say x_i and x_j.

$$AD(n,\theta) = (-1)^n (sin\theta^{2n+1})(\frac{1}{sin\theta}\frac{\partial}{\partial\theta})^n(\frac{\pi-\theta}{sin\theta})$$

$$\theta = \cos^{-1}\left(\frac{(x.y)}{||x||\,||y||}\right)$$

Now, the first three forms of angular dependency can be computed by exploring the above equations as follows:

- $AD(0,\theta) = \pi - \theta$
- $AD(1,\theta) = sin(\theta) + (\pi-\theta) \times cos(\theta)$
- $AD(2,\theta) = 3 \times sin(\theta) \times cos(\theta) + (\pi-\theta) \times (1 + 2 \times cos(\theta)^2)$

In multi layered arc-cosine kernel each layer may have different activation function(degree). The non-linear transformation in each layer is determined by the value of its associated degree. The arc-cosine kernel with different activation functions have qualitatively different geometric properties [127]. Let n be the degree of the arc-cosine kernel at a particular level. Then for the case $n = 0$ arc-cosine kernel maps input x to the unit hypersphere in feature space, with $k_0(x,x) = 1$ and it behaves like Radial Basis Function (RBF). For the case $n = 1$, arc-cosine kernel preserves the norm of input with $k_1(x,x) = ||x||^2$ which is similar to linear kernel and in the case of $n \geq 1$, it expand the dynamic range of the input, with $k_n(x,x) = ||x||^{2n}$ and it is similar to polynomial kernel. From the above analysis it is clear that, in some special cases, particularly with

degree 0, arc-cosine kernel satisfies the required condition $k(x_i, x_i) = z$, (a constant) for being a kernel function in CVM. Following section explain the process of building scalable deep kernel machine by combining multilayer arc-cosine kernel and core vector machine.

3.1.2 Building Scalable Deep Kernel Machines

From the discussions in Section 3.1 and 3.1.1, it is clear that the arc-cosine kernel can be employed to build scalable deep kernel learning architecture with core vector machine. The first step in CVM is reformulation of quadratic programming problem of support vector machines to an equivalent Minimum Enclosing Ball (MEB) problem. It then exploits the well-known $(1+\epsilon)$ MEB approximation algorithm to solve the reformulated MEB problem. This approximation algorithm using the subset of sample set called core set enables CVM as an appropriate choice for building kernel machine for data intensive applications.

Equation 3.2 shows an equivalent MEB formulation of two class support vector machine, where the transformed kernel $\hat{k}_{ij} = y_i y_j (k_{ij} + 1) + \frac{\delta_{ij}}{C}$. In this transformed kernel k_{ij} represent the actual kernel function and it should satisfy the condition $k_{ii} = z$, a constant.

Now, the scalable deep kernel machine, Deep Core Vector Machine (DCVM) can be modeled by inducing the multilayer arc-cosine in the Equation 3.2. The mathematical model of multi-layered kernel induced MEB problem is then defined as :

$$\min_{\alpha} \sum_{i,j=1}^{N} \alpha_i \alpha_j \hat{k}_n^l(x_i, x_j) \quad s.t. \sum_{i=1}^{N} \alpha_i = 1, \quad \alpha_i \geq 0 \ \ \forall i \tag{3.3}$$

where the transformed kernel

$$\hat{k}_n^l(x_i, x_j) = y_i y_j (k_n^l(x_i, x_j) + 1) + \frac{\delta_{ij}}{C} \tag{3.4}$$

Here $K_n^l(x_i, x_j)$ represent l- fold arc-cosine kernel with degree n and $C > 0$ is an user defined parameter that controls the trade-off between the margin and training error.

The recursive expression of l - fold arc-cosine kernel can be expressed as :

$$k_n^l(x_i, x_j) = \frac{1}{\pi}[k_n^{(l-1)}(x_i, x_i)k_n^{(l-1)}(x_j, x_j)]^{\frac{n}{2}} AD(n, \theta_n^{(l-1)}) \tag{3.5}$$

In this l-fold composition of arc-cosine kernel, the angle $\theta_n^{(l)}$ between data samples x_i and x_j in the feature space can be expressed as :

$$\theta_n^{(l)} = cos^{-1}\left(\frac{k_n^{(l)}(x_i, x_j)}{\sqrt{k_n^{(l)}(x_i, x_i)k_n^{(l)}(x_j, x_j)}}\right) \tag{3.6}$$

The multi-layered, deep arc-cosine kernel can be formed by combining arc-cosine kernel in an iterative manner, with same or different activation function in each layer. The hierarchical multilayer arc-cosine kernel can be implemented as discussed in Chapter 2 (Algorithm 3). In this hierarchy, each layer computation is determined by the activation value of that layer and previous layer output.

3.1.3 Algorithm: Deep Core Vector Machines

As in the case of CVM, the idea of core sets obtained by a faster $(1+\epsilon)$ approximation [84, 86] can also be exploited to scale up the multi-layered kernel induced MEB problem for Deep Core Vector Machines. This $(1+\epsilon)$ approximation can be accomplished by an iterative strategy proposed by [83]. In this method, the current ball $B(c_i, r_i)$ at the i_{th} iteration is expanded with a multiplicative factor of $(1+\epsilon)$, and then updates the core set with a new vector which is the furthest point outside the ball $B(c_i, (1+\epsilon)r_i)$. This process is repeated until all the data points in dataset are covered by $B(c_i, (1+\epsilon)r_i)$. This iterative $(1+\epsilon)$ approximation for DCVM is shown in Algorithm 5.

In this algorithm T $= \{(x_i, y_i)\}_{i=1}^N$ represent training data set. The center c of the ball can be extracted by the function $c\_MEB(.)$ as: $c = \sum_{i=1}^m \alpha_i \hat{\phi}_n^l(x_i)$, where m is the number of data samples in the core set, $\alpha = [\alpha_1, \alpha_2 ... \alpha_m]^{'}$ be the Lagrange multipliers and $\hat{\phi}_n^l$ is the feature mapping corresponding to the transformed multilayer kernel $\hat{k}_n^l$. The list of activation or degree for each layer of arc-cosine kernel is given by the set L and its size determines the number of layers. Similarly, $r\_MEB(.)$ computes radius r of current ball as: $r = \sqrt{\alpha^{'} diag(\hat{k}_n^l) - \alpha^{'} \hat{k}_n^l \alpha}$. The initial core set (S_0) is obtained by

Algorithm 5: Deep Core Vector Machine

DCVM (T, ϵ, L)

```
// T Dataset ; ϵ Approximation factor ; L List of activation value for
   each layer of arc-cosine kernel
begin
    S_0 ← {Initial core vectors};
    c = c_MEB(S_0);
    r = r_MEB(S_0) ;
    S = S_0;
    while (There exist A ∈ T Such that φ̂_n^l(A) falls outside of the ball
    B(c, (1 + ϵ)r)) do
        Find a data sample A such that φ̂_n^l(A) is furthest away from c;
        Update S = S ∪ {A} ;
        Find new MEB(S) ;
        Update c = c_MEB(S);
        Update r = r_MEB(S)
```

choosing one data sample from each of the classes [89]. The process of finding new core vector necessitate the distance computation between the center c of current MEB and $\hat{\phi}_n^l(A_j)$, $\forall A_j \in T$. The data sample $A \in T$ in the feature space $\hat{\phi}_n^l(A)$ which is outside of the $(1+\epsilon)$ ball $(c, r(1+\epsilon))$ and furthest from c is taken as core vector and it is added to the core set. The distance between the center c of current MEB and $\hat{\phi}_n^l(A_j)$ can be computed by the equation:

$$\left\| c - \hat{\phi}_n^l(A_j) \right\|^2 = \sum_{A_m, A_N \in S} \alpha_m \alpha_N \hat{k_n^l}(A_m, A_N) - 2 \sum_{A_i \in S} \alpha_i \hat{k_n^l}(A_i, A_j) + \hat{k_n^l}(A_j, A_j) \quad (3.7)$$

In this equation, the feature mapping is carried implicitly by using the transformed multilayer kernel $\hat{k_n^l}$. It can be computed by the Equation 3.4 and its subsequences. The crux of this transformed kernel is the multilayer arc-cosine kernel. This deep layered arc-cosine kernel uses an incremental approach for its training process and it is detailed in Algorithm 3. For example, the arc-cosine kernel of depth 1 with degree n can be written by using the Equation 3.5 as:

$$k_n^1(x, y) = \frac{1}{\pi} [k_n^{(0)}(x, x) k_n^{(0)}(y, y)]^{\frac{n}{2}} AD(n, \theta_n^{(0)}) \quad (3.8)$$

where the base kernels $k^0(x, x) =< x, x >$ and $, k^0(y, y) =< y, y >$.

The training process in DCVM, except kernel computation is same as that of

core vector machines. Execution of each layer in this multilayer kernel depends on the given activation value. In the proposed DCVM, the kernel parameter is a structure L, represents the list of activation values corresponding to each layer of the hierarchical structure. The size of L determines the number of layers in arc-cosine kernel and hence the depth of DCVM. In machine leaning literature, any concrete method has not been reported for the selection of number of layers and list of activation values for the multi-layered arc-cosine kernel and still this exist as an open problem. It is a common practice to train the arc-cosine model with all the combination of selected range of activation values. The MEB approximation algorithm for deep core vector machine terminates when there is no data sample outside of the $(1 + \epsilon)$ ball. The high accuracy and low computational complexity of DCVM shows its high potential in the context of deep kernel machines.

3.2 Experimental Results

The performance of proposed deep core vector machines is illustrated in comparison with other related learning models such as CVM, SVM, DSVM and Deep Convolutional Neural Network(Deep CNN). The experiments on these models were attempted on seven bench mark datasets taken from both libSVM [137] and UCI machine learning repository [138]. It has been delineated in Table 3.1. All the method except Deep Convolutional Neural Network(Deep CNN) uses sparse representation of data.

Table 3.1: Composition of datasets used

Sl.No.	Dataset	#Clss	#Dim	#Train	#Test
1	Optdigit	9	64	3,823	1,797
2	Satimage	6	36	4,435	2000
3	Pendigit	10	16	7,494	3,498
4	Letter	26	16	15,000	5,000
5	Ijcnn1	2	22	49,990	91,701
6	News 20	20	62,061	15,935	3,993
7	KDD CUP 2010	2	29,890,095	19,264,097	748,401

3.2.1 Implementation

Publicly available C++ package, libCVM-2.2 [139] elucidates the implementation details of CVM. The basic implementation of multilayer arc-cosine kernel and its employment on support vector machine has been taken from the package libSVM-2.91 [140]. These two packages played vital roles in implementing the proposed Deep Core Vector Machine (DCVM). The radial basis function (rbf kernel) with $\gamma = \frac{1}{\text{number\_of\_features}}$ has been used to evaluate the performance of support vector machine. In our experiments, all the learning models such as CVM/BVM (libCVM-2.2), SVM & DSVM (libSVM-2.91) and proposed DCVM exercised with package specified default values for all its parameters except degree of arc-cosine kernel. Arc-cosine kernel with all the combinations of activation values 0 to 3 were attempted in DSVM and DCVM. A python library Keras with Theano as back end is used to implement Deep CNN. By fixing softmax as the output layer activation function, various layer-activation-optimization combinations were experimented with Deep CNN. The common attributes and its values [141] attempted in Deep CNN are listed in Table 3.2. All experiments were done on a server machine configured with AMD Opteron 6376 @ 2.3 GHZ / 16 core processor and 16 GB RAM. The generalization performance in terms of prediction accuracy (in %) and CPU time (in seconds) have been recorded in all the experiments.

TABLE 3.2: List of common attributes and its values used in Deep Convolution Neural Network Model

Attributes	Values
Loss function	categorical cross entropy
Convolution layer -filter Size	3×1
Max-pooling size	2×1
No.of Epochs	10
Batch size	10
Activation Functions tested	$\{relu, liear, sigmoid\}$
Optimizers tested	$\{sgd, adam, adagrad, adamax\}$

3.2.2 Performance Evaluation

This section illustrates the performance of proposed DCVM in comparison with other deep learning structures such as DSVM and Deep CNN. The performance of deep kernel learning structures (DCVM & DSVM) over its shallow based counterparts(CVM & SVM) were also studied. The experimental results of Deep CNN on various datasets are detailed in Table 3.3. The number of convolutional layers and number of filters used in each layer is shown in columns #Layers and Filters respectively. The activation function and optimization technique that qualify maximum accuracy (Acc) is shown in column Act and Optm respectively. The training time in second is stored in the column CPUtime. In the case of Kdd-Cup-2010 dataset, if the number of layers exceed 2, Deep CNN does not converges even after 5 days of its training. The prediction accuracy of

Table 3.3: The performance of Deep CNN in terms of accuracy and CPU times

Datasets	#Layers	Filters	Act	Optm	Acc	CPUtime
Optdigit	6	(64,64,128,128,256,256)	linear	sgd	97.01	303.21
Satimage	6	(64,64,128,128,256,256)	relu	adam	90.87	417.43
Pendigit	8	(64,64,128,128,128,256,256,256)	relu	adamax	98.04	627.82
Letter	8	(64,64,128,128,128,256,256,256)	linear	sgd	90.57	1022.72
Ijcnn1	8	(64,64,128,128,128,256,256,256)	relu	adamax	98.97	3312.7
News 20	7	(64,64,128,128,128,256,256)	linear	sgd	83.87	14630.68
Kdd-Cup-2010	2	(16,32)	linear	adam	43.25	112347.40

DCVM, the best among traditional CVM / BVM, DSVM, SVM and Deep CNN are given in Table 3.4. For the datasets 1- 5, the prediction accuracy of CVM/BVM and SVM were taken from the base paper [90]. For the remaining datasets, our experimental result on libCVM-2.2 and libSVM-2.91 packages were recorded. In columns DCVM & DSVM, the values in parenthesis denote the list of activation value of the multilayer arc-cosine kernel that bring out maximum accuracy. For example, in the case of Letter dataset, DCVM performs well in 2-layer arc-cosine kernel with degrees $[1, 0]$. Whereas DSVM perform well in 3-layer arc-cosine kernel with degrees $[0, 1, 2]$. The highest accuracy on each dataset is indicated by bold face values. The NT values in this table shows that there is no result within 5 days of training.

The scalability aspects of proposed DCVM is empirically evaluated by comparing its training time with that of DSVM on various datasets and it has been shown

TABLE 3.4: The generalization performance in terms of prediction accuracy of Deep CVM , CVM/BVM , Deep SVM, SVM and Deep CNN.

Dataset	DCVM	CVM/BVM	DSVM	SVM	DCNN
Optdigit	**97.72 (0)**	96.38	97.02(0,1)	96.77	97.01
Satimage	**92.15(0,1,2)**	89.60	90.35(0,1,2)	89.55	90.87
Pendigit	**98.37(0,1)**	97.97	97.94(0,1,2)	97.91	98.04
Letter	**96.94 (1,0)**	94.47	95.02(0,1,2)	94.25	90.57
Ijcnn1	**99.00 (0,1,2)**	98.67	97.99 (0,1)	98.97	98.97
News 20	**84.07 (0,1,2)**	79.61	83.92 (0,1,2)	72.38	83.87
Kdd-Cup-2010	**88.75 (0)**	63.83	NT	61.00	43.25

in Table 3.5. The training time shown in this table is the time taken by the respective models on each datasets against the accuracy shown in the Table 3.4. As in the case of accuracy tabulation, the NT values in this table also indicate there were no result within 5 days. Bold face values in this table represent minimum training time taken for each dataset.

TABLE 3.5: Training time of DCVM and DSVM on various datasets

Dataset	DCVM	DSVM	DCNN
Optdigit	**1.23**	2.02	203.21
Satimage	2.73	**2.14**	217.49
Pendigit	3.02	**2.90**	427.82
Letter	**28.01**	30.03	631.46
Ijcnn1	**316.97**	383.23	7312.71
News 20	**329.43**	403.09	14630.68
Kdd-Cup-2010	**95879**	NT	112347.40

The faster generalization performance of DCVM has been further illustrated by the graph shown in Figure 3.1. This graph plots the training time taken by DCVM & DSVM on different subsets of KDD-CUP-2010 dataset with varying sizes ($10^4, 10^5, 10^6$ and 10^7). All those trained models are tested against a common subset of KDD-CUP-2010 test dataset with size 25000. The training time for each subset represent the CPU time, recorded in second against highest accuracy.

The major observations from these experiments can be summarized as:

- In terms of accuracy, DCVM perform better than the best of CVM/BVM models, traditional deep support vector machines and Deep CNN.
- DCVM is much faster than DSVM on large datasets.

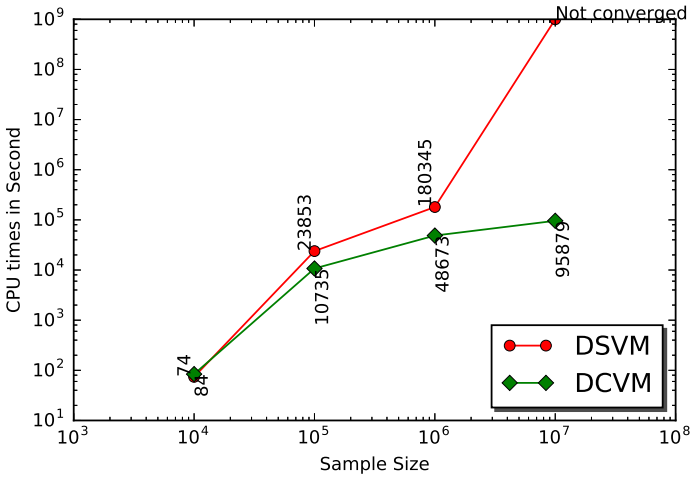

FIGURE 3.1: Training time of DSVM and DCVM on subsamples of KDD-cup-2010 datasets with different size

- With the increase in size of training samples , the growth rate of training time of DCVM is low when compared to that of DSVM.
- On KDD-CUP 2010, DSVM does not converge even after five days of execution when the sample size exceeds 10^6.
- On small datasets DSVM converges more quickly, because of its high sophisticated heuristics.

3.3 Conclusions

This work is a novel effort to build a scalable deep kernel machine. This new learning model facilitates the scalability by exploiting the scalable counterpart of SVM, the Core Vector Machine. The deep kernel computation in this model is accomplished by the well known multilayer kernel, the arc-cosine kernel. In this work we have analyzed the behavior of multilayer arc-cosine kernel and proved that in certain cases the arc-cosine kernel satisfies the required condition mandated by CVM. Various experiments on different datasets have been conducted to demonstrate the generalization performances in terms of prediction accuracy at a lesser training time when compared with other deep learning approaches. The experimental results show the potential of DCVM as a scalable deep kernel machine.

CHAPTER 4

Deep Multiple Multilayer Kernel Learning in Core Vector Machines

Over the last few years, we have been witnessing a dramatic progress of deep learning in many real world applications. The representation / feature learning plays a vital role in extracting useful information form raw data, for the successful implementation of deep learning algorithms. To take this journey further, it would be highly desirable to conceive more abstracted representation of data by embracing multiple layers of feature learning. Deep learning attempts to build high-level abstractions in data by exploiting learning models composed of multiple layers of non-linear transformations [25, 32]. Multilayer feature extraction and the capability of combining both supervised and unsupervised paradigms manifest the prominence of deep learning in the diverse machine learning arena. Deep learning approaches have been mainly pursuing in the area of neural network. There are a few attempts to impart deep learning features in kernel machines. A prominent invention in this direction is the modeling of Multilayer Kernel Machine (MKM) framework [49] which re-create multilayer representation / feature learning perceptiveness of deep learning architecture in kernel machines. Each layer in this multilayer learning structure utilizes KPCA to facilitate feature extraction. The fixed, solitary kernel computation involved in the KPCA may adversely affect the generalization performance of MKM framework. In addition to this fixed kernel computation, MKM also encounters scalability issues due to the high computational complexity caused by the final classifier, the Deep Support Vector Machine (DSVM).

It has been observed that, the limitation of fixed, solitary kernel computation can be well addressed by an approach employing a linear or convex combination of

base kernels called Multiple Kernel Learning (MKL) [114]. Conventional supervised MKL algorithms necessitate class labels to obtain the optimal kernel. However, recently developed unsupervised MKL algorithm [121] exhibit the potential to evaluate the linear or convex combination of multiple single-layered kernels, purely from unlabelled data. In the previous chapter (Chapter 3) it has been proved that the scalability issue in DSVM can be addressed by Deep Core Vector Machine (DCVM), the combination of arc-cosine kernel and Core Vector Machine (CVM).

Building on the above facts, this chapter attempt to build a scalable deep kernel machines with multiple layers of unsupervised feature extraction, by revamping the traditional Multilayer Kernel Machine. Each KPCA based feature extraction layer in this proposed framework is modeled by the combination of both single-layer and multilayer kernels in an unsupervised manner. Recently developed multilayer arc-cosine kernel can be used as multilayer kernel in MKL framework. Core vector machine with arc-cosine kernel is used as the final layer classifier which ensure the scalability in this model.

The rest of the chapter is organized as follows. Since the proposed model exploits unsupervised multiple kernel combination including both single-layer and multilayer kernels in its feature extraction layers, it is discussed in Section 4.1. The overall design of the proposed deep multi-layered kernel learning in core vector machines is discussed in Section 4.2. Experimental details and performance evaluations are discussed in Section 4.3. Section 4.4 present conclusions.

4.1 Unsupervised Multiple Kernel Learning with Single-layer and Multilayer Kernels

Conventionally, the kernel machines involves solitary static kernels for data representations. The selection of such static kernel seems to be expensive and an inappropriate kernel selection may causes the lessening of generalization performance of the machines. Multiple Kernel Learning (MKL) exhibit the potential to address this solitary kernel

learning problem. MKL provide an optimal kernel by learning a linear or convex combination on a set of predefined kernels [114]. These predefined kernels may represent different notion of similarity. MKL is not only a mechanism for reducing the burden of choosing right kernel but also for combining data from different sources such as audio,video and images. Let $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{N}$ be the training dataset, where $x_i \in R^d$ is the input feature vector and y_i is the unknown class label of x_i. The multiple kernel learning with a set of m predefined kernel functions $(k_p(.,.)_{p=1}^{m}$ and their combination parameterized by μ can be expressed as :

$$K_\mu(x_i, x_j) = F\left(\left\{k_p(x_i^p, x_j^p)\right\}_{p=1}^{m} | \mu\right) \tag{4.1}$$

where $F(.)$ is the combination function that map $\Re^m \rightarrow \Re$ and it can be linear, non-linear or data dependent. Kernel functions k_p take m feature representations of data instances $x_i = \{x_i^p\}_{p=1}^{m}$ and $x_j = \left\{x_j^p\right\}_{p=1}^{m}$; where $x_i^p, x_j^p \in \Re^{d_p}$. d_p is the dimensionality of the p^{th} feature representation. Traditional supervised multiple kernel learning approaches necessitate class labels of the training samples to obtain the optimal kernel. It does not seems to be feasible for many scenarios where the class labels may not be available. The availability of massive amount of unlabelled data and the capability of new learning approaches to handle unlabelled data, motivated the researchers to explore unsupervised approaches to combine predefined kernel functions. One of such innovative move in this direction is Unsupervised Multiple Kernel Learning (UMKL) [121]. It finds an optimal kernel combination of m predefined kernel functions, purely based on unlabelled data. The formulation of optimization problem [121] of unsupervised multiple kernel learning involves the following two intuitive principle:

- The optimal kernel $\mathcal{K}$ enables the reconstruction of each training sample x_i form its localized bases weighted by k_{ij}. It means, for each data sample x_i, the optimal kernel ($\mathcal{K}$) minimize the approximation error $\left\|x_i - \sum_j k_{ij} x_j\right\|^2$.
- $\mathcal{K}$ minimizes the distortion $\sum_{i,j} k_{ij} \left\|x_i - x_j\right\|^2$ over all training data.

In their formulation, the locality preserving principle has been accomplished by using a set of local bases for each data sample x_i denoted as $\mathcal{B}_i$. By combining the above

intuitions and facts they formulated UMKL as [121]:

$$\min_{k\in\mathcal{K},\mathcal{B}} \frac{1}{2}\sum_{i=1}^{N}\left\|x_i - \sum_{x_j\in\mathcal{B}_i} k_{ij}x_j\right\|^2 + \gamma_1\sum_{i=1}^{N}\sum_{x_j\in\mathcal{B}_i} k_{ij}\left\|x_i - x_j\right\|^2 + \gamma_2\sum_i |\mathcal{B}_i| \tag{4.2}$$

where γ_1 deals the trade-off between error and the locality distortion. γ_2 is used to control the size of local basis size. The above formulation can be simplified by constraining the local bases to fixed size B as :

$$\min_{k\in\mathcal{K},\mathcal{B}} \frac{1}{2}\sum_{i=1}^{N}\left\|x_i - \sum_{x_j\in\mathcal{B}_i} k_{ij}x_j\right\|^2 + \gamma_1\sum_{i=1}^{N}\sum_{x_j\in\mathcal{B}_i} k_{ij}\left\|x_i - x_j\right\|^2 \tag{4.3}$$

It can be expressed in matrix form as [121]:

$$\min_{\boldsymbol{\mu}\in\boldsymbol{\Delta},\mathbf{D}} \frac{1}{2}\left\|\mathbf{X}(\mathbf{I} - \mathbf{K}\circ\mathbf{D})\right\|_F^2 + \gamma_1 * \text{ tr } \mathbf{K}\circ\mathbf{D}\circ\mathbf{M}(\mathbf{11^T}) \tag{4.4}$$

In this expression, bold upper case letters denote matrices, bold lower case letters denote vectors. The i^{th} column in matrix $\mathbf{X}$ represent the data point x_i. $\mathbf{D} \in \{0,1\}^{N\times N}$ is a matrix in which each column vector $\boldsymbol{d_i}$ represent the local bases for x_i. This equation also enforce a constraint that $\|\boldsymbol{d_i}\|_1 = B, \text{for} i = 1, 2, \ldots, N$. The fixed constant $B << N$ is used to control the size of B_i for each x_i. $\boldsymbol{\Delta} = \left\{\boldsymbol{\mu} : \boldsymbol{\mu}^T\mathbf{1} = 1, \boldsymbol{\mu} \geq 0\right\}$ is the domain of a simplex. The matrix $\mathbf{M}$, the Euclidean distance on $\mathbf{X}$, is defined as : $[\mathbf{M}]_{ij} = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$. The notation '$\circ$' represent Hadamard product, element wise multiplication of two matrices, $\|\cdot\|_F^2$ denotes the Frobenius-norm of a matrix and 'tr' denotes the trace of a matrix. The kernel $\mathbf{K}$ is defined by $[\mathbf{K}]_{i,j} = \sum_{p=1}^{m}\mu_p k^p(x_i, x_j), 1 \leq i, j \leq N$.

The optimization task given in Equation 4.4 can be solved by an alternative optimization algorithm in which $\boldsymbol{\mu}$ and $\mathbf{D}$ are solved alternatively. In this approach, first, $\boldsymbol{\mu}$ is solved by fixing the variable $\mathbf{D}$. By ignoring all the constraints on $\mathbf{D}$, the objective function w.r.t $\boldsymbol{\mu}$ can be expressed as [121]:

$$J(\boldsymbol{\mu}) = \boldsymbol{\mu}^T\left(\sum_{t=1}^{m}\sum_{i=1}^{N} k_{t,i}k_{t,i}^T \circ \mathbf{d_i}\mathbf{d_i}^T \circ \mathbf{P}\right)^T\boldsymbol{\mu} + \mathbf{z}^T\boldsymbol{\mu} \tag{4.5}$$

where $[\mathbf{z}]_t = \sum_{i=1}^{N}(2\gamma\mathbf{v}_i \circ \mathbf{d}_i - 2\mathbf{b}_i \circ \mathbf{d}_i)^T \mathbf{k}_{t,i}$, $\mathbf{P} = \mathbf{X}^T\mathbf{X}$, and $\mathbf{k}_{t,i} = \left[k^t(x_i, x_1), \ldots, k^t(x_i, x_N)\right]^T$ is the i^{th} column of the t^{th} kernel matrix, b and v are columns of P and M corresponding to x_i respectively. γ_1 is abbreviated as γ.

In the second approach a greedy algorithm is used to solve $\mathbf{D}$, after fixing $\boldsymbol{\mu}$ and the kernel $\mathbf{K}$. Since the columns ($\mathbf{d}_i$) of $\mathbf{D}$ are independent of each others it can be solved separately [121] and the objective function w.r.t $\mathbf{d}$ can be expressed as:

$$J(\mathbf{d}) = \mathbf{d}^T(\mathbf{k}\mathbf{k}^T \circ \mathbf{b})\mathbf{d} + (2\gamma\mathbf{k} \circ \mathbf{v} - 2\mathbf{k} \circ \mathbf{p})^T\mathbf{d}$$

in this expression common subscripts of $\mathbf{d}, \mathbf{k}, \mathbf{p}, \mathbf{v}$ are omitted and it can be further simplified as [121]:

$$J(\mathbf{d}) = \mathbf{d}^T\mathbf{Q}\mathbf{d} + \mathbf{c}^T\mathbf{d}$$

where $\mathbf{Q} = \mathbf{k}\mathbf{k}^T$ and $c = 2\gamma\mathbf{k} \circ \mathbf{v} - 2\mathbf{k} \circ \mathbf{p}$. This method add one sample x^* into the base set B_i of x_i. The sample x^* that minimize the increase of $J(\mathbf{d})$ is selected, i.e.

$$x^* = \underset{x_j \in \mathcal{X} - B_i}{arg\min}\ 2\sum_{t:x_t \in B_i} Q_{tj} + Q_{jj} + c_j$$

Traditional unsupervised MKL model uses only single-layered kernels like Gaussian kernels, polynomial kernels etc. Recently developed arc-cosine kernels enable multi-layered kernel computation. The arc-cosine kernel has its own layered architecture and the computation in each layer depends on both the kernel values computed by the previous layer and the activation value in the current layer. As discussed in Chapter 2 (Section 2.4.1), the simplified form of n^{th} order arc-cosine kernel with angular dependency $AD(n, \theta)$ can be expressed as :

$$\mathcal{K}(x, y|n) = \frac{1}{\pi}\|x\|^n \|y\|^n AD(n, \theta) \tag{4.6}$$

where the angular dependency

$$AD(n, \theta) = (-1)^n(sin\theta^{2n+1})(\frac{1}{sin\theta}\frac{\partial}{\partial\theta})^n(\frac{\pi - \theta}{sin\theta}), \quad \text{for } \forall n = 1, 2, 3.. \tag{4.7}$$

Let $[n_1, n_2, ...n_l]$ be the list of activation values corresponding to the layers $1, 2, ..l$ respectively. Then the l layered arc-cosine kernel can be represented as:

$$\mathcal{K}^{(l)}_{(n1,n2..nl)}(x_i, x_j) = \langle \phi_{nl}(\phi_{nl-1}(...\phi_1(x_i))), \phi_{nl}(\phi_{nl-1}(...\phi_1(x_j)))\rangle \tag{4.8}$$

The multi layer arc-cosine kernel can be computed recursively as :

$$k^{(l+1)}_n(x, y) = \frac{1}{\pi}\left[k^{(l)}(x, x)\ k^{(l)}(y, y)\right]^{\frac{n}{2}} AD(n, \theta^{(l)}_n) \tag{4.9}$$

The layered kernel learning process start with initial kernels: $k(x_i, x_i) =< x_i, x_i >$, $k(x_j, x_j) =< x_j, x_j >$ and $k(x_i, x_j) =< x_i, x_j >$.

Now, the Unsupervised Multiple Multi-layered kernel Learning (UsMM$_l$KL) can be easily modeled by fabricating UMKL (Equation 4.3) with arc-cosine kernel. Since the execution of arc-cosine kernel depends on the degree of kernel at each layer, it may have different behavior on different execution instance of its iterative function. Each instance of arc-cosine kernel in UsMM$_l$KL framework attributes a set of kernels, depends on the nature and depth of activation list and contributes another recursive kernel computation ($[K]_{i,j} = \sum_{t=1}^{m} \mu_t k^t(.,.)$) in Equation 4.4. The depth of such recursive computation is determined by the size of activation list associated with each instance of arc-cosine kernel. Algorithm 3, discussed in Chapter 2, is utilized to compute the kernel matrix associated with the arc-cosine kernel. The process of computing kernel weight in an unsupervised multiple multi-layered kernel learning framework is summarized in Algorithm 6. Since our proposed model involves multiple layers, the alternating optimization strategy seems to be too costly and hence we chose to solve μ by fixing the variable D in advance. The matrix D is computed beforehand by taking B nearest neighbors of x_i from the training set. In this algorithm the activation list for each instance of arc-cosine kernel is passed as list of integers along with other kernel parameters, *parm*. The size of activation list governs the depth of recursive computation by the particular instance of arc-cosine kernel. b_i and v_i are columns of P and M corresponding to x_i respectively.

Algorithm 6: Unsupervised Multiple Multilayer Kernel Learning

$UsMM_lKL\ (X, K, param, D, \gamma)$

inputs : X : A matrix in which i^{th} column vector is the point x_i.
$K = \{k_1, k_2....k_m\}$: the set of 'm' predefined kernel functions. Param: Kernel parameters, $D \in \{0,1\}^{N \times N}$ is a matrix in which each column vector $\mathbf{d}_i$ represent the local bases for x_i, γ: Turning parameter

output: Optimal kernel weight μ

begin

$\mathbf{M}[i,j] \leftarrow x_i^T x_i + x_j^T x_j - 2x_i^T x_j$;

$\boldsymbol{\mu} \leftarrow 1/m$;

$\mathbf{P} \leftarrow \mathbf{X}^T\mathbf{X}$;

repeat

$\mathbf{W} \leftarrow \sum_{t=1}^{m}\sum_{i=1}^{N} \mathbf{k}_{t,i}\,\mathbf{k}_{t,i}^T \circ \mathbf{d}_i\mathbf{d}_i^T \circ \mathbf{P}$;

$[\mathbf{z}]_t \leftarrow \sum_{i=1}^{N} \left(2\gamma\mathbf{v}_i \circ \mathbf{d}_i - 2\mathbf{b}_i \circ \mathbf{d}_i\right)^T \mathbf{k}_{t,i}$;

$\boldsymbol{\mu} = arg\min_{\boldsymbol{\mu}} \left(\boldsymbol{\mu}^T\mathbf{W}\boldsymbol{\mu} + \mathbf{z}^T\boldsymbol{\mu}\right)$;

until *there is no more changes in* $\boldsymbol{\mu}$;

return $\boldsymbol{\mu}$;

This UsMM$_l$KL can be well fitted in KPCA for feature extraction. The following section explain the process of building scalable deep multiple multi-layered kernel machine by incorporating Unsupervised Multiple Multi-layered kernel Learning (UsMM$_l$KL) framework.

4.2 Deep Multiple Multilayer Kernel Learning in Core Vector Machines

Multilayer Kernel Machine (MKM) [49] opens up a new insight in leveraging multiple layers of feature extractions in kernel machines. In this architecture, the feature extraction layers were modeled with KPCA followed by a supervised feature selection method. In this model DSVM was usually used as the final layer classifier. This machine struggled with two limitations such as fixed kernel computation and scalability. The fixed kernel used in this architecture does not guarantee the optimum choice of the kernel for the task being modeled. The quadratic formulation of DSVM, in the output layer, impose a high computational complexity in many real world application that involve large size datasets. The following section exploring the possibilities to address these limitations of

MKM and proposes a new scalable deep kernel learning architecture with unsupervised deep multiple multi-layered kernel learning.

The first noticeable limitation in existing MKM, the fixed kernel computation can be solved by imparting Multiple Kernel Learning (MKL) approaches. The unsupervised MKL algorithm [121] tend to be an approach to deal with unlabelled dataset. The multilayer arc-cosine kernels open up a favorable opportunity to device MKL with multi-layered kernels. In considerations with these facts, the proposed model start with the formulation of unsupervised multiple multi-layered kernel learning (UsMM$_l$KL) framework. As discussed in Section 4.1, this unsupervised combination of single-layered and multi-layered kernel, can be accomplished by fabricating traditional unsupervised MKL with multilayer arc-cosine kernel. This UsMM$_l$KL frame work is then used in combine with KPCA for feature extraction. This revamped KPCA facilitates multiple multi-layered kernel computations. The multiple feature extraction layers can be formed by stacking this revamped KPCA in a hierarchical fashion.

The second problem of MKMs, the scalability aspects of final layer classifier can be well addressed by core vector machines. In CVM the quadratic optimization problem of SVMs has been reformulated as an equivalent minimum enclosing ball problem and then solved by a core set obtained by a faster $(1 + \epsilon)$ approximation algorithm. Deep core vector machines (DCVM) [142] can be modeled by leveraging multilayer arc-cosine kernel in core vector machines. The formulation of DCVM is well explained in Chapter 3. This DCVM can easily be used as the output layer classifier in the multi layer kernel machine which ensure scalability.

Finally, the existing MKM is restructured with multiple layers of revamped KPCA as feature extraction layers and DCVM as the output layer classifier. The overall architecture of the proposed framework is shown in Figure 4.1. This deep kernel learning architecture consists of multiple layers, with each layer performing feature extraction using revamped KPCA in an UsMM$_l$KL method followed by supervised feature selection. The supervised feature selection allows to retain only those set of features that are most relevant for the task being modeled. Like deep neural network learning architecture, after $\mathcal{L}$ layers of feature extraction, the refined data is given to the scalable deep core

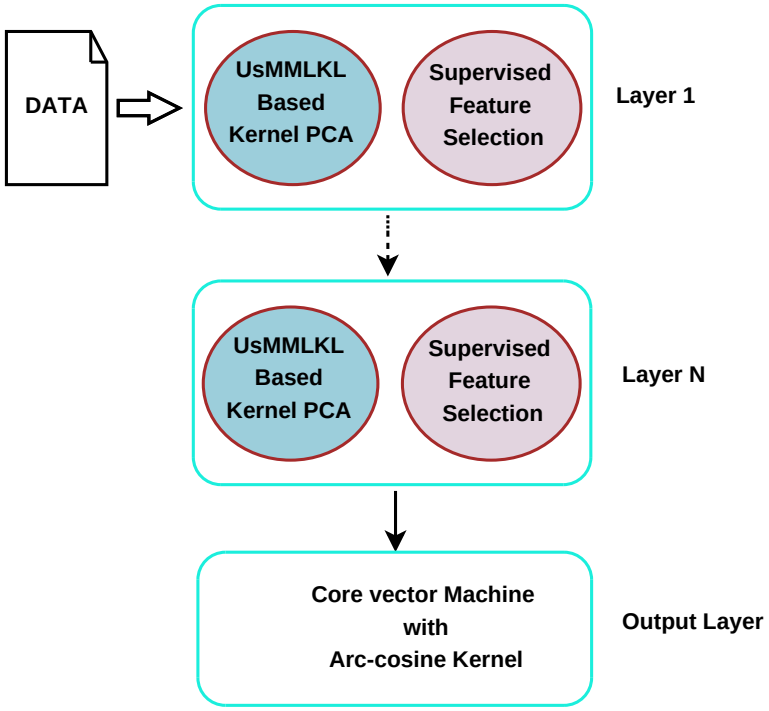

FIGURE 4.1: Deep core vector machines with multiple layers of feature extraction. Each kernel PCA based feature extraction layer is modeled by leveraging the convex combination of both single-layered and multi-layered kernels in an unsupervised manner.

vector machines. The proposed method is summarized in Algorithm 7. In this algorithm, the optimum kernel weight is computed by a sub function UsMM$_l$KL (), represents the unsupervised multiple multi-layered kernel learning model.

Algorithm 7: Deep core vector machines with unsupervised multiple multi-layered Kernel PCA

Deep_UMM$_l$KL $(X, y, \mathcal{L}, KL, Param, D, \gamma)$

inputs : X : A matrix in which i^{th} column vector is the point x_i, Y: true Labels, $\mathcal{L}$: Number of layers, KL: $\mathcal{L}$ x m matrix in which $\mathcal{K}^{(l)} = \{k_1^{(l)}, k_2^{(l)}, k_m^{(l)}\}$ is the list of kernels in layer l, kernel parameters Param, $D \in \{0,1\}^{N \times N}$ is a matrix in which each column vector $\mathbf{d}_i$ represent the local bases for x_i, Turning parameter γ.

output: Predicted labels Y'

begin

  foreach *l in range($\mathcal{L}$)* **do**

    $\mu^l \leftarrow$ UsMM$_l$KL$(X,\ \mathcal{K}^{(l)},\ Param^l,\ D,\ \gamma)$;

    $K_{new} \leftarrow \sum_{t=1}^{m} \mu_t^l * K_t^{(l)}$;

    $X\_kpca \leftarrow KPCA(K_{new})$;

    $X\_new \leftarrow$ selected features (X_kpca);

    $X \leftarrow X\_new$;

  Give the final set of features to the deep core vector machine ;

  return Y';

The proposed multilayer learning model combine kernels in an unsupervised manner and uses a supervised core vector machine as the final classifier. The main characteristic of this proposed model is that it exhibits the prominent features of deep neural networks such as the hierarchical representation of data and the capability for combining both supervised and unsupervised methods. However, unlike deep neural networks, this model ensure global optimum with its convex loss function. The other theoretical advantage of this model is the margin maximization, the key feature of SVM/CVM, that reduces the risk of over-fitting. The use of CVM as the final classifier in our model resolves the scalability problem of exciting MKMs to some extent. The main problem faced by our model is the multiple use of multilayer kernel which elevate the structural complexity of the model. The other noticeable issue of this approach is the restriction on the kernel functions possible ($k(x_i, x_i) = c$, a constant) due to the final classifier, the CVM.

4.3 Experimental Results

The datasets and their compositions used in our experiments are given in Table 4.1. It include both binary and multi-class datasets having varying size and dimensions. All the datasets except Ijcnn1 and Usps were taken from UCI machine learning repository [138] and Ijcnn1 and Usps were taken from libSVM repository [137]. The feature extraction

TABLE 4.1: The composition of datasets taken from both libsvm and UCI repositories

Sl.No.	Dataset	#Clss	#Dim	#Train	#Test
1	Australian Credit	2	8	460	230
2	Breast-cancer	2	10	400	283
3	Diabetes	2	8	512	256
4	Ijcnn1	2	22	49,990	91,701
5	Letter	26	16	15,000	5,000
6	Optdigit	9	64	3,823	1,797
7	Satimage	6	36	4,435	2000
8	Pendigit	10	16	7,494	3,498
9	Usps	10	256	7,291	2007

module, the kernel PCA based unsupervised multiple multilayer kernel learning has been implemented in python 2.7.6 and deep core vector machine (DCVM), the final

layer classifier, has been implemented in C++ [1]. The python package *subprocess* was used to invoke DCVM within the python framework.

In the training phase, we choose 2500 or entire training dataset, whichever is less, for cross validating the activation list of arc-cosine kernel from all the combination of degree $0, 1, 2$ and 3. The core vector machine with arc-cosine kernel was used as classifier in this case. The activation list corresponding to the maximum accuracy obtained in this validation phase were selected as the default activation list for arc-cosine kernel in the subsequent process. The Table 4.2 shows the selected activation list against each dataset.

TABLE 4.2: The activation list of arc-cosine kernel obtained during cross validation phase.

Sl.No	Dataset	Activation List	
		Unnormalized Data	Normalized Data
1	Australian Credit	[0,2]	[0,2]
2	Breast-Cancer	[0,1,2]	[0,2]
3	Diabetes	[0,1]	[0,2]
4	Ijcnn1	NA	[0,2]
4	Letter	[0,1,2]	[0,2]
5	Optdigit	[0,2]	[0,1,2]
6	Pendigit	[0,2]	[0,2]
7	Satimage	[0,1]	[0,1,2]
8	Usps	NA	[0,2]

In each layer of feature extraction module, we set apart 3000 or entire training data points, which ever is less, for computing the optimal kernel weight in an unsupervised manner. The list of kernels for the multilayer feature extraction were given as a matrix. The i^{th} row in this matrix represent the list of kernels for i^{th} layer of feature extraction module. Even though any kernel combinations can be used, we have used an arc-cosine kernel in between two RBF kernels as the kernel combination in each layer. After obtaining the optimal kernel value, the entire datasets were used for feature extraction. The feature selection was done by using univariate feature selection method available in scikit-learn[2] library [143]. Based on ranking produced by the selection method, top five percent of features were selected, since empirically it was giving

[1] DCVM has been implemented by combining the features from both the packages libCVM-2.2 [139] and libSVM-2.91

[2] Available at http://scikit-learn.org/stable/modules/feature_selection.html

a consistent performance. Final classification were carried out by CVM with arc-cosine kernel. In all the experiments, we have recorded the standard metric measures such as precision, recall, F1-Score and accuracy.

4.3.1 Performance Evaluation

The performance of proposed method has been evaluated on both normalized and unnormalized form of all the datasets except Ijcnn1 and Usps. The dataset name suffixed with the word *scaled* indicates normalized dataset.Three well known techniques such as *Standard Scalar, Robust Scalar* and *MinMax scalar* [143] were used for normalizing data samples and the variation in prediction accuracy has been shown in Table 4.3. We have experimented with up to 3 layers of feature extraction, for evaluating the performance of multilayer computation. The L1, L2 and L3 columns in Table 4.3 represent the prediction accuracy of each normalization techniques with one , two and three layers of feature extraction respectively. The prediction accuracy of proposed method on all the dataset without using any normalization techniques has been shown in Table 4.4. The variation in accuracy with different layers of feature extraction has been illustrated by the graph shown in Figure 4.2.

TABLE 4.3: Layer wise prediction accuracy of the proposed method with different normalization methods

Sl.No	Dataset	Standard Scaler			Robust Scaler			MinMax Scaler		
		L1	L2	L3	L1	L2	L3	L1	L2	L3
1	Australian Credit_scale	86.52	86.52	**86.96**	80.44	81.30	85.48	81.30	81.30	82.61
2	Breast-Cancer_scale	97.88	**98.23**	98.23	98.23	97.88	98.23	97.53	97.53	97.53
3	Diabetes_scale	72.66	73.44	73.04	71.88	73.44	**78.52**	74.60	76.56	77.73
4	Letter_scale	96.74	96.79	**97.73**	89.58	90.22	90.52	95.02	95.22	95.36
5	Optdigit_scale	97.66	97.72	97.99	96.60	96.99	97.50	99.05	99.11	**99.22**
6	pendigit_scale	99.40	99.54	99.57	99.29	99.37	99.40	99.40	99.54	**99.63**
7	satimage_scale	92.70	93.20	**93.50**	93.05	93.05	93.02	90.85	91.00	90.80

The classification report including average precision, recall, F-measures and the prediction accuracy of proposed method and deep core vector machine on both normalized and unnormalized dataset have been shown in Table 4.5. In this table, the classification report of proposed method represent the best among those layer wise performances shown in above tables. The bold face values indicate the highest accuracy on each dataset.

TABLE 4.4: The layer wise prediction accuracy with out using normalization method.

Sl.No	Dataset	Prediction Accuracy		
		L1	L2	L3
1	Australian Credit	67.83	74.94	**75.83**
2	Breast-Cancer	68.87	**71.94**	71.94
3	Diabetes	64.45	65.63	**70.70**
4	Letter	96.90	97.67	**98.08**
5	Optdigit	97.94	98.11	**98.57**
6	pendigit	99.06	**99.49**	99.49
7	satimage	88.45	**89.40**	89.10
8	ijcnn1 (Scaled to [-1 1])	99.20	99.37	**99.64**
9	Usps (scaled to [-1 1])	97.41	97.71	**98.06**

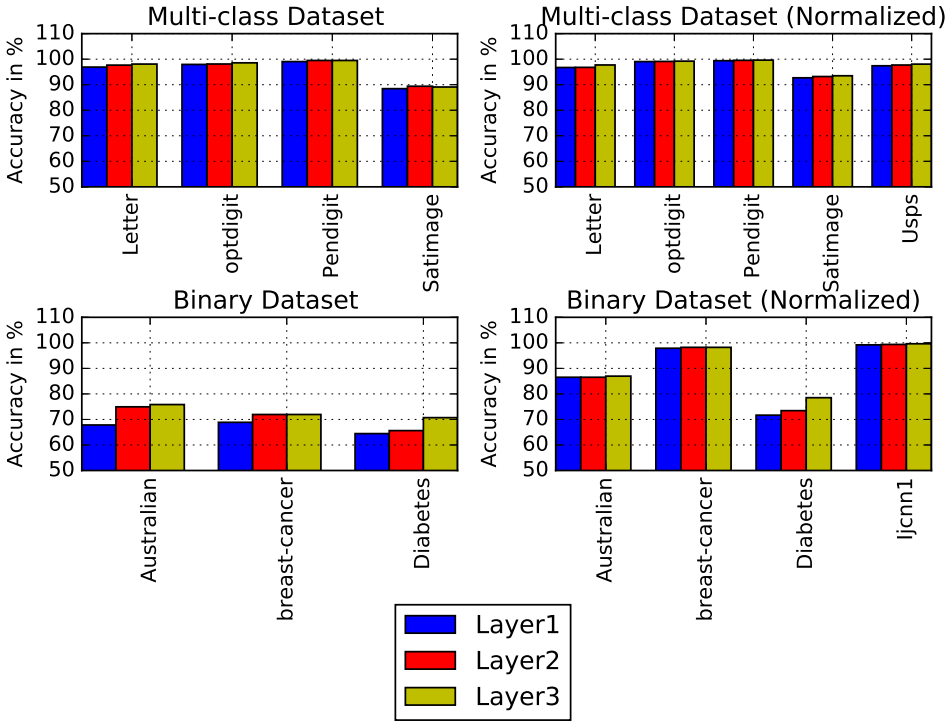

FIGURE 4.2: A comparison in accuracy with different number of feature extraction layers

From Table 4.5 & Figure 4.2, following observations have been made.

- In all the datasets, proposed method shows the best result in average precision, recall and F1-measures.
- Proposed algorithm consistently improves the accuracy of single layer deep core vector machines.

TABLE 4.5: The generalization performance in terms of Average Precision, Recall, F-core and overall prediction accuracy of Deep CVM and proposed method.

Sl.No	Dataset	DCVM				Proposed method			
		Preci	Reca	F1-sc	Acc	Preci	Reca	F1-Sc	Acc
1	Australian Credit	.754	.52	.753	75.21	.773	.758	.765	**75.83**
2	Breast-Cancer	.784	.675	.569	67.49	.793	.696	.741	**71.94**
3	Diabetes	.671	.684	.673	68.35	.694	.707	.697	**70.70**
4	Letter	.969	.969	.969	96.94	.981	.981	.981	**98.08**
5	Optdigit	.977	.977	.977	97.66	.987	.985	.986	**98.57**
6	pendigit	.984	.984	.984	98.37	.996	.995	.995	**99.49**
7	satimage	.855	.858	.852	85.80	.891	.894	.892	**89.40**
	Scaled dataset								
8	Australian Credit_scale	.857	.857	.857	85.65	.870	.870	.869	**86.96**
9	Breast-cancer_scale	.961	.961	.961	96.11	.983	.982	.982	**98.23**
10	Diabetes_scale	.760	.766	.757	76.56	.784	.785	.779	**78.52**
11	Letter_scale	.967	.967	.967	96.66	.981	.976	.978	**97.73**
12	Optdigit_scale	.978	.977	.977	97.72	.992	.992	.992	**99.22**
13	Pendigit_scale	.982	.982	.982	98.19	.996	.996	.996	**99.63**
14	Satimage_scale	.920	.921	.920	92.15	.935	.935	.934	**93.50**
15	Ijcnn_scale	.990	.991	.990	99.00	.993	.997	.995	**99.64**
16	Usps_scale	.952	.952	.952	95.22	.981	.981	.981	**98.06**

- In most of the datasets, the proposed algorithm achieves higher accuracies as the number of layers increased.
- As in the case of DCVM, proposed method also attains higher accuracies on normalized datasets.
- Proposed method also shows its potential in the context of multilayer unsupervised feature extraction, as in the case of deep learning models.

4.4 Conclusions

This work is a novel effort to build a scalable deep kernel machine with multiple layers of feature extractions. This model addressed the fixed kernel computation and scalability issues prevailing in existing MKM architecture. The unsupervised MKL formulation involving both single-layered and multi-layered kernels resolves fixed kernel computation. This unsupervised multiple muli-layered kernel learning framework can effectively be fitted in KPCA for feature extraction. Stacking of such layers in a hierarchical fashion constitute multiple layers of feature extraction in the proposed model. The utilization of Deep Core Vector Machine (DCVM) as the final layer classifier ensures scalability in

the proposed model. The empirical results show that the proposed method considerably outperform conventional core vector machine with arc-cosine kernel. The improvement in accuracy as the number layers increased shows the potential of multiple layers of feature learning. Hence the proposed learning architecture modeled with multiple layers of unsupervised feature extraction followed by a scalable final classifier can be observed as a novel and pragmatic deep kernel machine.

CHAPTER 5

Deep Kernel Learning in Extreme Learning Machines

The contemporary machine learning paradigms such as neural networks and support vector machines consistently emphasized on the requisite for learning parameter tuning in every layers for efficient generalization. However, there were some observations reflecting efficient learning without repeated parameter tuning [144]. The Extreme Learning Machine (ELM) is one such learning method that eliminate repeated parameter tuning. ELM was originally developed as a fast learning algorithm for Single Layer Feed-forward Networks (SLFN). The ELM has been later re-modeled with universal approximation and classification capabilities. The unified learning method of ELM allows using different types of feature mappings like random feature mappings and kernel methods [145]. Kernel based Extreme Learning Machines, one of the prominent research area in ELM, exploited kernel functions to tackle the situation in which the feature mapping functions of hidden nodes are unknown to the user [146]. A unified learning model for binary, multi-class classification and regression is proposed in this approach. Kernels such as linear kernel, polynomial kernel, Gaussian kernel, exponential kernel, etc. have been widely used in conjunction with extreme learning machines. However, these kernels constitute only solitary layered non-linear feature transformations in ELM. These shallow based ELM frameworks lack the potential in extracting complex structures and generating efficient representations from raw features. These limitations demand the necessity for utilizing deep learning paradigms with the escalated number of layers for efficient feature extractions.

The deep learning architecture, the recent advancement in artificial neural networks opens up a breakthrough in diverse artificial intelligence tasks[33, 34, 29, 28, 36, 147]. The deep learning approaches possess the potential to outperform contemporary shallow architectural models. The wide spread acceptance of deep learning in the realm of neural networks motivated the researchers to model kernel machines with deep learning capabilities. The multiple, multi-layered arc-cosine kernels [49] attributed deep learning features in the kernel based learning machines such as support vector machines. The multiple layers in arc-cosine kernels are used to achieve complex computations as in multi-layered neural network. Each layer in this hierarchy is defined with an activation value, referred as its degree, which determines the non-linear transformations in that layer. Arc-cosine kernels of different degrees have qualitatively different geometric properties. The arc-cosine kernels exhibit dominant potentialities in enforcing deep learning features in kernel based extreme learning machines.

This work is a novel effort to build deep learning algorithms with non-iterative parameter tuning capability by combining Extreme Learning Machine and arc-cosine kernels. The rest of this chapter is organized as follows. The mathematical formulation of extreme learning machines and its kernel version is explained in section 5.1. The proposed deep extreme learning machine and its computations are explained in section 5.2. Experimental analysis and performance evaluations are elucidated in section 5.3. Conclusions are conveyed in section 5.4.

5.1 Extreme Learning Machines

Extreme Learning Machines (ELM) was first proposed by Huang et al. [93], as a fast learning algorithm for SLFNs. The attractive features of these learning paradigm encompasses:

- Randomly generated input weights and hidden layer biases.
- The hidden layer parameters need not be tuned because they are independent of input samples.
- Non-iterative (Analytical) computation of output weights, as linear systems.

Single layered feed-forward neural networks with N distinct training samples $(x_i, y_i)_{i=1}^{N} | x_i \in \Re^n, y_i \in \Re^m$, can be depicted in Figure. 5.1. The input weight matrix $W_{N\times L}$ and the

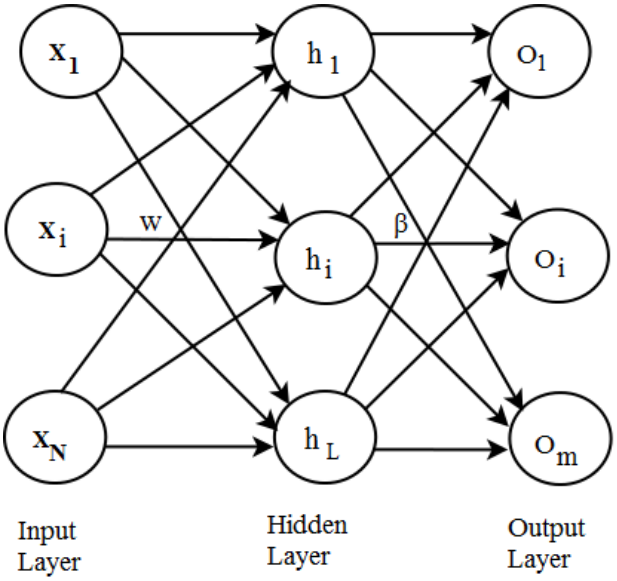

FIGURE 5.1: Basic model of Single Layer feed Forward Network

output weight matrix $\beta_{L\times m}$ can be represented as:

$$W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1L} \\ w_{21} & w_{22} & \dots & w_{2L} \\ \vdots & \vdots & \vdots & \vdots \\ w_{N1} & w_{N2} & \dots & w_{NL} \end{bmatrix}_{N\times L} \quad \beta = \begin{bmatrix} \beta_{11} & \beta_{12} & \dots & \beta_{1m} \\ \beta_{21} & \beta_{22} & \dots & \beta_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ \beta_{L1} & \beta_{L2} & \dots & \beta_{Lm} \end{bmatrix}_{L\times m}$$

The mathematical model of SLFN can be expressed as [94]:

$$\sum_{i=1}^{L} \beta_i A(w_i.x_j + b_i) = O_j, \quad \text{for j = 1 to N} \tag{5.1}$$

In this formulation, b_i represents bias of i^{th} hidden node and A()[1] be the activation of hidden nodes. The input weight vector w_i connects i^{th} hidden node and the input nodes whereas the output weight vector β_i connects i^{th} hidden node and output nodes. According to [93], there exist β_i, w_i and b_i such that actual output O_j equals to the

[1]The hidden layer neuron may have different type of activation function (sigmoid, hyperbolic tangent, threshold, etc.)

target output y_j, then SLFN approximate these N samples with zero mean square error.

$$\text{i.e.} \quad \sum_{j=1}^{N} \|O_j - y_j\| = 0$$

Now the Equation 5.1 becomes

$$\sum_{i=1}^{L} \beta_i A(w_i.x_j + b_i) = y_j \quad \text{, for j = 1 to N} \tag{5.2}$$

In simple form, the output function of SLFN can be expressed as:

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

and these N equations(one for each data sample) can be written in matrix form as:

$$f_L(x) = H\beta = Y \tag{5.3}$$

where H is a $N \times L$ matrix of the form

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_i) \\ \vdots \\ h(x_n) \end{bmatrix} = \begin{bmatrix} A(w_1.x_1 + b_1) & \dots & A(w_L.x_1 + b_L) \\ \vdots & \dots & \vdots \\ A(w_1.x_N + b_1) & \dots & A(w_L.x_N + b_L) \end{bmatrix} \tag{5.4}$$

"β" is the output weight matrix $(L \times m)$, which is of the form: $\begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}$ and "Y" is the output matrix $(N \times m)$. The conventional gradient based training method of SLFN fine tunes the parameters $w, b \ and \ \beta$ with respect to the objective function:

$$\min_{w,b,\beta} \|H\beta - Y\| \tag{5.5}$$

The corresponding cost function is [94]

$$E = \sum_{j=1}^{N} \left(\sum_{i=1}^{L} \beta_i h_i(x_j) - y_j \right) \tag{5.6}$$

where $h_i(x_j) = A(w_i.x_j + b_i)$. Different from traditional gradient based methods[2], ELM optimizes not only for the smallest error but also the smallest norm of output weights. The objective function of ELM [93] can now be expressed as:

$$\min_{\beta} \|H\beta - Y\|, \text{ subjected to } min \|\beta\| \tag{5.7}$$

The smallest norm least square solution of the Equation 5.7 can be obtained analytically as:

$$\hat{\beta} = H^{\dagger}Y \tag{5.8}$$

where $H^{\dagger}$ is the Moore-Penrose generalized inverse of matrix H. The $H^{\dagger}$ can be computed as: $H^{\dagger} = H^T(\frac{I}{C} + HH^T)^{-1}$, where "$C$" is the regularization parameter. The output function of ELM can be expressed as:

$$f(x) = h(x)\beta = h(x)H^T(\frac{I}{C} + HH^T)^{-1}Y \tag{5.9}$$

Usually the feature transformation function of ELM is known to the user. In the case of unknown feature mapping, the user can define kernel functions by applying mercer's theorem [146, 148]. The output function of ELM, Equation 5.9, then involves a dot product, HH^T. This dot product can easily be replaced with a kernel K_{ELM}. Each element in this kernel $K_{ELM}(i,j) = K(x_i, x_j) = \langle h(x_i), h(x_j) \rangle$. Equation 5.9 can be then modified for ELM with kernel function as:

$$f(x) = \begin{bmatrix} k(x, x_1) \\ \vdots \\ k(x, x_N) \end{bmatrix}^T (\frac{I}{C} + K_{ELM})^{-1}Y, \tag{5.10}$$

where $k_{ELM} = k(h(x_i), h(x_j))$

[2] The vector P of tunable parameters in the k^{th} iteration can be obtained as ($P_k = Pk - 1 \pm \eta \frac{\partial E(P)}{\partial P}$).

The solitary kernel functions including linear kernel, Gaussian kernel , polynomial kernel etc. have been extensively used in this formulation. These kernels map the input data in lower dimension to higher dimensional feature space. As shown in Equation 5.7, ELM attempts to achieve the minimum norm of its output weight vector β, besides the minimization of training error.

5.2 Deep Kernel Based Extreme Learning Machines

Traditional kernel based extreme learning machines exploited solitary kernels that constitute only single layer of feature extraction. It does not seems to be competent with many real world applications that involve highly complex and varying functions. This limitation necessitate the requirement of multi-layered feature extraction from the raw data. The new trend in machine learning is to explore the possibilities of equipping classifiers with deep learning capabilities, multiple layers of feature mapping. The multilayer kernel computation capability of arc-cosine kernel [49] enrich the process of extending deep learning features into kernel machines. The authors have attempted to explore deep arc-cosine kernels only on kernel machines like support vector machines that require iterative tuning of parameters. Extreme learning machine tends to become more popular by leveraging non-iterative parameter turning. By moving with new trends in machine learning , migrating shallow architectures into deep architectures, this section proposes a new deep kernel learning structure in the context of non-iterative learning paradigms. The proposed method exploits multilayer arc-cosine kernel and extreme learning machines, with comparable performance.

5.2.1 Building Deep Kernel based Extreme Learning Machines

The multilayer arc-cosine kernel has widely been used as the crux for building deep kernel machines. The main parameter of arc-cosine kernel, the activation list determines the depth and behavior (representation) of kernel computation. Unlike other kernels, the recursive mapping of arc-cosine kernel $\phi(...\phi(x))$ generates a qualitatively different representation than the original mapping $\phi(x)$. In this hierarchical computation, different nonlinear mappings (degree) and hence different kernel behavior can be attained at

different layers. This arc-cosine kernel has been evaluated only in the context of learning machines that necessitates iterative parameter turning.

Extreme learning machines and its kernel counterpart are fast machine learning approaches that facilitate non-iterative parameter tuning. ELM exploited kernel functions when the feature transformation function is unknown to the user. The formulation of kernel based extreme learning machine, as shown in Equation 5.10, constitutes only shallow based kernels such as RBF, polynomial,linear, etc.

In consideration with the above facts, a deep kernel based extreme learning machine is proposed by re-modeling the Equation 5.10 with "l"-fold arc-cosine kernel. Let $K_n^l()$ be the l fold arc-cosine kernel with activation value n. The mathematical model of Deep Kernel based Extreme Learning machine(DKELM) can be defined as:

$$f(x) = \begin{bmatrix} k_n^l(x, x_1) \\ \vdots \\ k_n^l(x, x_N) \end{bmatrix}^T \left(\frac{I}{C} + K_{ELM}\right)^{-1} Y, \tag{5.11}$$

where $k_{ELM} = k_n^l(h(x_i), h(x_j))$

Here $K_n^l(x_i, x_j)$ represents l- fold arc-cosine kernel with degree n. It can be computed iteratively as:

$$k_n^l(x_i, x_j) = \frac{1}{\pi}[k_n^{(l-1)}(x_i, x_i)k_n^{(l-1)}(x_j, x_j)]^{\frac{n}{2}} AD(n, \theta_n^{(l-1)}) \tag{5.12}$$

where $\theta^{(l)}$ is the angle between the images of x_i and x_j in the feature space induced by the l-fold composition.

$$\theta^{(l)} = cos^{-1}\left(\frac{k_n^{(l)}(x_i, x_j)}{\sqrt{k_n^{(l)}(x_i, x_i)k_n^{(l)}(x_j, x_j)}}\right) \tag{5.13}$$

$AD(n, \theta)$ is the angular dependency and its computation for the first three activation values are given in Algorithm 4. The detailed learning procedure for DKELM has been shown in Algorithm 8.

Algorithm 8: Deep kernel based extreme learning machine

DKELM $(H, [act - fact], Y)$

inputs : The Data matrix H,the list $[act - fact]$ contain activation factor for each layer in deep kernel. Y is the set of expected targets.

output: *Outputwt* ; The set of output weights used for prediction

begin

  for *i,j= 1 to num-rows(H)* **do**

    Ω_{ELM} = Deep_arc_cosine $(H_i, H_j, [act - fact])$;

  $Outputwt = (\frac{I}{C} + \Omega_{ELM})^{-1}.Y$;

return *Outputwt*;

In this algorithm, the sub procedure *Deep_arc_cosine* exploits multilayer arc-cosine kernel (Algorithm 3) to computes the deep kernel matrix Ω_{ELM}. The depth of recursive computation is determined by the size of given activation list. The feature extraction at different layers of arc-cosine kernel depends on the activation value for that layer and the kernel value computed in the previous layer. The multilayer arc-cosine kernels and its computations are elaborated in Section 2.4.1.

Once deep kernel matrix Ω_{ELM} is computed, the DKELM algorithm uses a simple generalized matrix inverse operation on deep kernel matrix. The generalized matrix inverse operation tend to be a linear solver to compute the output weights($Line3$ in Algorithm 8), similar to Lagrangian multipliers in SVM. The output weight vectors are used to predict the class label for training and testing dataset as:
$Predicted\_Targets =< \Omega_{ELM}, Outputwt) >$. In the case of testing, Ω_{ELM} represents the kernel matrix corresponding to the test dataset. The use of multilayer arc-cosine kernel pave the way to bring deep kernel learning capability in kernel based extreme learning machine. The comparable performance of DKELM in terms of accuracy and low computational complexity opens up a new learning algorithm in the context of deep kernel machines.

5.3 Experimental Results

This section elaborates the experiments conducted to show the improved generalization performance of the proposed deep kernel based extreme learning machines. The choice of datasets for the experiments are inspired by the paper [146] an they are utilized to

test the performance of deep kernel based ELM over shallow kernel based ELM. These dataset have been taken from both LIBSVM [137] and UCI Machine Learning Repository [138]. Dataset composition has been shown in Table 5.1.

TABLE 5.1: Details of Dataset used

Sl.No	Dataset	#Dim	#Train	#Test	#class
1	Australian Credit	6	460	230	2
2	Colon	2000	30	32	2
3	Diabetes	8	512	256	2
4	Glass	9	142	72	6
5	Iris	4	100	50	3
6	Leukemia	7129	38	34	2
7	Letter	16	13333	6667	26
8	Liver	6	230	115	2
9	Mushroom	22	1500	6624	2
10	Satimage	36	4435	2000	6
11	Segment	19	1540	770	7
12	Wine	13	118	60	3

The proposed DKELM has been implemented in python 2.7.6 by extending the kernel based ELM [3] with deep multi-layered arc-cosine kernel [4]. In all dataset, the experiments were repeated for ten different train - test combinations. In each combination, three scaling mechanisms such as standard scalar, robust scalar and minmax scalar methods were carried out. Also, all combinations of arc-cosine kernel with degree 0,1,2 and 3 were evaluated. Deep Support Vector Machine (DSVM) has been implemented by using the SVM module available in sklearn package (python 2.7.6) with arc-cosine function as the custom kernel. All experiments were done on a server machine configured with AMD Opteron 6376 @ 2.3 GHZ / 16 core processor and 16 GB RAM. In all the experiments, precision, recall, F1-score, prediction accuracy (in %) and CPU time (in seconds) have been recorded.

5.3.1 Performance Evaluation

A comparative analysis in terms of prediction accuracy of kernel based ELM (column KELM) and proposed DKELM method (column DKELM) is elucidated in Table 5.2.

[3]The ELM package elm-0.1.1 is available at https://pypi.python.org/pypi/elm

[4]libSVM-2.91:Available at
https://www.csie.ntu.edu.tw/ cjlin/libsvm/oldfiles/

The prediction accuracy of kernel based ELM (Gaussian kernel) were taken from the paper[146].

TABLE 5.2: Generalization performance in terms of prediction accuracy of Kernel based ELM(KELM) and proposed Deep kernel based ELM (DKELM)

Dataset	KELM	DKELM
Australian Credits	86.29	**90.90**
Colon	84.38	**93.81**
Diabetes	77.52	**80.70**
Glass	68.41	**73.62**
Iris	96.04	**99.01**
Leukemia	82.35	**98.80**
Letter	97.41	**97.80**
Liver	72.14	**79.14**
Mushroom	88.84	**99.30**
Satimage	92.35	**93.03**
Segment	96.53	**97.10**
Wine	98.48	**99.63**

Experiments are attempted on Deep kernel based frameworks such as deep support vector machine(column DSVM) and proposed deep kernel based Extreme learning machine. A comparative analysis in terms of prediction accuracy of deep support vector machine(column DSVM) and the proposed DKELM method (column DKELM) is elucidated in Table 5.3.

TABLE 5.3: Generalization performance in terms of prediction accuracy of Deep Support Vector Machine (DSVM) and proposed Deep kernel based ELM (DKELM)

Dataset	DSVM	DKELM
Australian Credits	84.03	**90.90**
Colon	84.12	**93.81**
Diabetes	75.83	**80.70**
Glass	71.69	**73.62**
Iris	96.89	**99.01**
Leukemia	90.05	**98.80**
Letter	95.02	**97.80**
Liver	70.19	**79.14**
Mushroom	93.31	**99.30**
Satimage	90.35	**93.03**
Segment	95.47	**97.10**
Wine	94.45	**99.63**

In addition to the above analysis, a comparison in terms of prediction accuracy of existing models and proposed deep kernel learning models has been done and it is shown in Table 5.4. This table includes a comparison between existing models,

viz; Core Vector Machine (column CVM), Kernel based Extreme Learning Machines (column KELM) and proposed deep kernel learning models, viz; Deep Core Vector Machines (column DCVM), Deep Multiple Multilayer Kernel learning in CVM (column DMM_lKLCVM) and Deep Kernel based Extreme Learning Machine (column DKELM). The performance of CVM on all the datasets are evaluated by using the package libCVM-2.2 [139].

TABLE 5.4: A comparison in generalization performance of existing models and proposed models

Dataset	Existing Models		Proposed Models		
	CVM	KELM	DCVM	DMM_lKLCVM	DKELM
Letter	94.47	97.41	96.94	98.08	97.80
Satimage	89.60	92.35	92.15	93.50	93.03
Australian Credits	79.83	86.29	86.65	86.95	90.90
Diabetes	73.62	77.52	76.56	78.52	80.70

Table 5.5 represents average precision, recall and F1-score computed on each dataset for the proposed Deep kernel based ELM (DKELM). The scaling mechanism and activation list of arc-cosine kernel that brings out maximum accuracy in deep learning frameworks such as DSVM and DKELM has been shown in Table 5.6.

TABLE 5.5: The average precision, Recall and F1-score of DKELM on various datasets

Dataset	Precision	Recall	F1-score
Australian Credits	0.907	0.914	0.910
Colon	0.938	0.938	0.938
Diabetes	0.818	0.798	0.808
Glass	0.786	0.694	0.734
Iris	0.999	0.991	0.990
Leukemia	0.988	0.989	0.988
Letter	0.980	0.970	0.979
Liver	0.793	0.791	0.788
Mushroom	0.997	0.991	0.994
Satimage	0.934	0.918	0.926
Segment	0.973	0.971	0.972
Wine	0.998	0.992	0.990

A comparison in training time taken by DSVM and DKELM has been shown in the Table 5.7. A comparative analysis of Deep support vector machine (DSVM) and deep kernel based ELM (DKELM) in terms of training time is elucidated in Table 5.7.

TABLE 5.6: Activation list and Scaling mechanism that brings out maximum accuracy in DKELM and DSVM

Dataset	DKELM		DSVM	
	Activation List	Scaling	Activation list	Scaling
Australian Credits	[2,0,3,1]	Robust scalar	[2,0,3,1]	Robust scalar
Colon	[3,2,1,0]	Minmax Scalar	[2,3,1,0]	Minmax scalar
Diabetes	[0,1]	Robust scalar	[0,1,3]	Standard scalar
Glass	[1]	Standard Scaler	[0, 1, 2]	Standard scalar
Iris	[1]	Minmax scalar	[1]	Robust scalar
Leukemia	[3, 2, 1, 0]	Minmax scalar	[3, 2, 1, 0]	Minmax scalar
Letter	[1]	Minmax scalar	[0, 1]	Standard scalar
Liver	[0,1]	Standard Scaler	[0, 1, 2]	Minmax scalar
Mushroom	[3, 2, 1, 0]	Standard Scaler	[3, 2, 1, 0]	Standard scalar
Satimage	[0, 2]	Robust scalar	[0, 2]	Robust scalar
Segment	[1]	Minmax scalar	[1, 0]	Minmax scalar
Wine	[3, 1, 2, 0]	Minmax scalar	[2, 1, 3, 0]	Minmax scalar

TABLE 5.7: Training time (in seconds) of Deep Support Vector Machine (DSVM) and Deep kernel based ELM (DKELM)

Dataset	DSVM	DKELM
Letter	148.503	**66.785**
Mushroom	3.769	**1.869**
Satimage	15.984	**7.013**
Segment	1.990	**1.021**
Colon	0.06	**0.06**
Leukemia	0.049	**0.041**
Iris	0.02	**0.02**
Wine	0.023	**0.014**
Glass	0.005	**0.004**
Australian Credits	0.189	**0.182**
Diabetes	0.082	**0.079**

The improved performance in terms of training time of DKELM over DSVM has been further illustrated in Figure 5.2.

The following observations can be made from the above experimental results.

- From Table 5.3, it is clear that proposed DKELM attains improved accuracy on KELM.
- The prediction accuracy given in Table 5.3 and the average precision, recall and F1-score given in Table 5.5 show that the proposed DKELM consistently improves the accuracy of DSVM.

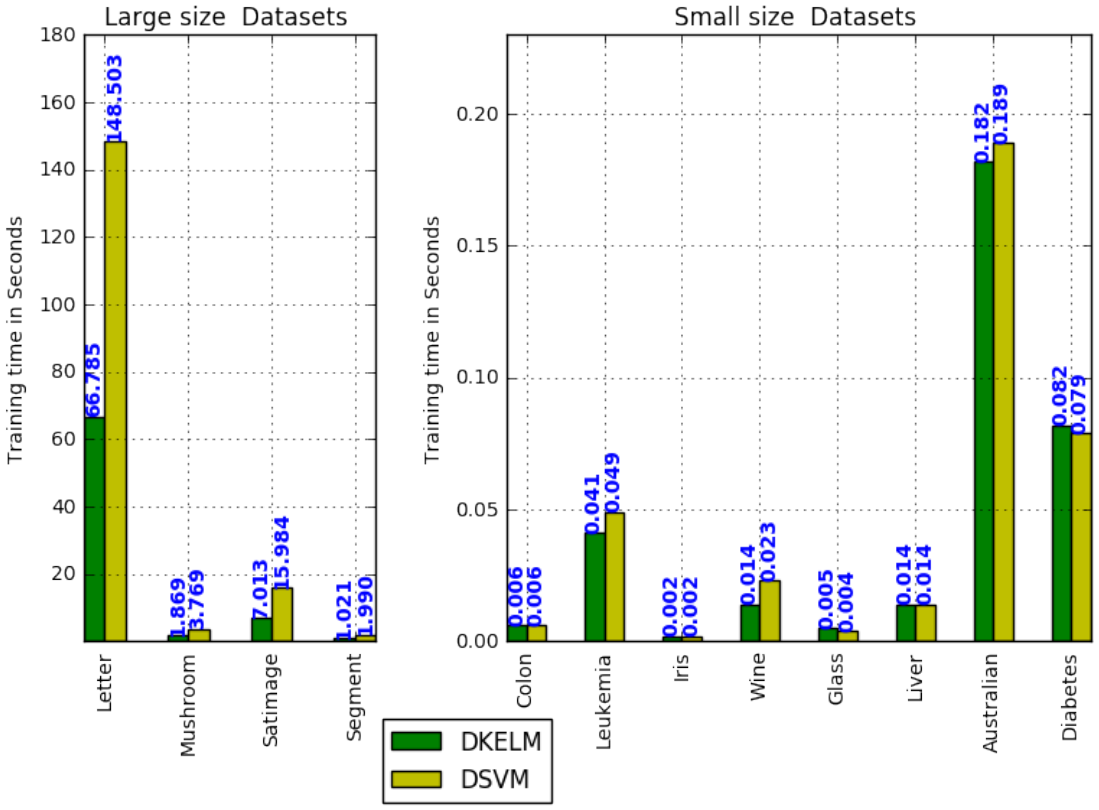

FIGURE 5.2: A comparison in training time (in seconds) between DKELM and DSVM

- The prediction accuracy given in Table 5.4 show that the proposed deep kernel learning models consistently improves the existing models.
- From Figure 5.2, it can be seen seen that the difference in training time required for DSVM and DKELM increases as the size of dataset is increased.
- For the large dataset 'letter', proposed DKELM runs more than twice as faster than DSVM.
- The improved accuracy and less training time shows that the proposed deep kernel based ELM has high potential in the context of deep kernel machines.

5.4 Conclusions

Extreme learning machines (ELM) tend to be an initiative towards eliminating iterative parameter tuning, which is a trivial issue faced by neural networks and kernel machines. The ELM eliminates this overwhelming process by enforcing analytical computation of

output weights. This is achieved by the utilization of random weight assignments in the hidden nodes. ELM was originally developed as a fast learning algorithm for single layer feed-forward neural networks (SLFN). Later, it has been re-modeled with universal approximation and classification capabilities. Kernel based ELM is one among different models of ELM which uses the kernel trick for the feature transformation. The existing kernel based ELMs are using single layer of kernel functions and hence belongs to the class of shallow architectures. This research explores a new deep kernel based learning model with extreme learning machine and arc-cosine kernels. Arc-cosine kernels possess the potential to mimic deep layer frameworks in shallow architectures. The multi-layer feature transformation by the arc-cosine kernel and the fast training mechanism without iterative parameter tuning adopted by the extreme learning machine turns out to be the prominent features of DKELM. The performance of proposed DKELM is compared with solitary layered KELM and DSVM. In all cases, proposed model shows improved accuracy. In the case of training time, proposed model converges extremely fast as compared to that of deep support vector machines.

CHAPTER 6

Conclusions and Future Works

6.1 Conclusions

In the recent past, deep learning architectures have been making tremendous progress in the context of enabling pragmatic applications of machine intelligence. The deep learning models attempt to discover abstract representation of training data by exploiting multiple layers of feature transformations. The modern hardware technologies coupled with algorithmic advancements in optimization techniques to solve complex learning problems make deep learning more attractive and powerful when compared to traditional shallow learning architectures. Deep learning originated in the context of neural networks and have made many success stories in the field of image prediction, language translation, signal processing, action recognition, face recognition etc. Emergence of unsupervised, greedy layer-wise learning procedure bestow an efficient way of training deep neural network learning architectures. Deep Belief Networks, Convolutional Neural Networks, Recurrent Neural Networks, etc. are some of the well known deep learning architectures.

The astounding performance of deep neural network learning motivated the researchers to shift their attention towards imparting deep learning capabilities in other learning approaches, particularly in kernel machines. The elegant characteristics that favor kernel machines over other models are:

- Strong theoretical foundation.

- The utilization of convex loss functions ensure globally optimal solutions.
- Structural risk minimization : structural complexity of the model is learned from data. for example, in SVM the model complexity is controlled automatically by adjusting the number of support vectors during training.
- Some kernel machines (SVM and its variants) relies on the concepts of margin maximization and hence reduce the risk of over-fitting.

Recently developed arc-cosine kernel exhibits capability for multiple layers of kernel computations / feature extractions. It is widely used in conjunction with SVMs and produces promising results in many real world applications. Multilayer Kernel Machine (MKM) was another attempt in kernel machines towards multiple layers of feature extraction. These contributions open up an extendable research avenue in building advanced techniques in kernel machines. From the literature review, it is learned that the scalability, multiple multilayer kernel learning, unsupervised feature extraction by exploiting multiple kernel learning, etc. were unexplored in the context of deep kernel machines.

This research is an effort to build deep structured scalable kernel machine architectures similar to deep neural network for supervised classification. This research explored the possibility of enhancing kernel machines with deep learning principles. Some of the dimensions explored in this work are (i) Feasibility of building a scalable deep kernel machine similar to SVM like machines. (ii) Scalable deep kernel machines with multiple layers of unsupervised feature extraction. (iii) Deep kernel learning in non-iterative learning approaches like extreme learning machines. In this direction, three deep kernel learning architectures were proposed.

- Deep learning in core vector machines : It modeled a scalable deep kernel machines by combining arc-cosine kernel and Core Vector Machine. This research analyzed the behavior of multilayer arc-cosine kernel and proved that in certain cases the arc-cosine kernel satisfies the required condition mandated by CVM and thus proposed Deep Core Vector Machine algorithm. Experimental results show that the proposed DCVM outperforms traditional CVM, DSVM and deep CNN.

- Deep Multiple Multilayer Kernel Learning in Core Vector Machines: This work is an effort to build a scalable deep kernel machines with multiple layers of unsupervised feature extraction. In this model feature extraction is accomplished by KPCA, similar to MKM. KPCA in this proposed model is modeled as a MKL framework by combining both single-layered and multi-layered kernels in an unsupervised manner. Recently developed multilayer arc-cosine kernels ensure multilayered kernels in MKL framework. Core vector machine with arc-cosine kernel is used as the final layer classifier which ensure the scalability in this model.The empirical results show that the proposed method considerably outperform conventional core vector machine with arc-cosine kernel.

- Deep kernel based extreme learning machines: It is an attempt to build deep kernel learning features in fast non-iterative learning algorithms. It combines the multilayer feature transformation of arc-cosine kernel and the fast non-iterative training mechanism of ELM. The experiments in this models shows its out performance over conventional kernel based extreme learning machines.

This thesis attempted to build scalable kernel machines with deep learning capabilities as discussed above. From the theoretical analysis and experimental results, it is observed that the proposed methods are promising candidates in the context of deep kernel machines.

6.2 Future Works

The theoretical and empirical analysis done through this thesis work unearth the following avenues for further explorations.

- The generalization of deep core vector machines to the multi-view version for handling the input data with multiple feature representations is a potentially explorable problem.

- The generalization performance of 'Deep Multiple Multilayer Kernel Learning in Core Vector Machines' is highly influenced by the structure of multilayer MKL

framework. The number of layers and appropriate base kernels contribute this multilayer MKL structure. A theoretical study on the optimal structure of multilayer MKL model is a potential future direction

- A study on feasibility of deep multilayer kernel learning approach on structured output prediction task that involves very complex decision function is another interesting research problem.

- The multiple kernel learning based feature extraction involves extensive amount of matrix manipulation. The scalability aspects in this context is another potential future direction.

- The deep learning algorithms are found to be effective for memory related neural network models such as Recurrent neural network, Memory network, Turing neural network etc. A study on the formulation of recurrent kernels that mimics the above models is another explorable direction.

Bibliography

[1] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[2] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[3] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533, 1986.

[4] Vladimir Vapnik. *Statistical learning theory.* Wiley, New York, 1998.

[5] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[6] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines: And Other Kernel-based Learning Methods*. Cambridge University Press, New York, NY, USA, 2000.

[7] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[8] Shun-ichi Amari and Si Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[9] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

[10] K-R Muller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernhard Scholkopf. An introduction to kernel-based learning algorithms. *IEEE transactions on neural networks*, 12(2):181–201, 2001.

[11] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[13] Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4):18–28, 1998.

[14] Jeffrey D Hart and Thomas E Wehrly. Kernel regression estimation using repeated measurements data. *Journal of the American Statistical Association*, 81(396):1080–1088, 1986.

[15] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

[16] Hiroyuki Takeda, Sina Farsiu, and Peyman Milanfar. Kernel regression for image processing and reconstruction. *IEEE Transactions on image processing*, 16(2):349–366, 2007.

[17] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002.

[18] Gerhard Jäger. Maximum entropy models and stochastic optimality theory. *Architectures, rules, and preferences: variations on themes by Joan W. Bresnan. Stanford: CSLI*, pages 467–479, 2007.

[19] Charles Sutton and Andrew McCallum. *An introduction to conditional random fields for relational learning*, volume 2. Introduction to statistical relational learning. MIT Press, 2006.

[20] Charles Sutton, Andrew McCallum, et al. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[21] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[22] Keinosuke Fukunaga and L Hostetler. Optimization of k nearest neighbor density estimates. *IEEE Transactions on Information Theory*, 19(3):320–326, 1973.

[23] Padhraic Smyth. Clustering sequences with hidden markov models. In *Advances in neural information processing systems*, pages 648–654, 1997.

[24] Ara V Nefian and Monson H Hayes. Hidden markov models for face recognition. In *Acoustics, Speech and Signal Processing, Proceedings of the IEEE International Conference on*, volume 5, pages 2721–2724. IEEE, 1998.

[25] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[26] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[27] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[28] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

[29] Geoffrey E Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434, 2007.

[30] Li Deng. Three classes of deep learning architectures and their applications: a tutorial survey. *APSIPA transactions on signal and information processing*, 2012.

[31] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[32] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[33] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[34] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[35] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pages 1137–1144. MIT Press, 2006.

[36] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *AISTATS*, volume 1, page 3, 2009.

[37] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008.

[38] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[39] Jun Deng, Zixing Zhang, Erik Marchi, and Bjorn Schuller. Sparse autoencoder-based feature transfer learning for speech emotion recognition. In *Affective Computing and Intelligent Interaction (ACII), Humaine Association Conference on*, pages 511–516. IEEE, 2013.

[40] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Tara N Sainath. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[41] Yajie Miao, Florian Metze, and Shourabh Rawat. Deep maxout networks for low-resource speech recognition. In *Automatic Speech Recognition and Understanding (ASRU), IEEE Workshop on*, pages 398–403. IEEE, 2013.

[42] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[43] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[44] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1110–1118, 2015.

[45] Sheng-hua Zhong, Yan Liu, and Yang Liu. Bilinear deep learning for image classification. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 343–352. ACM, 2011.

[46] Harris Drucker, Christopher JC Burges, Linda Kaufman, Alex J Smola, and Vladimir Vapnik. Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161, 1997.

[47] Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6(Apr):363–392, 2005.

[48] Roman Rosipal, Mark Girolami, Leonard J Trejo, and Andrzej Cichocki. Kernel PCA for feature extraction and de-noising in nonlinear regression. *Neural Computing & Applications*, 10(3):231–243, 2001.

[49] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

[50] Andre Wibisono, Jake Bouvrie, Lorenzo Rosasco, and Tomaso Poggio. Learning and invariance in a family of hierarchical kernels. 2010.

[51] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 370–378, 2016.

[52] Chih-Chieh Cheng and Brian Kingsbury. Arccosine kernels: Acoustic modeling with infinite neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5200–5203. IEEE, 2011.

[53] Youngmin Cho and Lawrence K Saul. Large-margin classification in infinite neural networks. *Neural computation*, 22(10):2678–2697, 2010.

[54] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[55] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

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

[57] Andrew L. Beam. Deep learning 101 - part 1: History and background. 2017.

[58] G Dahl, D Yu, L Deng, and A Acero. Context-dependent dbn-hmms in large vocabulary continuous speech recognition. In *Proc. ICASSP*, 2011.

[59] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[60] Itamar Arel, Derek C Rose, and Thomas P Karnowski. Deep machine learning-a new frontier in artificial intelligence research. *IEEE computational intelligence magazine*, 5(4):13–18, 2010.

[61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[63] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[64] Joel Heck and Fathi M Salem. Simplified minimal gated unit variations for recurrent neural networks. *arXiv preprint arXiv:1701.03452*, 2017.

[65] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009.

[66] Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

[67] Mikael Bodén and Janet Wiles. Context-free and context-sensitive dynamics in recurrent neural networks. *Connect. Sci.*, 12:197–210, 2000.

[68] Christopher M.. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[69] Marc G Genton. Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312, 2001.

[70] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[71] Jason Weston and Chris Watkins. Support vector machines for multi-class pattern recognition. In *ESANN*, volume 99, pages 219–224, 1999.

[72] Asa Ben-Hur, David Horn, Hava T Siegelmann, and Vladimir Vapnik. Support vector clustering. *Journal of machine learning research*, 2(Dec):125–137, 2001.

[73] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137–142, 1998.

[74] Edgar Osuna, Robert Freund, and Federico Girosit. Training support vector machines: an application to face detection. In *Computer vision and pattern recognition, Proceedings, IEEE computer society conference on*, pages 130–136. IEEE, 1997.

[75] FJ Huang and Y LeCun. Large-scale learning with SVM and convolutional nets for generic object categorization. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.

[76] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104. ACM, 2004.

[77] Vladimir Vapnik, Steven E Golowich, Alex Smola, et al. Support vector method for function approximation, regression estimation, and signal processing. *Advances in neural information processing systems*, pages 281–287, 1997.

[78] You Ji and Shiliang Sun. Multitask multiclass support vector machines: model and experiments. *Pattern Recognition*, 46(3):914–924, 2013.

[79] Ivor W Tsang, James Tin-Yau Kwok, and Pak-Ming Cheung. Very large SVM training using core vector machines. In *AISTATS*, 2005.

[80] Guanglu Zhou, Jie Sun, and Kim-Chuan Toh. Efficient algorithms for the smallest enclosing ball problem in high dimensional space. *Novel Approaches to Hard Discrete Optimization*, 37:173, 2003.

[81] Bernd Gärtner. Fast and robust smallest enclosing balls. *Algorithms-ESA'99*, pages 693–693, 1999.

[82] Bernd Gärtner and Sven Schönherr. An efficient, exact, and generic quadratic programming solver for geometric optimization. In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 110–118. ACM, 2000.

[83] Mihai Bādoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 250–257. ACM, 2002.

[84] Mihai Bădoiu and Kenneth L Clarkson. Optimal core-sets for balls. *Computational Geometry*, 40(1):14–22, 2008.

[85] Mihai Badoiu and Kenneth L Clarkson. Smaller core-sets for balls. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 801–802. Society for Industrial and Applied Mathematics, 2003.

[86] Piyush Kumar, Joseph SB Mitchell, and E Alper Yildirim. Approximate minimum enclosing balls in high dimensions using core-sets. *Journal of Experimental Algorithmics (JEA)*, 8:1–1, 2003.

[87] David MJ Tax and Robert PW Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004.

[88] Tingting Mu and Asoke K Nandi. Multiclass classification based on extended support vector data description. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(5):1206–1216, 2009.

[89] S Asharaf, M Narasimha Murty, and Shirish Krishnaj Shevade. Multiclass core vector machine. In *Proceedings of the 24th international conference on Machine learning*, pages 41–48. ACM, 2007.

[90] Ivor W Tsang, Andras Kocsor, and James T Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th international conference on Machine learning*, pages 911–918. ACM, 2007.

[91] IW-H Tsang, JT-Y Kwok, and Jacek M Zurada. Generalized core vector machines. *IEEE Transactions on Neural Networks*, 17(5):1126–1140, 2006.

[92] S Asharaf, M Narasimha Murty, and Shirish Krishnaj Shevade. Cluster based core vector machine. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 1038–1042. IEEE, 2006.

[93] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, Proceedings. IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE, 2004.

[94] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

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

[96] Bo Lu, Guoren Wang, Ye Yuan, and Dong Han. Semantic concept detection for video based on extreme learning machine. *Neurocomputing*, 102:176–183, 2013.

[97] Abdul Adeel Mohammed, Rashid Minhas, QM Jonathan Wu, and Maher A Sid-Ahmed. Human face recognition based on multidimensional PCA and extreme learning machine. *Pattern Recognition*, 44(10):2588–2597, 2011.

[98] Weiwei Zong and Guang-Bin Huang. Face recognition based on extreme learning machine. *Neurocomputing*, 74(16):2541–2551, 2011.

[99] Xiang-guo Zhao, Guoren Wang, Xin Bi, Peizhen Gong, and Yuhai Zhao. Xml document classification based on elm. *Neurocomputing*, 74(16):2444–2451, 2011.

[100] AH Nizar, ZY Dong, and Y Wang. Power utility nontechnical loss analysis with extreme learning machine method. *IEEE Transactions on Power Systems*, 23(3):946–955, 2008.

[101] Guang-Bin Huang, Qin-Yu Zhu, KZ Mao, Chee-Kheong Siew, Paramasivan Saratchandran, and Narasimhan Sundararajan. Can threshold networks be trained directly? *IEEE Transactions on Circuits and Systems Part 2: Express Briefs*, 53(3):187–191, 2006.

[102] Bin Li, Xuewen Rong, and Yibin Li. An improved kernel based extreme learning machine for robot execution failures. *The Scientific World Journal*, 2014, 2014.

[103] Guorui Feng, Guang-Bin Huang, Qingping Lin, and Robert Gay. Error minimized extreme learning machine with growth of hidden nodes and incremental learning. *IEEE Transactions on Neural Networks*, 20(8):1352–1357, 2009.

[104] Yuan Lan, Yeng Chai Soh, and Guang-Bin Huang. Ensemble of online sequential extreme learning machine. *Neurocomputing*, 72(13):3391–3395, 2009.

[105] Hai-Jun Rong, Guang-Bin Huang, Narasimhan Sundararajan, and Paramasivan Saratchandran. Online sequential fuzzy extreme learning machine for function approximation and classification problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(4):1067–1072, 2009.

[106] Nikhitha K Nair and S Asharaf. Tensor decomposition based approach for training extreme learning machines. *Big Data Research*, 10:8–20, 2017.

[107] Xinwang Liu, Lei Wang, Guang-Bin Huang, Jian Zhang, and Jianping Yin. Multiple kernel extreme learning machine. *Neurocomputing*, 149:253–264, 2015.

[108] Xin Bi, Xiangguo Zhao, Guoren Wang, Pan Zhang, and Chao Wang. Distributed extreme learning machine with kernels based on mapreduce. *Neurocomputing*, 149:456–463, 2015.

[109] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[110] Quan Wang. Kernel principal component analysis and its applications in face recognition and active shape models. *arXiv preprint arXiv:1207.3538*, 2012.

[111] Jong-Min Lee, ChangKyoo Yoo, Sang Wook Choi, Peter A Vanrolleghem, and In-Beum Lee. Nonlinear process monitoring using kernel principal component analysis. *Chemical engineering science*, 59(1):223–234, 2004.

[112] Heiko Hoffmann. Kernel PCA for novelty detection. *Pattern recognition*, 40(3):863–874, 2007.

[113] Devy Widjaja, Carolina Varon, Alexander Dorado, Johan AK Suykens, and Sabine Van Huffel. Application of kernel principal component analysis for single-lead-ecg-derived respiration. *IEEE Transactions on Biomedical Engineering*, 59(4):1169–1176, 2012.

[114] Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

[115] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Two-stage learning kernel algorithms. In *ICML*, pages 239–246. Citeseer, 2010.

[116] Gert RG Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine learning research*, 5(Jan):27–72, 2004.

[117] Sören Sonnenburg, Gunnar Rätsch, and Christin Schäfer. A general and efficient multiple kernel learning algorithm. In *Advances in neural information processing systems*, pages 1273–1280, 2006.

[118] Alexander Zien and Cheng Soon Ong. Multiclass multiple kernel learning. In *Proceedings of the 24th international conference on Machine learning*, pages 1191–1198. ACM, 2007.

[119] Mehmet Gönen and Ethem Alpaydin. Localized multiple kernel learning. In *Proceedings of the 25th international conference on Machine learning*, pages 352–359. ACM, 2008.

[120] Cijo Jose, Prasoon Goyal, Parv Aggrwal, and Manik Varma. Local deep kernel learning for efficient non-linear SVM prediction. In *International Conference on Machine Learning*, pages 486–494, 2013.

[121] Jinfeng Zhuang, Jialei Wang, Steven CH Hoi, and Xiangyang Lan. Unsupervised multiple kernel learning. *Journal of Machine Learning Research-Proceedings Track*, 20:129–144, 2011.

[122] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Representing shape with a spatial pyramid kernel. In *Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 401–408. ACM, 2007.

[123] Theodoros Damoulas and Mark A Girolami. Probabilistic multi-class multi-kernel learning: on protein fold recognition and remote homology detection. *Bioinformatics*, 24(10):1264–1270, 2008.

[124] Alexander Binder and Motoaki Kawanabe. Enhancing recognition of visual concepts with primitive color histograms via non-sparse multiple kernel learning. In *Workshop of the Cross-Language Evaluation Forum for European Languages*, pages 269–276. Springer, 2009.

[125] Peter Gehler and Sebastian Nowozin. Infinite kernel learning. Technical Report TR-178, Max Planck Institute For Biological Cybernetics, 2008.

[126] Eric V Strobl and Shyam Visweswaran. Deep multiple kernel learning. In *Machine Learning and Applications (ICMLA), 12th International Conference on*, volume 1, pages 414–417. IEEE, 2013.

[127] Youngmin Cho and Lawrence K Saul. Analysis and extension of arc-cosine kernels for large margin classification. *arXiv preprint arXiv:1112.3712*, 2011.

[128] Mingyuan Jiu and Hichem Sahbi. Laplacian deep kernel learning for image annotation. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 1551–1555. IEEE, 2016.

[129] Tamir Hazan and Tommi Jaakkola. Steps toward deep kernel methods from infinite neural networks. *arXiv preprint arXiv:1508.05133*, 2015.

[130] Rabha O Abd-Elsalam, Yasser F Hassan, and Mohamed W Saleh. New deep kernel learning based models for image classification. *International Journal of Advanced Computer Science and Applications*, 8(7):407–411, 2017.

[131] Jinfeng Zhuang, Ivor W Tsang, and Steven CH Hoi. Two-layer multiple kernel learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 909–917, 2011.

[132] Ilyes Rebai, Yassine BenAyed, and Walid Mahdi. Deep multilayer multiple kernel learning. *Neural Computing and Applications*, 27(8):2305–2314, 2016.

[133] Guanglu Zhou, Kim-Chuan Tohemail, and Jie Sun. Efficient algorithms for the smallest enclosing ball problem. *Computational Optimization and Applications*, 30(2):147–160, 2005.

[134] E Alper Yildirim. Two algorithms for the minimum enclosing ball problem. *SIAM Journal on Optimization*, 19(3):1368–1391, 2008.

[135] Kaspar Fischer, Bernd Gärtner, and Martin Kutz. Fast smallest-enclosing-ball computation in high dimensions. In *ESA*, volume 2832, pages 630–641. Springer, 2003.

[136] Kaspar Fischer and Bernd Gärtner. The smallest enclosing ball of balls: combinatorial structure and algorithms. *International Journal of Computational Geometry & Applications*, 14(04n05):341–378, 2004.

[137] Chih-Chung Chang and Chih-Jen Lin. LibSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[138] CL Blake and Christopher J Merz. UCI repository of machine learning databases [http://www. ics. uci. edu/˜ mlearn/mlrepository. html]. Irvine, CA: University of California. *Department of Information and computer science*, 55, 1998.

[139] Ivor Tsang, Andras Kocsor, and James Kwok. LibCVM toolkit. https://github.com/aydindemircioglu/libCVM, 2011.

[140] Chih-Chung Chang and Chih-Jen Lin. LIBSVM-2.91: A library for support vector machines. Software available at https://www.csie.ntu.edu.tw/~cjlin/libsvm/oldfiles/, 2011.

[141] François Chollet. Keras. https://github.com/fchollet/keras, 2015.

[142] A L Afzal and S Asharaf. Deep kernel learning in core vector machines. *Pattern Analysis and Applications*, pages 1–9, 2017.

[143] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[144] Guang-Bin Huang, Qin-Yu Zhu, and Chee Kheong Siew. Real-time learning capability of neural networks. *IEEE Transactions on Neural Networks*, 17(4):863–878, 2006.

[145] Guang-Bin Huang, Lei Chen, and Chee Kheong Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006.

[146] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529, 2012.

[147] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321. ACM, 2015.

[148] Guang-Bin Huang. An insight into extreme learning machines: random neurons, random features and kernels. *Cognitive Computation*, 6(3):376–390, 2014.

List of Publications

International Journals

- **Afzal, A.L.** and Asharaf, S., "Deep kernel learning in core vector machines". Pattern Analysis and Applications, Springer. [Impact Factor : 1.352; Indexed in : Science Citation Index Expanded (SciSearch), Journal Citation Reports/Science Edition, SCOPUS, etc..]

- **Afzal A.L.** and Asharaf S. "Deep multiple multilayer kernel learning in core vector machines", Expert Systems with Applications, Elsevier. [Impact Factor : 3.928; Indexed in : Science Citation Index Expanded, Scopus, etc..]

Manuscript Submitted

- **Afzal, A.L.**, Nikhitha k. Nair and Asharaf, S., "Kernel based extreme learning machines". Pattern Analysis and Applications, Springer. [Impact Factor : 1.352; Indexed in : Science Citation Index Expanded (SciSearch), Journal Citation Reports/Science Edition, SCOPUS, etc..]

Pattern Anal Applic (2018) 21:721–729
https://doi.org/10.1007/s10044-017-0600-4

THEORETICAL ADVANCES

Deep kernel learning in core vector machines

A. L. Afzal[1] · **S. Asharaf**[2]

Received: 4 August 2016 / Accepted: 18 January 2017 / Published online: 11 February 2017

Abstract In machine learning literature, deep learning methods have been moving toward greater heights by giving due importance in both data representation and classification methods. The recently developed multilayered arc-cosine kernel leverages the possibilities of extending deep learning features into the kernel machines. Even though this kernel has been widely used in conjunction with support vector machines (SVM) on small-size datasets, it does not seem to be a feasible solution for the modern real-world applications that involve very large size datasets. There are lot of avenues where the scalability aspects of deep kernel machines in handling large dataset need to be evaluated. In machine learning literature, core vector machine (CVM) is being used as a scaling up mechanism for traditional SVMs. In CVM, the quadratic programming problem involved in SVM is reformulated as an equivalent minimum enclosing ball problem and then solved by using a subset of training sample (Core Set) obtained by a faster $(1+\epsilon)$ approximation algorithm. This paper explores the possibilities of using principles of core vector machines as a scaling up mechanism for deep support vector machine with arc-cosine kernel. Experiments on different datasets show that the proposed system gives high classification accuracy with reasonable training time compared to traditional core vector machines, deep support vector machines with arc-cosine kernel and deep convolutional neural network.

Keywords Deep kernel · Core vector machines · Scalability · Support vector machines · Arc-cosine kernel

1 Introduction

The attempts to build algorithms to solve cognitive tasks such as visual object or pattern recognition, speech perception, and language understanding, etc. have attracted the attention of many machine learning researchers in the recent past. The theoretical and biological arguments in this context strongly suggest that building such systems requires deep architectures that involve many layers of nonlinear information processing. This deep structured learning paradigm is commonly called deep learning or hierarchical learning. In machine learning research, some of the motivating factors favoring deep learning architectures are: the wide range of functions that can be parameterized by composing weakly nonlinear transformations, the appeal of hierarchical distributed representations and the potential for combining unsupervised and supervised methods [5, 13, 14, 17, 20]. Deep learning approach has originally emerged and been widely used in the area of neural networks. Techniques developed from deep learning researches have already been impacting a wide range of signal and information processing application areas [12, 18, 27]. Deep architectures learn complex mappings by transforming their inputs through multiple layers of nonlinear processing. They involve highly nonlinear optimizations and many heuristics for gradient-based learning,

✉ A. L. Afzal
afzal.res15@iiitmk.ac.in

S. Asharaf
asharaf.s@iiitmk.ac.in

[1] Data Engineering Lab, Indian Institute of Information Technology and Management-Kerala (IIITM-K), Thiruvananthapuram, India

[2] Indian Institute of Information Technology and Management-Kerala (IIITM-K), Thiruvananthapuram, India

which do not guarantee global optimal solutions due to the non-convexity of the optimization problem.

In the recent past, machine learning researchers have tried to adapt deep learning techniques for other machine learning paradigms like kernel machines. The machine learning approaches leveraging the concepts of kernel trick are referred as the kernel machine. The kernel machines have become rather popular in machine learning literature because of its very strong mathematical slant and guaranteed global optima. Any machine learning algorithm modeled by the inner product between data samples can be transformed into kernel machines by choosing an appropriate kernel that compute the inner product of data samples implicitly in the feature space [19]. The recently developed multi layered kernel called arc-cosine kernel [8] leverages the possibilities of extending deep learning features into the kernel machines. The arc-cosine kernel has multiple layers of nonlinear transformation that mimics the computations in multilayer neural network. The nonlinear transformation in each layer is determined by the given value of its activation or degree. Arc-cosine kernels of different degrees have qualitatively different geometric properties. There are reported works which demonstrate the possibility of deep kernel machine by combining the well-known kernel machine, support vector machine, with deep arc-cosine kernel and it has been widely used in applications that involve small-size datasets [8]. However, these deep support vector machine (DSVM)[1] models show high computational complexity and necessitate much of exploration on its scalability aspects.

Traditional Binary [11, 26] and multi-class [15] SVMs have been usually formulated as Quadratic Programing (QP) problem, and this formulation has a time complexity[2] of $O(n^3)$ and a space complexity of $O(n^2)$, for 'n' training samples. If an l-layered arc-cosine kernel is used for modeling a deep kernel machine, then its time complexity will be the order of $O(l * n^3)$. It may take months or years for learning modern real-world applications that involve large amount of data samples. It is a long-term research to scale up kernel machines particularly SVMs so as to conciliate modern real-world applications that involve very large size datasets.

Many attempts have been made to solve this scalability problem in kernel machines. The recent development in this category, the core vector machine (CVM), resolves the scalability problem by exploiting the "approximateness" property of SVMs and reformulating the quadratic problem (QP) as an equivalent minimum enclosing ball problem (MEB) and then uses an $(1+\epsilon)$ MEB approximation algorithm to obtain the core set (a subset of training samples) in the kernel-induced feature space [23]. This would attain a reduced time complexity as linear and a space complexity which is independent of both dimension and size of input data sample. Even though the CVM achieves a faster execution on large dataset it can only be used with certain kernel functions and methods. This limitation has been solved in generalized core vector machines [22] by extending the CVM implementation with the center-constrained MEB problem. It still preserve the time complexity as linear and space complexity that is independent of both dimension and size of input dataset. Ball vector machines (BVM) is an another implementation of CVM in which the authors use an easier enclosing ball (EB) problem where the ball's radius is fixed [25]. They also proposed three $(1+\epsilon)$ approximation algorithm to obtain the core set and its implementation do not use any numerical solver. Another major contribution toward the performance of CVM is cluster based core vector machine [1]. It improves the performance of CVM by using Gaussian Kernel function irrespective of the orderings of pattern belonging to different classes with in the dataset. According to the authors, their method employs a selective sampling based training of CVM using a novel-based scalable hierarchical clustering algorithm. A powerful approach to scale up SVM for multi-class classification is multi-class core vector machine (MCVM) [2] that uses CVM techniques to solve the quadratic multi-class problem defining an SVM with vector-valued output. Even though all these proposals have been performing well for traditional SVM, the scalability aspects of deep support vector machine need much of exploration. This paper explores the possibility of using CVM as a scaling up mechanism for deep kernel machines, particularly deep support vector machines.

The rest of the paper is organized as follows. The basic idea of core vector machine and arc-cosine kernel are given in Sects. 2 and 3, respectively. Section 4 deals with proposed scalable deep kernel machine. Experimental results and main observations are included in Sect. 5. Section 6 gives conclusions.

2 Core vector machines

The state-of-the-art SVM implementations involves quadratic programming (QP) problem and typically have a time complexity of $O(n^3)$. To reduce the time and space complexities, core vector machine [23] has been proposed as a scaling up mechanism for SVM. Such architecture tend to obtain approximately optimal solution for SVM by reformulating the QP problem as an equivalent minimum enclosing ball (MEB) and then uses an efficient minimum

[1] DSVM: support vector machines with arc-cosine kernel.

[2] Solving the quadratic problem and choosing the support vectors involves the order of n^2, while solving the quadratic problem directly involves inverting the kernel matrix, which has complexity on the order of n^3.

enclosing ball approximation algorithm with the idea of core set, a subset of training samples. It can be further explained as in the following paragraphs.

Let $A = \{a_1, a_2, \ldots, a_n\}$ where $a_i \in \Re^d$ be the data points then the MEB(A), expressed as $B_A(c, r)$, computes the minimum radius of a ball that covers all the data points in A with center c and radius r. The primal and its equivalent dual form of MEB(A) can be expressed in the following two equations [24]:

$$(c, r) = \min_{c,r} \; r^2 : \|c - a_i\|^2 \le r^2; \qquad \forall i \tag{1}$$

$$\begin{aligned} &\max_{\alpha} \sum_{i=1}^{n} \alpha_i \langle a_i a_i \rangle - \sum_{i,j=1}^{n} \alpha_i \alpha_j \langle a_i a_j \rangle \\ &s.t. \quad \sum_{i=1}^{n} \alpha_i = 1, \quad \alpha_i \ge 0; \quad \forall i \end{aligned} \tag{2}$$

Any QP of this form can be considered as MEB problem[3]. The equivalent MEB problem in kernel-induced feature space for the kernel function 'k' satisfies the condition $k(a, a) = z$, a constant, can be expressed as [25]

Primal form:

$$(c, r) = \min_{c,r} \; r^2 : \|c - \phi(a_i)\|^2 \le r^2; \quad \forall i \tag{3}$$

Dual form:

$$\begin{aligned} &\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i k_{ii} - \sum_{i,j=1}^{n} \alpha_i \alpha_j k_{ij} \\ &s.t. \quad \alpha_i \ge 0; \quad \text{for i} = 1 \ldots \text{m}, \quad \sum_{\text{i}=1}^{\text{n}} \alpha_{\text{i}} = 1 \end{aligned} \tag{4}$$

where $k_{ij} = k(a_i, a_j) = \langle \phi(a_i), \phi(a_j) \rangle$. It is observed that if a kernel is either isotropic[4] (e.g., Gaussian kernel) or a dot product - $k(x, y) = k_l(x^{'} y)$ (e.g., polynomial kernel with normalized inputs) or any normalized kernel then it satisfies the condition $k_{ii} = z$ and hence the kernel machines involving these types of kernel can be easily converted to MEB problem. For example, the two class SVM can be reformulated as an MEB problem and expressed in Wolfe dual form [2] as

$$\begin{aligned} &\min_{\alpha} \sum_{i,j=1}^{n} \alpha_i \alpha_j \left(y_i y_j k(x_i x_j) + y_i y_j + \frac{\delta_{ij}}{C} \right) \\ &s.t. \sum_{i=1}^{m} \alpha_i = 1, \quad \alpha_i \ge 0 \quad \forall i \end{aligned} \tag{5}$$

where $x_i \in \Re^{\mathfrak{d}}$, $y_i \in \{+1, -1\}$, α is Lagrangian multiplier and δ_{ij} is the Kronecker delta function.[5] By using the transformed kernel $\hat{k}_{ij} = y_i y_j (k_{ij} + 1) + \frac{\delta_{ij}}{C}$, the above equation can be reformulated as

$$\min_{\alpha} \sum_{i,j=1}^{n} \alpha_i \alpha_j \hat{k}_{ij} \qquad s.t. \sum_{i=1}^{n} \alpha_i = 1 \quad \alpha_i \ge 0 \quad \forall i \tag{6}$$

when $k(x, x) = c$, a constant, is satisfied, the transformed kernel $\hat{k}$ also satisfies the condition $\hat{k}(x, x) = c_1$, some constant. Hence, the two class SVM problem, as shown in Eq. 5, can be further scaled by the idea of core sets obtained by a faster $(1 + \epsilon)$ approximation of MEB problem [3, 16]. Let $B_x(c, r)$ be the MEB(X) for $X \subset A$, then an expansion by a factor of $(1 + \epsilon)$ on MEB(X), $B_x(c, r(1 + \epsilon))$ covers all the points in A, and then X is said to be the core set of A. More formally, for a given $\epsilon \ge 0$; $B_x(c, (1 + \epsilon)r)$ is an $(1 + \epsilon)$ approximation of MEB(A) if $A \subseteq B_x(c, (1 + \epsilon)r)$. An iterative strategy proposed by [4] has been used to achieve this $(1 + \epsilon)$ approximation. In each iteration, the radius r_i of the current evaluation $B_x(c_i, r_i)$ is incremented by a factor of $(1 + \epsilon)$ and a point in A, farthest and outside of the ball $B_x(c_i, (1 + \epsilon)r_i)$, is added to the core set. This process is repeated until there is no such point in the given dataset A.

3 Arc-cosine kernel: a deep kernel

Deep learning processes have been originated and pursuing in the area of artificial neural network. Meanwhile, some of the researchers turn their attention toward the exploration of the possibilities of deep learning in kernel machines particularly on well-known maximal margin classifier, support vector machines (SVM). The first approach in this area was the optimization of multiple, multilayered arc-cosine kernel [9] and the authors defined the nth-order kernel in the family of arc-cosine kernel as

$$k_n(x, y) = 2 \int \mathrm{d}w \frac{e^{-\frac{\|W\|^2}{2}}}{(2\Pi)^{\frac{d}{2}}} \Theta(w.x) \Theta(w.y) (w.x)^n (w.y)^n \tag{7}$$

where Θ denotes the Heaviside function; $\Theta(z) = \frac{1}{2}(1 + \text{sign(z)})$. The above equation has a dependency on both magnitudes of the inputs and angle between them. In simple form, it can be written as:

$$k_n(x, y) = \frac{1}{\pi} \|x\|^n \|y\|^n j_n(\theta) \tag{8}$$

where the angular dependency

[3] $\langle a, b \rangle$ represent dot product between vectors a,b.

[4] When a kernel depends only on the norm of the lag vector between two examples, and not on the direction, then the kernel is said to be isotropic: $k(x, z) = k_l(\|x - y\|)$.

[5] $\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \ne j \end{cases}$

$$j_n(\theta) = (-1)^n(\sin\theta^{2n+1})\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\right)^n\left(\frac{\pi-\theta}{\sin\theta}\right) \quad (9)$$

The authors also provided first three expressions of $j_n(\theta)$ as

$$j_0(\theta) = \pi - \theta$$
$$j_1(\theta) = \sin\theta + (\pi-\theta)\cos\theta$$
$$j_2(\theta) = 3\sin\theta\cos\theta + (\pi-\theta)(1+2\cos^2\theta).$$

For the case $n = 0$, the kernel function takes the simple form:

$$k_0(x,y) = 1 - \frac{\theta}{\pi} \quad (10)$$

The angle θ between the input vectors x and y is

$$\theta = \cos^{-1}\left(\frac{(x.y)}{||x||||y||}\right) \quad (11)$$

The multilayered, deep arc-cosine kernel can be formed by combining single layered arc-cosine kernel (Eq. 8) in an iterative manner, with same or different activation function in each layer. The layered kernel learning process start with initial kernels: $k^0(x_i,x_i) = <x_i,x_i>$, $k^0(x_j,x_j) = <x_j,x_j>$ and $k^0(x_i,x_j) = <x_i,x_j>$. The working procedure of multilayered arc-cosine kernel [8] has been shown in Algorithm 1. In this algorithm, the subfunction Compute_J (Algorithm 2) is used to compute the angular dependency $J_n\theta$.

This kernel has been successfully used in conjunction with SVM and has demonstrated promising results in many real-world applications. However, their generalization performance is often controlled by the size of datasets.

4 Deep core vector machine

In multilayered arc-cosine kernel, each layer may have different activation function (degree). The nonlinear transformation in each layer is determined by the given value of its degree. The arc-cosine kernel with different activation functions has qualitatively different geometric properties [9]. Let n be the degree of the arc-cosine kernel at a particular level. Then, for the case $n = 0$ arc-cosine kernel maps input x to the unit hypersphere in feature space, with $k_0(x,x) = 1$ and it behaves like radial basis function (RBF). For the case $n = 1$, arc-cosine kernel preserves the norm of input with $k_1(x,x) = ||x||^2$ which is similar to linear kernel and in the case of $n \geq 1$, it expands the dynamic range of the input, with $k_n(x,x) = ||x||^{2n}$ and it is similar to polynomial kernel. From the above analysis of arc-cosine kernel, we can see that some special cases, particularly with degree 0, satisfies the required condition ($k(x_i,x_i) = c$, a constant) for being a kernel function in CVM. Based on this observation, we are modeling a new

Algorithm 1: Deep arc-cosine kernel : $K_{ij} = Arccosine(X_i, X_j, [act\_list])$

Input: Two data samples X_i and X_j ,the list of activation or degree in each layer $[act\_list]$
Output: K_{ij}, the kernel value for the samples X_i, X_j

Set k_{ii} = **dot**(X_i, X_i);
Set k_{jj} = **dot**(X_j, X_j);
Set k_{ij} = **dot**(X_i, X_j);
For each n in [act-list]
  $\theta = \cos^{-1}\left(\frac{k_{ij}}{\|k_{ii}\|\|k_{jj}\|}\right)$;
  Find $J_n\theta$ = **Compute_J**$(\mathbf{n}, \theta)$;
  $K_{ij} = \frac{1}{\pi} \times (k_{ij})^{\frac{n}{2}} \times J_n\theta$;
  $K_{ii} = \frac{1}{\pi} \times (k_{ii})^{n} \times J_n\theta$;
  $K_{jj} = \frac{1}{\pi} \times (k_{jj})^{n} \times J_n\theta$;
Return k_{ij}

Algorithm 2: Computation of $J_n\theta$: $Jn\theta = Compute\_J(n,\theta)$

If n == o then
  $J_n\theta = \pi - \theta$;
Else If n == 1 then
  $J_n\theta = sin(\theta) + (\pi - \theta) \times cos(\theta)$;
Else If n == 2 then
  $J_n\theta = 3 \times sin(\theta) \times cos(\theta) + (\pi - \theta) \times (1 + 2 \times cos(\theta)^2)$;
Return $J_n\theta$

scalable deep core vector machine by combining the deep arc-cosine kernel with the core vector machine.

Equation 6 shows an equivalent MEB formulation of two class support vector machine, where the transformed kernel $\hat{k}_{ij} = y_i y_j (k_{ij} + 1) + \frac{\delta_{ij}}{C}$. In this transformed kernel, k_{ij} represents the actual kernel function and it should satisfy the condition $k_{ii} = c$, a constant. Similarly, we can write the equivalent MEB formulation of deep kernel machine with multiple, multilayered (deep) kernel as

$$\min_{\alpha} \sum_{i,j=1}^{n} \alpha_i \alpha_j \hat{k}_n^l(x_i, x_j) \quad s.t. \sum_{i=1}^{n} \alpha_i = 1, \quad \alpha_i \geq 0 \quad \forall i \tag{12}$$

where the transformed kernel

$$\hat{k}_n^l(x_i, x_j) = y_i y_j (k_n^l(x_i, x_j) + 1) + \frac{\delta_{ij}}{C} \tag{13}$$

Here $K_n^l(x_i, x_j)$ represents l-fold arc-cosine kernel with degree n. It is recursive in nature and can be expressed as

$$k_n^l(x_i, x_j) = \frac{1}{\pi} [k_n^{(l-1)}(x_i, x_i) k_n^{(l-1)}(x_j, x_j)]^{\frac{n}{2}} j_n \theta_n^{(l-1)} \tag{14}$$

where $\theta_n^{(l)}$ is the angle between the images of x and y in the feature space induced by the l-fold composition.

$$\theta_n^{(l)} = \cos^{-1}\left(\frac{k_n^{(l)}(x_i, x_j)}{\sqrt{k_n^{(l)}(x_i, x_i) k_n^{(l)}(x_j, x_j)}}\right) \tag{15}$$

$j_n\theta$ is the angular dependency and its various forms are discussed in Sect. 3. The details of learning process in deep core vector machine using multilayered arc-cosine kernel (DCVM) have been given in the following section.

4.1 DCVM algorithm

The learning process in deep core vector machine uses the concept of core set obtained by $(1+\epsilon)$ approximation of MEB, as in CVM. The iterative $(1+\epsilon)$ approximation algorithm for DCVM is shown in Algorithm 3. The symbols used in this algorithm are:

- T: the data set.
- S_i: the current core set.
- c_i, r_i: the ball center and radius at ith iteration.
- k: the kernel matrix $\sum_{i,j=1}^{n} k_n^l(x_i, x_j)$
- $\hat{k}_n^l, \hat{\phi}_n^l$: transformed deep kernel and its associated map. n is the activation function at layer l.

This algorithm employs the following user defined parameters

- The parameter L represents the list of degree (Activation function) of each layer. size (L) gives the number of layers.
- ϵ for MEB approximation algorithm.

Algorithm 3: DCVM : Deep Core Vector Machine

1 **Set** $i = 0$ **and initialize the core set** S_i
2 **Train MEB**(S_i) **using the Equation 12**
3 **Set the values for** c_i, R_i

- $c_i = C_{MEB}(S_i) = \sum_{j=1}^{m} \alpha_i \phi_n^l(x_i)$
- $R_i = r_{MEB}(S_i) = \sqrt{\alpha' diag(k) - \alpha' k \alpha}$

4 **Expand MEB**(S_i) **as** $(c_i, R_i(1+\epsilon))$
5 **Find a point A** $\in T$ **such that** $\hat{\phi}_n^l(A)$ **falling outside the ball** $(c_i, R_i(1+\epsilon))$ **and furthest away from** c_i
6 **If there is no such point A exist then**

EXIT

Else

$S_{i+1} = S_i \cup A$
$i = i + 1$

7 **Go to step 2.**

Algorithm starts with the initialization of core set S with one point from each of the classes [2]. In each iteration, the data sample $A \in T$ in the feature space $\hat{\Phi}_n^l(A)$ which is outside of the $(1+\epsilon)$ ball $(c_i, R_i(1+\epsilon))$ and furthest from c_i is taken as core vector. It involves computing the distance between the current center c_i and $\hat{\Phi}_n^l(A_j)$, $\forall A_j \in T$. It seems to be very expensive when the size of dataset is large. Here we are using a method mentioned in [23] to speedup the computational effort to find the furthest data point. This distance computation uses the multilayered arc-cosine kernel (k_n^l) evaluation instead of explicit $\hat{\Phi}_n^l(A)$.

Table 1 The seven publicly available datasets taken from libSVM and UCI repositories

S. No.	Dataset	#Clss	#Dim	#Train	#Test
1	Optdigit	9	64	3823	1797
2	Satimage	6	36	4435	2000
3	Pendigit	10	16	7494	3498
4	Letter	26	16	15,000	5000
5	Ijcnn1	2	22	49,990	91,701
6	News 20	20	62,061	15,935	3993
7	KDD-CUP-2010	2	29,890,095	19,264,097	748,401

$$\left\|c_i - \hat{\Phi}_n^l(A_j)\right\|^2 = \sum_{A_m, A_n \in S_i} \alpha_m \alpha_n \hat{k}_n^l(A_m, A_n) - 2\sum_{A_i \in S_i} \alpha_i \hat{k}_n^l(A_i, A_j) + \hat{k}_n^l(A_j, A_j) \tag{16}$$

The transformed kernel in this equation can be computed by Eq. 13 and its subsequences. The deep layered arc-cosine kernel uses an incremental approach for its training process, and it can be computed by using Algorithm 1. For example, the arc-cosine kernel of depth 1 with degree n can be computed as:

$$k_n^1(x,y) = \frac{1}{\pi}[k_n^{(0)}(x,x)k_n^{(0)}(y,y)]^{\frac{n}{2}} j_n(\theta_n^{(0)}) \tag{17}$$

where the base kernels $k^0(x,x) = <x,x>$ and, $k^0(y,y) = <y,y>$.

The training process in DCVM, except kernel computation, is same as that of core vector machines. In the proposed DCVM, the kernel parameter is a structure L, representing the list of activation values in each layer. The size of L determines the depth of the arc-cosine kernel. In machine leaning literature, any concrete method has not been reported for the selection of number of layers and list of activation values for the multilayered arc-cosine kernel and still this exist as an open problem. It is a common practice to train the arc-cosine model with all the combination of selected range of activation values. The MEB approximation algorithm for deep core vector machine terminates when there is no data sample outside of the $(1+\epsilon)$ ball. The high accuracy and low computational complexity of DCVM show its high potential in the context of deep kernel machines.

5 Experimental results

Experiments have been conducted on seven data sets taken from both libSVM site [7] and UCI machine learning repository [6][6]. Table 1 shows the dataset composition. The training data size is ranged from 3823 to 19,264,097, and the range of dimension is from 16 to 29,890,095. All the methods except deep convolutional neural network (Deep CNN) use sparse representation of data.

The proposed deep core vector machine (DCVM) has been implemented in C++ by combining the features from both the packages libCVM-2.2 [21] and libSVM-2.91[7]. The experimental results of CVM/BVM were obtained by using the package libCVM-2.2, whereas the libSVM-2.91 was used for SVM and deep SVM (DSVM). In both the cases, the package-specified default values were used for all its parameters. Deep CNN has been implemented in python 2.7.6 using the library Keras with Theano backend. All experiments were done on a server machine configured with AMD Opteron 6376 @ 2.3 GHz/16 core processor and 16 GB RAM. The radial basis function (rbf kernel) with $\gamma = 1/number\_of\_features$ has been used in SVM. In the case of DSVM and DCVM, experiments were repeated for arc-cosine kernel with all the combinations of activation values 0, 1, 2 and 3. By fixing softmax as the output layer activation function, various layer-activation-optimization combinations were experimented with Deep CNN. The list of common attributes and its values [10] used in Deep CNN are displayed in Table 2. The generalization performance in terms of prediction accuracy (in %) and CPU time (in s) have been recorded in all the experiments.

5.1 Performance evaluation

This section compares the accuracy of our proposed DCVM with DSVM and deep CNN. we also compare the variation in accuracies while transforming shallow-based SVM and CVM into deep SVM and deep CVM, respectively. The performance of deep CNN on various datasets are shown in Table 3. The column names Act, Optm and Acc represent activation function, optimization technique and prediction accuracy, respectively. Column #Layers record the number of convolutional layers and the column name. Filters represent the combination of number of

[6] Optdigit was taken from UCI machine learning repository and all other datasets were taken from libSVM.

[7] Available at: https://www.csie.ntu.edu.tw/cjlin/libsvm/oldfiles/.

Table 2 List of common attributes and its values used in deep convolution neural network model

Attributes	Values
Loss function	Categorical cross-entropy
Convolution layer-filter size	3×1
Max-pooling size	2×1
No. of epochs	10
Batch size	10
Activation functions tested	$\{relu, liear, sigmoid\}$
Optimizers tested	$\{sgd, adam, adagrad, adamax\}$

filters. The column CPUtime stores the training time in seconds. In the case of Kdd-Cup-2010 dataset, if the number of layers exceed 2, Deep CNN does not converges even after 5 days of its training.

Table 4 shows the classification accuracy of deep core vector machine (in column DCVM), the best among traditional CVM and BVM (in column CVM/BVM), deep support vector machine (in column DSVM), support vector machine (in column SVM) and deep CNN (in column DCNN) on various datasets. For the datasets 1–5, the prediction accuracy values in column CVM/BVM and SVM were taken from the paper [25]. For the remaining datasets, our experimental result on libCVM-2.2 and libSVM-2.91 packages were recorded. In our experiment, default values were assigned to all the parameters except degree of arc-cosine kernels. In the case of DCVM and DSVM, the corresponding activation values for the l-fold arc-cosine kernel are shown in parenthesis. For example, in the case of Satimage dataset, DCVM and DSVM performs well in 3-layer arc-cosine kernel with degrees 0, 1 and 2, respectively. The bold face values indicates the highest accuracy on each dataset. The NT values in this table shows that there is no result within 5 days of training. The following factors can be observed from Table 4.

- In terms of accuracy, DCVM performs better than the best of CVM/BVM models, traditional deep support vector machines and deep CNN.
- Introducing deep arc-cosine kernel into CVM is more appreciated than that of SVM.

For comparing the performance in terms of computation time, we have recorded the training time taken by DSVM and DCVM on various datasets and it is shown in Table 5. As in the case of accuracy, the NT values in this table also show that there is no result within 5 days. Minimum training time taken for each dataset is shown in bold values. With the increase in training size, it is observed that the training time for DCVM is comparatively lesser than that of DSVM.

Performance of the proposed DCVM in terms of training time against the size of dataset can be further illustrated by the graph shown in Fig. 1. In this experiment, we have created four data subset of sizes $10^4, 10^5, 10^6$ and 10^7 respectively, from the KDD-CUP-2010 raining dataset

Table 3 The performance of deep CNN in terms of accuracy and CPU times

Datasets	#Layers	Filters	Act	Optm	Acc	CPUtime
Optdigit	6	(64,64,128,128,256,256)	Linear	Sgd	96.88	303.21
Satimage	6	(64,64,128,128,256,256)	Relu	Adam	90.87	417.43
Pendigit	8	(64,64,128,128,128,256,256,256)	Relu	Adamax	98.04	627.82
Letter	8	(64,64,128,128,128,256,256,256)	Linear	Sgd	89.57	1022.72
Ijcnn1	8	(64,64,128,128,128,256,256,256)	Relu	Adamax	98.97	3312.7
News 20	7	(64,64,128,128,128,256,256)	Linear	Sgd	83.87	14630.68
Kdd-Cup-2010	2	(16,32)	Linear	Adam	43.25	112,347.40

Table 4 The generalization performance in terms of prediction accuracy of Deep CVM, CVM/BVM, deep SVM, SVM and deep CNN

Dataset	DCVM	CVM/BVM	DSVM	SVM	DCNN
Optdigit	**97.71 (0)**	96.38	97.02 (0,1)	96.77	97.01
Satimage	**92.15 (0,1,2)**	89.60	90.35 (0,1,2)	89.55	90.87
Pendigit	**98.37 (0,1)**	97.97	97.94 (0,1,2)	97.91	98.04
Letter	**96.94 (1,0)**	94.47	95.02 (0,1,2)	94.25	90.57
Ijcnn1	**99 (0,1,2)**	98.67	97.99 (0,1)	98.97	98.97
News 20	**84.07 (0,1,2)**	79.61	83.92 (0,1,2)	72.38	83.87
Kdd-Cup-2010	**88.75 (0)**	63.83	NT	61	43.25

For the datasets (1–5), the values in CVM/BVM and SVM were taken from [25]. The list of activation values for the deep arc-cosine kernel has been shown in parenthesis

Table 5 Training time of DCVM and DSVM on various datasets

Dataset	DCVM	DSVM	DCNN
Optdigit	**1.23**	2.02	203.21
Satimage	2.73	**2.14**	217.49
Pendigit	3.02	**2.90**	427.82
Letter	**28.01**	30.03	631.46
Ijcnn1	**316.97**	383.23	7312.71
News 20	**329.43**	403.09	14,630.68
Kdd-Cup-2010	**95879**	NT	112,347.40

The training times are recorded in seconds and it is against the accuracy shown in Table 4

mentioned in Table 1. The DCVM and DSVM were trained with all the combination of activation values 0, 1, 2 and 3 of arc-cosine kernel. The training time taken by DCVM and DSVM against highest accuracy was recorded for plotting the graph. A common data subset of size 25,000, created from the KDD-CUP-2010 testing dataset, was used as testing dataset. The major observations from this graph are:

- DCVM is much faster than DSVM on large datasets.
- As the number of training sample increases, the growth rate of training time of DCVM is highly appreciative when compared to that of DSVM.
- On KDD-CUP 2010, DSVM does not converge even after 5 days of execution when the sample size exceeds 10^6.
- On small datasets, DSVM converges more quickly, because of its highly sophisticated heuristics.

6 Conclusions

This paper models deep core vector machine as a scalable mechanism for deep support vector machines. While analyzing the behavior of arc-cosine kernel, it has been observed that in some cases of activation function arc-cosine kernel satisfies the required condition for being a kernel in core vector machines. Our proposed algorithm combines the widely used scaling up mechanism for SVM such as core vector machines and the deep arc-cosine kernel. Various experiments on different datasets have been conducted to demonstrate the generalization performances in terms of prediction accuracy and training time of our proposed deep core vector machine. Even in the case of large size datasets, our model converges with high accuracy in a lower training time. The empirical results also show the potential of DCVM in the context of scalable deep kernel machines.

Exploration of the possibilities of using core vector machine in the context of scalable deep multiple kernel learning model is a potential future dimension. The generalization of deep core vector machines to the multi-view version for handling the input data with multiple feature representations is another area for further exploration.

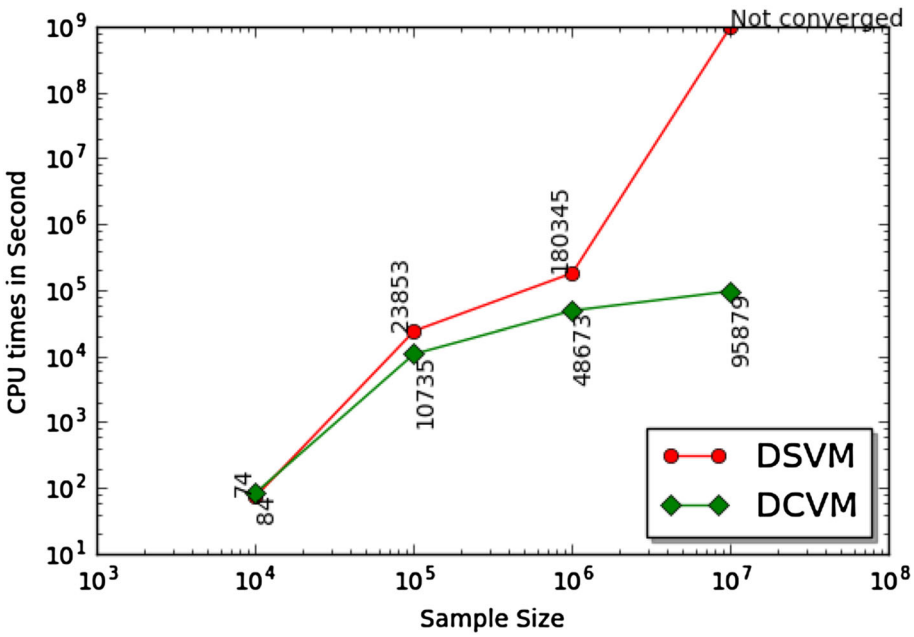

Fig. 1 Training time of DSVM and DCVM on different size of KDD-cup-2010 datasets

References

1. Asharaf S, Murty MN, Shevade SK (2006) Cluster based core vector machine. In: Sixth international conference on data mining (ICDM'06), IEEE, pp 1038–1042
2. Asharaf S, Murty MN, Shevade SK (2007) Multiclass core vector machine. In: Proceedings of the 24th international conference on machine learning, ACM, pp 41–48
3. Bădoiu M, Clarkson KL (2008) Optimal core-sets for balls. Comput Geom 40(1):14–22
4. Bădoiu M, Har-Peled S, Indyk P (2002) Approximate clustering via core-sets. In: Proceedings of the thirty-fourth annual ACM symposium on theory of computing, ACM, pp 250–257
5. Bengio Y, Lamblin P, Popovici D, Larochelle H et al (2007) Greedy layer-wise training of deep networks. Adv Neural Inf Process Syst 19:153
6. Blake C, Merz CJ (1998) UCI repository of machine learning databases [http://www.ics.uci.edu/∼mlearn/mlrepository.html]. University of California. Department of Information and Computer Science 55, Irvine
7. Chang CC, Lin CJ (2011) Libsvm: a library for support vector machines. ACM Trans Intell Syst Technol (TIST) 2(3):27
8. Cho Y, Saul LK (2009) Kernel methods for deep learning. In: Advances in neural information processing systems, pp 342–350
9. Cho Y, Saul LK (2011) Analysis and extension of arc-cosine kernels for large margin classification. arXiv preprint arXiv:11123712
10. Chollet F (2015) Keras. https://github.com/fchollet/keras
11. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297
12. Hinton G, Deng L, Yu D, Dahl GE, Ar Mohamed, Jaitly N, Senior A, Vanhoucke V, Nguyen P, Sainath TN et al (2012) Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Sign Process Mag 29(6):82–97
13. Hinton GE (2007) Learning multiple layers of representation. Trends Cogn Sci 11(10):428–434
14. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
15. Ji Y, Sun S (2013) Multitask multiclass support vector machines: model and experiments. Pattern Recogn 46(3):914–924
16. Kumar P, Mitchell JS, Yildirim EA (2003) Approximate minimum enclosing balls in high dimensions using core-sets. J Exp Algorithm (JEA) 8:1–1
17. Lee H, Grosse R, Ranganath R, Ng AY (2009) Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: Proceedings of the 26th annual international conference on machine learning, ACM, pp 609–616
18. Miao Y, Metze F, Rawat S (2013) Deep maxout networks for low-resource speech recognition. In: Automatic speech recognition and understanding (ASRU), 2013 IEEE Workshop on, IEEE, pp 398–403
19. Muller KR, Mika S, Ratsch G, Tsuda K, Scholkopf B (2001) An introduction to kernel-based learning algorithms. IEEE Trans Neural Netw 12(2):181–201
20. Salakhutdinov R, Hinton GE (2009) Deep boltzmann machines. In: AISTATS, vol 1, p 3
21. Tsang I, Kocsor A, Kwok J (2011) Libcvm toolkit. https://github.com/aydindemircioglu/libCVM
22. Tsang IH, Kwok JY, Zurada JM (2006) Generalized core vector machines. IEEE Trans Neural Netw 17(5):1126–1140
23. Tsang IW, Kwok JT, Cheung PM (2005) Core vector machines: Fast svm training on very large data sets. J Mach Learn Res 6:363–392
24. Tsang IW, Kwok JTY, Cheung PM (2005) Very large svm training using core vector machines. In: AISTATS
25. Tsang IW, Kocsor A, Kwok JT (2007) Simpler core vector machines with enclosing balls. In: Proceedings of the 24th international conference on machine learning, ACM, pp 911–918
26. Vapnik V, Golowich SE, Smola A, et al (1997) Support vector method for function approximation, regression estimation, and signal processing. Adv Neural Inf Process Syst 281–287
27. Zhong Sh, Liu Y, Liu Y (2011) Bilinear deep learning for image classification. In: Proceedings of the 19th ACM international conference on Multimedia, ACM, pp 343–352

Expert Systems With Applications 96 (2018) 149–156

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Deep multiple multilayer kernel learning in core vector machines

Afzal A.L.[a,*], Asharaf S.[b]

[a] Data Engineering Lab, Indian Institute of Information Technology and Management - Kerala (IIITM-K), Thiruvananthapuram, India
[b] Indian Institute of Information Technology and Management - Kerala (IIITM-K), Thiruvananthapuram, India

ARTICLE INFO

Article history:
Received 15 June 2017
Revised 22 November 2017
Accepted 30 November 2017
Available online 2 December 2017

Keywords:
Deep kernel machines
Unsupervised MKL
Core vector machines
Scalability
Arc-cosine kernel
Multilayer kernel learning

ABSTRACT

Over the last few years, we have been witnessing a dramatic progress of deep learning in many real world applications. Deep learning concepts have been originated in the area of neural network and show a quantum leap in effective feature learning techniques such as auto-encoders, convolutional neural networks, recurrent neural networks etc. In the case of kernel machines, there are several attempts to model learning machines that mimic deep neural networks. In this direction, Multilayer Kernel Machines (MKMs) was an attempt to build a kernel machine architecture with multiple layers of feature extraction. It composed of many layers of kernel PCA based feature extraction units with support vector machine having arc-cosine kernel as the final classifier. The other approaches like Multiple Kernel Learning (MKL) and deep core vector machines solve the fixed kernel computation problem and scalability aspects of MKMs respectively. In addition to this, there are lot of avenues where the use of unsupervised MKL with both single and multilayer kernels in the multilayer feature extraction framework have to be evaluated. In this context, this paper attempts to build a scalable deep kernel machines with multiple layers of feature extraction. Each kernel PCA based feature extraction layer in this framework is modeled by the combination of both single and multilayer kernels in an unsupervised manner. Core vector machine with arc-cosine kernel is used as the final layer classifier which ensure the scalability in this model. The major contribution of this paper is a novel effort to build a deep structured kernel machine architecture similar to deep neural network architecture for classification. It opens up an extendable research avenue for researchers in deep learning based intelligent system leveraging the principles of kernel theory. Experiments show that the proposed method consistently improves the generalization performances of existing deep core vector machine.

1. Introduction

In recent past, the application domains of machine learning have been extended over many highly complicated and continuously varying functions like speech perception, visual object or pattern recognition, language understanding etc. The recent breakthroughs in these applications have been made possible through the recent advancement in machine learning paradigm called deep learning. The performance of learning algorithm on a particular application is highly influenced by the choice of data representation being used. The motivating factor that favor deep learning is that it gives due importance in both data representation and classification methods. In general, deep learning is a form of representation learning that attempts to build high-level abstractions in data using learning model composed of multiple non-linear transformations (Bengio, Courville, & Vincent, 2013; Bengio et al., 2009). The fast learning algorithm for multilayer neural networks by Hinton (Hinton, Osindero, & Teh, 2006) was the breakthrough in deep learning. The other contributions such as greedy layer wise training (Bengio, Lamblin, Popovici, Larochelle et al., 2007), sparse representation with energy-based model (Ranzato, Poultney, Chopra, & LeCun, 2006), sparse representation for deep belief model (Lee, Ekanadham, & Ng, 2008), convolutional neural networks (Lee, Grosse, Ranganath, & Ng, 2009) etc. makes deep learning model more effective and popular. The techniques developed from deep learning research have been widely used in many real world applications such as speech emotion recognition (Deng, Zhang, Marchi, & Schuller, 2013), sentence modeling (Kalchbrenner, Grefenstette, & Blunsom, 2014), face recognition (Lawrence, Giles, Tsoi, & Back, 1997), action recognition (Du, Wang, & Wang, 2015), etc. The training process in deep learning architecture involves highly non-linear optimizations and many heuristics for gradient-based learning, which gets trapped in local optimal solutions due to the non-convexity of the optimization problem.

* Corresponding author.
E-mail addresses: afzal.res15@iiitmk.ac.in (A. A.L.), asharaf.s@iiitmk.ac.in (A. S.).

https://doi.org/10.1016/j.eswa.2017.11.058
0957-4174/

On the other hand, the kernel machines succeeded in attaining more attention because of its ability to eliminate local optima and guaranteed global optimum with the help of convex loss functions. Kernel machines leverage the concept of kernel trick for the non-linear transformation of data samples. Support Vector Machines (SVM) (Cortes & Vapnik, 1995), Core Vector Machines (CVM) (Tsang, Kwok, & Cheung, 2005), Kernel based Principle Component Analysis (Kernel PCA) (Rosipal, Girolami, Trejo, & Cichocki, 2001), etc., are the common kernel based learning algorithms. In recent years, some of the researchers turn their attention towards the development of kernel machines with deep learning capabilities (Cho & Saul, 2009; Wilson, Hu, Salakhutdinov, & Xing, 2016). Recently developed multilayer arc-cosine kernel (Cho & Saul, 2009) enrich the kernel machines with multiple layers of kernel computation. Arc-cosine kernel may have different activation value or degree that shapes its behavior. A multilayer arc-cosine kernel with same or different activation values in each layer mimics the computations in multilayer neural network and it has been well evaluated with popular kernel machines like support vector machines and core vector machines.

The multilayer feature learning perceptiveness of deep learning architecture have been re-created in kernel machines through the model called Multilayer Kernel Machines(MKMs) (Cho & Saul, 2009). It includes multiple layers of kernel PCA based feature extraction and a deep support vector machine (DSVM)[1] as the final classifier. The kernel PCA based feature extraction involves only a single fixed kernel computation. The generalization performance of these machines heavily depends on the kernel being selected and this limitation has been addressed by the linear or convex combination of base kernels called Multiple Kernel Learning (MKL) (Bach, Lanckriet, & Jordan, 2004). Traditional MKL algorithms follow supervised learning strategy which required class labels. However, the vast amount and high availability of unlabeled data demands an unsupervised method for the combination of multiple kernels. In the paper (Zhuang, Wang, Hoi, & Lan, 2011), the authors have proposed an unsupervised multiple kernel learning method which evaluate the linear combination of multiple single layer kernels purely from unlabeled data. Even though this method has been widely used in many applications, the possibility of combining single layer and multi layer kernels in an unsupervised method and their application in the context of feature extraction need much more exploration.

The other major problem faced by these MKMs are its high computational complexity caused by the deep support vector machine used as the final classifier. DCVMs are often formulated as in quadratic form (Cortes & Vapnik, 1995; Ji & Sun, 2013) and it makes the computation very complex and expensive[2] for the applications that involves large size datasets. The concepts of core vector machine (Tsang et al., 2005) and its subsequent works (Asharaf, Murty, & Shevade, 2006; 2007; Tsang, Kwok, & Zurada, 2006; Tsang, Kocsor, & Kwok, 2007) were mainly focused on the scalability aspects of shallow based support vector machines. The scalability aspects of deep support vector machines were addressed by deep core vector machines (Afzal & Asharaf, 2017) and it has been modeled by coupling the multilayer arc-cosine kernel with the core vector machines. However further exploration is required to evaluate the feasibility of deep core vector machines to address the scalability aspects of MKMs.

The other potential learning strategies that elucidate the possibilities of deep learning in Kernel machines is Multi Layer Multiple Kernel Learning (MLMKL). In MLMKL, the convex or linear combination of all the base kernels in antecedent layers is passed as new inputs to the base kernels in subsequent layers. These methods have been successfully used with support vector machines and they often demonstrated promising results (Rebai, BenAyed, & Mahdi, 2016). However, in this layered kernel machines the possibilities of using unsupervised MKL with multilayer kernel and the scalable deep core vector machines has to be explored further. The main facts from the above analysis of related methodologies has been summarized in Table 1.

By perceiving the above facts, this paper explores the possibility of building a deep core vector machine with multiple layers of feature extractions. Each layer in this multilayer framework exploits kernel PCA for feature extraction. Kernel PCA is modeled by leveraging the combination of both single and multilayer kernels in an unsupervised fashion.

The rest of the paper is organized as follows. The basic idea of deep kernel machine has been given in Section 2. Unsupervised multiple kernel learning has been explained in Section 3. The proposed Deep multiple multilayer kernel learning in core vector machines has been included in Section 4. Experimental details and performances of proposed method are shown in Section 5. Section 6 gives conclusions.

2. Deep Kernel machines

Data representation learning plays a key role in the success of machine learning algorithm because it make easier to extract useful information when building classifiers or other predictors (Bengio et al., 2013). In order to amplify the scope and ease of applicability of machine learning, it would be highly desirable to build a learning model that should exploit multilayer feature learning for the deep representation of data. The number of feature learning layers or depth is one among other evaluating criteria in these models.

Deep learning is the recent popular learning architecture in neural network that gives due respects in both data representation and classification or prediction task. Deep learning models extract more useful, abstract data representation through multiple nonlinear transformations of data samples. The deep learning architecture has an appeal of hierarchical distributed representations and the potential for combining unsupervised and supervised methods (Bengio et al., 2007; Hinton, 2007; Hinton et al., 2006; Lee et al., 2009; Salakhutdinov & Hinton, 2009). In the area of artificial neural network, the deep learning methods have already proved its efficiency in many signal and information processing applications (Hinton et al., 2012; Miao, Metze, & Rawat, 2013; Zhong, Liu, & Liu, 2011).

Inspiring from these success of deep learning techniques in neural network, there have been many attempts to build deep kernel machines. The first notable work in this context is the multilayer arc-cosine kernel (Cho & Saul, 2009) that initiates deep learning capability in kernel machines. This arc-cosine kernel is the crux of many popular deep kernel machines such as deep support vector machine and and its scalable deep core vector machines.

2.1. Multi-layered deep kernel: arc-cosine kernel

A breakthrough in modeling a kernel machine that mimics deep neural network was the optimization of multilayer arc-cosine kernel (Cho & Saul, 2011). The arc-cosine kernel behaves differently with different activation values and hence represents a family of kernels. According to Cho and Saul (2009), the nth order kernel in

[1] Support vector machines with arc-cosine kernel.

[2] Solving the quadratic problem and choosing the support vectors involves the order of n^2, while solving the quadratic problem directly involves inverting the kernel matrix, which has complexity on the order of n^3. It has a space complexity of $O(n^2)$, for 'n' training samples. If a deep kernel machine is modeled with a multilayer arc-cosine kernel (say $l - layers$) then computational complexity becomes $O(l*n^3)$.

Table 1
Main pros and cons of related methodologies.

Learning Models	Pros	Cons
Deep neural network learning architecture	It gives due respect in both data representation and classification or prediction tasks. The automatic feature extraction reduces the risk of feature engineering. Layer-wise training method is adopted for reducing the complexity of training. It has best-in-class performance on problems on unstructured media like text, sound and images. It handles both labeled and unlabeled data effectively.	Because of the non-convexity in optimization they do not guarantee global optimization. The degree of non-linearity is controlled heuristically by number of hidden nodes and layers - it does not involves structural risk minimization.
Deep support vector machines	The presence of structural risk minimization techniques entitle the DSVM to control the no-linearity automatically by adjusting the number of support vectors during training. The appearance of convex optimization guarantee global solution. Multilayer arc-cosine kernel has been used to apply no-linear projection into higher dimensional space. It relies on the concept of margin maximization and hence reduce the risk of over-fitting.	DSVM encounters bottlenecks on massive datasets due to its high computational complexity (cubic in the number of training points). It deals only with labeled dataset (Supervised learning model).
Multilayer kernel machines	The kernel PCA based feature extraction layers support hierarchical data representation. The top level classifier, deep support vector machine, used in this frame work retains all the above said advantages of DSVM.	The main pitfalls of these models are its fixed, single kernel computing and high training cost of deep support vector machine. Selection of task specific choice of kernel is the another big issue in this frame work.
Deep core vector machine	It has been modeled as a scalable deep support vector machine by exploiting the minimum enclosing ball and $(1+\epsilon)$ approximation algorithm. DCVM achieves smaller asymptotic complexity than DSVM -reduces the cubic time complexity of DSVM as linear in number of training samples (m) and a space complexity that is independent of m. It retains all other advantages of deep support vector machines such as convex optimization , structural risk minimization, margin maximization, etc.	The core vector machines can operate only on certain kernel that satisfies $k(x_i, x_i) = k$, a constant - Minimum Enclosing Ball (MEB) problem in kernel-induced feature space for the kernel function k should satisfies the condition $k(x_i, x_i) = c$, a constant. Multilayer feature extraction capabilities of DCVM has to be explored further.
Multiple kernel learning	Instead of single, fixed kernel computation it uses linear or convex combinations of base kernels. MKL can be implemented in both supervised and unsupervised manner.	Feasibility of single and multilayer kernel combination has not yet been explored

the family of arc-cosine kernel can be expressed as:

$$k_n(x,y) = \quad 2\int dw \frac{e^{-\frac{\|w\|^2}{2}}}{(2\Pi)^{\frac{d}{2}}}\Theta(w.x)\Theta(w.y)(w.x)^n(w.y)^n \tag{1}$$

where the Heaviside function $\Theta(z) = \frac{1}{2}(1 + sign(z))$. The simplified form of arc-cosine kernel with angular dependency $AD_n(\theta)$ can be expressed as :

$$k_n(x,y) = \frac{1}{\pi}\|x\|^n\|y\|^n AD_n(\theta) \tag{2}$$

where

$$AD_n(\theta) = (-1)^n(sin\theta^{2n+1})(\frac{1}{sin\theta}\frac{\partial}{\partial\theta})^n(\frac{\pi-\theta}{sin\theta}) \tag{3}$$

The value of $AD_n(\theta)$ depends on two parameters: the degree or activation value n and the angle θ between the input vectors, say x and y .

$$\theta = \cos^{-1}\left(\frac{(x.y)}{||x||||y||}\right) \tag{4}$$

The computation of $AD_n(\theta)$ has been shown in Algorithm 1 .

This algorithm is limited for first three expressions of $J_n\theta$. The Eq. (2) represent the computation of single layer arc-cosine kernel having an activation value n. In the case of multilayer arc-cosine kernel the value of n may be different in different layers. Let n_1, $n_2..n_l$ be the activation values corresponding to the layers 1, 2, ..l respectively. Now the l layered arc-cosine kernel can be represented as:[3]

$$k^{(L)}_{(n1,n2..nl)}(x,y) =< \phi_{nl}(\phi_{nl-1}(\dots\phi_1(x))), \phi_{nl}(\phi_{nl-1}(\dots\phi_1(y))) > \tag{5}$$

[3] $<.,.>$ represent inner product.

Algorithm 1: Algorithm for computing Angular dependency.

```
Angular_depen (n, θ)
  inputs : Degree or activation value:n; Angle between
           input vectors θ
  output: The angular dependency:AD_nθ
  begin
    switch n do
      case 0
        AD_nθ = π − θ
      case 1
        AD_nθ = sin(θ) + (π − θ) × cos(θ)
      case 2
        AD_nθ = 3 × sin(θ) × cos(θ) + (π − θ) × (1 + 2 ×
        cos(θ)^2)
  return AD_nθ;
```

The multi layer arc-cosine kernel can be computed recursively as :

$$k^{(L+1)}_n(x,y) = \frac{1}{\Pi}\left[k^{(L)}(x,x)\ k^{(L)}(y,y)\right]^{\frac{n}{2}} J_n(\theta^{(L)}_n) \tag{6}$$

The layered kernel learning process start with initial kernels: $k^0(x_i,x_i) =< x_i,x_i >$, $k^0(x_j,x_j) =< x_j,x_j >$ and $k^0(x_i,x_j) =< x_i,x_j >$.

Since the execution of arc-cosine kernel depends on the degree of kernel at each layer, it may have different behavior on different execution instance of its iterative function. The working procedure of multilayer arc-cosine kernel has been summarized in Algorithm 2. In this algorithm, the sub function *Angular_depen* (Algorithm 1) is used to compute the angular dependency $AD_n\theta$.

The deep kernel learning in kernel machine can be modeled efficiently by plugging this multilayer arc-cosine kernel in any ker-

Algorithm 2: Computing multi layered arc-cosine kernel.

MLArccosine $(x, y, [act\_list])$
inputs : Two data samples x and y; The list of activation or degree in each layer $[act\_list]$
output: The kernel value denoted by $k^l(x, y)$
begin
$k_{ii} \leftarrow$ ***Inner-product***(x, x);
$k_{jj} \leftarrow$ ***Inner-product***(y, y);
$k_{ij} \leftarrow$ ***Inner-product***(x, y);
foreach $n \in [act - list]$ **do**
$\theta \leftarrow \cos^{-1}\left(\frac{k_{ij}}{\|k_{ii}\|\|k_{jj}\|}\right)$;
$AD_n\theta \leftarrow$ **Angular_depen**$(\mathbf{n}, \theta)$;
$K_{ij} = \frac{1}{\pi} \times (k_{ij})^{\frac{n}{2}} \times AD_n\theta$;
$K_{ii} = \frac{1}{\pi} \times (k_{ii})^{n} \times AD_n\theta$;
$K_{jj} = \frac{1}{\pi} \times (k_{jj})^{n} \times AD_n\theta$;
$k^l_{x,y} \leftarrow k_{i,j}$;
return $k^l_{x,y}$;

nel machines. One such well known example in this context is the deep support vector machines (DSVMs) (Cho & Saul, 2009). Even though DSVM shows better performance on many machine learning problems it encounters bottleneck, due to its high computational cost, on many real world application that involve massive data. The scalability aspects of support vector machines has been well addressed by the concept of core vector machines. The feasibility of using arc-cosine kernel in core vector machines were discussed in Afzal and Asharaf (2017). This scalable deep kernel model is also showing its better performance on many benchmark datasets. However, the exploitation of multilayer feature extraction capability of deep neural networks in this model required further exploration.

3. Unsupervised multiple kernel learning with multilayer kernels

The generalization performance of traditional kernel based feature extraction models and the classifiers were strongly depends on the choice of kernel functions being used. This often requires domain knowledge and hence the researchers need to spend considerable amount of time for selecting an appropriate kernel function for a particular application. Multiple Kernel learning (MKL) eliminate these limitations by learning a convex combination of a set of predefined kernel to obtain an optimal kernel (Bach et al., 2004). MKL reduces the burden of choosing right kernel and hence automate the kernel parameter turning. The other desirable advantage of MKL is that it allows to combine data from different sources such as audio,video and images. Traditional MKL algorithms follows supervised learning strategy. However, the availability of unlabeled data and the capability of deep learning architecture to process these unlabeled data motivated some researchers in the area of kernel machines to develop an unsupervised method to combine and obtain an optimal kernel from a set of kernels. The unsupervised multiple kernel learning has been first introduced in Zhuang et al. (2011). They have formulated the optimization task of unsupervised MKL by combining the following two intuitions on the optimal kernel $k_{ij} = k(x_i, x_j)$.

- k_{ij} can reconstruct the original training data x_i from the localized bases weighted by k_{ij} ; I.e $\left\|x_i - \sum_j k_{ij}x_j\right\|^2$ will be minimum.
- k_{ij} minimizes the distortion $\sum_{i,j} k_{ij}\left\|x_i - x_j\right\|^2$ over all training data.

In addition to this, the locality preserving principle can be exploited by using a set of local bases for each $x_i \in X$ denoted as B_i. By fixing the size of the local bases to some constant N_B, they have formulated the optimization problem for unsupervised multiple kernel learning, by combining the above intuitions as :

$$\min_{k\in\mathcal{K}} \frac{1}{2}\sum_{i=1}^{n}\left\|x_i - \sum_{x_j\in B_i} k_{ij}x_j\right\|^2 + \gamma * \sum_{i=1}^{n}\sum_{x_j\in B_i} k_{ij}\left\|x_i - x_j\right\|^2 \tag{7}$$

and its matrix equivalent as:

$$\min_{\mu\in\Delta, D} \frac{1}{2}\|X(I - K\circ D)\|_F^2 + \gamma * \text{ tr } K\circ D\circ M(11^T) \tag{8}$$

$$\text{subject to } [K]_{i,j} = \sum_{t=1}^{m}\mu_t k^t(x_i, x_j),\ 1 \le i, j \le n \text{ and}$$

$$\Delta = \left\{\mu : \mu^T\mathbf{1} = 1, \mu \ge 0\right\}$$

This optimization problem can be solved in two ways (Zhuang et al., 2011) : first by solving for μ with a fixed D (using convex optimization) and then solving for D by fixing μ (using a greedy mixed integer programming formulation). Since the proposed method uses a layered architecture, we are following the first strategy. The matrix D is computed beforehand by taking k nearest neighbors of x_i from the training set and represented by putting a one in the corresponding positions of d_i. Now the optimization problem can be rewritten as :

$$\min_{\mu\in\Delta} \frac{1}{2}\|X(I - K\circ D)\|_F^2 + \gamma * \text{ tr } K\circ D\circ M(11^T) \tag{9}$$

where γ controls the trade-off between the coding error and the locality distortion. $D \in \{0, 1\}^{n\times n}$, the local bases of each x_i as a column vector d_i. Each column vector $d_i \in \{0, 1\}^n$ has a 1 at those points j where, $x_j \in B_i$ and $\|d_i\|_1 = N_B, i = 1, 2, \ldots, n$. The matrix M, the Euclidean distance on X, is defined as : $[M]_{ij} = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$. The notation '$\circ$' denotes element wise multiplication of two matrices, $\|\cdot\|_F^2$ denotes the Frobenius-norm of a matrix and 'tr' denotes the trace of a matrix. The objective function is formulated as a convex quadratic programming problem :

$$J(\mu) = \mu^T\left(\sum_{t=1}^{m}\sum_{i=1}^{n} k_{t,i}k_{t,i}^T \circ d_i d_i^T \circ P\right)^T \mu + z^T\mu \tag{10}$$

where $[z]_t = \sum_{i=1}^{n}(2\gamma v_i \circ d_i - 2p_i \circ d_i)^T k_{t,i}$, $P = X^T X$, and $k_{t,i} = [k^t(x_i, x_1), \ldots, k^t(x_i, x_n)]^T$ is the ith column of the tth kernel matrix. p and v are columns of P and M corresponding to x_i respectively.

In the above unsupervised MKL model, only shallow based kernels have been used. The MKL with multilayer kernel can be modeled easily in conjunction with arc-cosine kernel. The arc-cosine kernel has its own layered architecture and the computation in each layer depends on both the kernel values computed by the previous layer and the activation value in the current layer. Since each layer in the arc-cosine kernel behaves like a single kernel, the multilayer arc-cosine kernel contributes a list of kernels in the MKL frame work. This unsupervised MKL with multilayer kernel can be used in conjunction with many kernel machines such as deep support vector machines, deep core vector machines, etc.

4. Deep multiple multilayer kernel learning in core vector machines

The Multilayer Kernel Machine (MKM) proposed in Cho and Saul (2009) has an architecture similar to that of neural network based deep learning machines. In this architecture, the feature extraction layers were modeled with kernel PCA followed by a supervised feature selection method and the final layer classifier was modeled with deep support vector machine (DSVM). While analyzing, we have noticed two limitations in this architecture. The fixed kernel used in this architecture does not guarantee the optimum choice of the kernel for the task being modeled. The quadratic formulation of SVM, in the output layer, impose a high computational complexity in many real world application that involve large size datasets. To address these limitations of existing MKMs, in this section we are proposing a new scalable deep kernel learning model, a deep core vector machines with unsupervised multiple multilayer kernel learning based feature extraction layers. In our proposed method, first we formulate an unsupervised multiple kernel learning framework which involves both single and multilayer kernels. This MKL framework is then used to revamp the kernel PCA with multiple kernel computation. Finally, the existing MKM is restructured with multiple layers of revamped kernel PCA as feature extraction layers and deep core vector machine as the output layer classifier. The details are given in the following paragraphs.

In the Unsupervised Multiple Kernel Learning (UsMKL) model shown in Eq. (7) uses only shallow based kernels. The recently developed arc-cosine kernel has a multilayer structure by itself. The behavior of each layer in arc-cosine kernel is often described by the given activation value and hence it expose multiple kernels by itself. Even though the arc-cosine kernel, the multilayer kernel has been well evaluated in many classification and regression task, their applicability in the context of MKL as feature extraction methods has not yet been addressed in the machine learning literature. By observing these facts, we are modeling an Unsupervised Multiple Multilayer Kernel Learning (UsMM$_l$KL) framework by using arc-cosine kernel in conjunction with UsMKL model. Each instance of arc-cosine kernel in our MKL framework can be evaluated with its associated set of activation values. If multilayer arc-cosine kernels have been introduced in UsMKL model, then $[K]_{i,j} = \sum_{t=1}^{m} \mu_t k^t(.,.)$ computation in Eq. (8) will add another recursive kernel computations for each occurrence of the arc-cosine kernel. The depth of such recursive computation is determined by the size of list of activation values associated with each instance of arc-cosine kernel. The kernel matrix associated with the arc-cosine kernel can be computed by using Algorithm 2. This UsMM$_l$KL can easily be applied in kernel PCA to form a multiple multilayer kernel learning based feature extraction layer. The multiple feature extraction layers can be formed by stacking this unsupervised multiple multilayer kernel based kernel PCA models in an hierarchical fashion.

The second problem of MKMs, the scalability aspects of DSVM has been well addressed by deep core vector machines (DCVM), core vector machines with arc-cosine kernel. In core vector machines the quadratic optimization problem of SVMs has been reformulated as an equivalent minimum enclosing ball problem and then solved by a core set obtained by a faster $(1+\epsilon)$ approximation algorithm. The formulation of deep core vector machines have been well explained in our previous paper (Afzal & Asharaf, 2017). This DCVM can easily be used as the output layer classifier in the multi layer kernel machine with unsupervised multiple multilayer kernel learning. The overall architecture of the proposed framework is shown in Fig. 1. The architecture consists of multiple layers, with each layer performing feature extraction using kernel PCA in an UsMM$_l$KL method followed by supervised feature selection. The supervised feature selection allows to retain

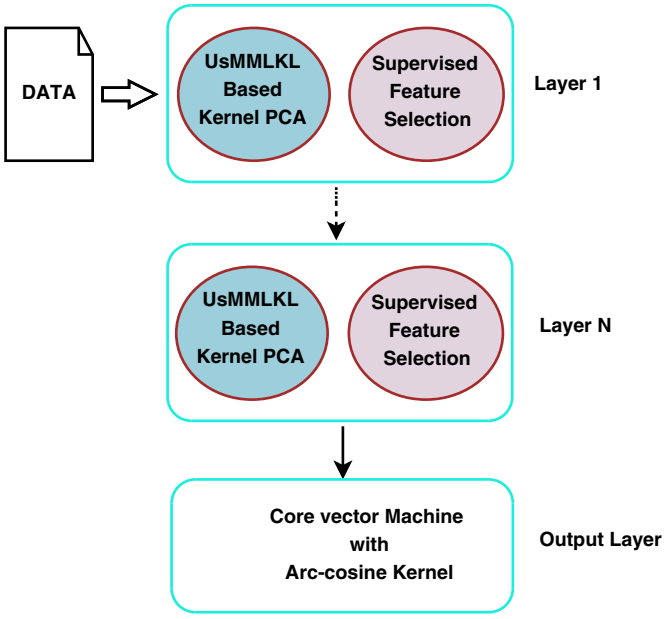

Fig. 1. Deep core vector machines with multiple layers of feature extraction. Each kernel PCA based feature extraction layer is modeled by leveraging the convex combination of both single and multilayer kernels in an unsupervised manner.

only those set of features that are most relevant for the task being modeled. Like deep neural network learning architecture, after $\mathcal{L}$ layers of feature extraction, the refined data is given to the scalable core vector machines with arc-cosine kernel. The proposed method is summarized in Algorithm 3. In this algorithm, the sub

Algorithm 3: Algorithm for computing scalable Multilayer kernel machines with Unsupervised Multiple multilayer kernel PCA.

Deep_UsMM$_l$KL $(X, y, \mathcal{L}, KL, N_B, \gamma)$
- **inputs** : Datset X, Labels y, Number of layers $\mathcal{L}$, List of Kernels KL - $\mathcal{L}$ x m matrix in which $\mathcal{K}^{(l)} = \{k_1^{(l)}, k_2^{(l)}, k_m^{(l)}\}$ is the list of kernels in layer l, Size of the local bases N_B, Turning parameter γ
- **output**: Optimal kernel weight μ^l for each layer and prediction accuracy
- **begin**
  - $X_{new} \leftarrow X$
  - **foreach** l *in range*$(\mathcal{L})$ **do**
    - $\mu^l \leftarrow$ UsMM$_l$KL(X_{new}, N_B, γ);
    - $K_{new} = \sum_{t=1}^{m} \mu_t^l * K_t^{(l)}$;
    - Perform Kernel PCA with K_{new};
    - $X_{new} \leftarrow X$ with selected features;
  - Final classification with Deep core vector machine (X_{new}, Y).
- **return** $AD_n\theta$;

function UsMM$_l$KL () represent Unsupervised Multiple Multilayer Kernel Learning model for computing the optimum kernel weight.

The proposed multilayer learning model combine kernels in an unsupervised manner and uses a supervised core vector machine as the final classifier. The main characteristic of this proposed model is that it exhibit the prominent features of deep neural networks such as the hierarchical representation of data and the capability of combining both supervised and unsupervised methods. However, unlike deep neural networks, this model ensure global optimum with its convex loss function. The other theoretical ad-

Table 2
The composition of datasets taken from both libsvm and UCI repositories.

Sl.No.	Dataset	#Clss	#Dim	#Train	#Test
1	Australian	2	8	512	256
2	Breast-cancer	2	10	400	283
3	diabetes	2	8	512	256
4	Ijcnn1	2	22	49,990	91,701
5	Letter	26	16	15,000	5000
6	Optdigit	9	64	3823	1797
7	Satimage	6	36	4435	2000
8	Pendigit	10	16	7494	3498
9	Usps	10	256	7291	2007

Table 3
The activation list of arc-cosine kernel obtained during cross validation phase.

Sl. no	Dataset	Activation list	
		Unnormalized data	Normalized data
1	Australian	[0,2]	[0,2]
2	Breast-Cancer	[0,1,2]	[0,2]
3	Diabetes	[0,1]	[0,2]
4	Ijcnn1	NA	[0,2]
4	Letter	[0,1,2]	[0,2]
5	Optdigit	[0,2]	[0,1,2]
6	Pendigit	[0,2]	[0,2]
7	Satimage	[0,1]	[0,1,2]
8	Usps	NA	[0,2]

vantage of this model is the margin maximization, the key feature of SVM/CVM, that reduces the risk of over-fitting. The use of CVM as the final classifier in our model resolves the scalability problem of exciting MKMs to some extent. The main problem faced by our model is the multiple use of multilayer kernel which elevate the structural complexity of the model. The other noticeable issue of this approach is the restriction on the kernel functions possible($k(x_i, x_i) = c$, a constant) due to the final classifier, the CVM.

5. Experimental result

The datasets and their compositions used in our experiments are given in Table 2. It include both binary and multi-class datasets having varying size and dimensions. All the datasets except Ijcnn1 and Usps were taken from UCI machine learning repository (Blake & Merz, 1998) and Ijcnn1 and Usps were taken from libSVM repository (Chang & Lin, 2011). The feature extraction module, the kernel PCA based unsupervised multiple multilayer kernel learning has been implemented in python 2.7.6 and deep core vector machine (DCVM), the final layer classifier, has been implemented in C++ .[4] The python package *subprocess* was used to invoke DCVM within the python framework.

In the training phase, we choose 2500 or entire training dataset, whichever is less, for cross validating the activation list of arc-cosine kernel from all the combination of degree '0','1', '2' and '3'. The core vector machine with arc-cosine kernel was used as classifier in this case. The activation list corresponding to the maximum accuracy obtained in this validation phase were selected as the default activation list for arc-cosine kernel in the subsequent process. The Table 3 shows the selected activation list against the dataset used.

In each layer of feature extraction module, we set apart 3000 or entire training data points, which ever is less, for computing the optimal kernel weight in an unsupervised manner. The list of kernels for the multilayer feature extraction were given as a matrix. The *i*th row in this matrix represent the list of kernels for *i*th layer of feature extraction module. Even though any kernel combinations can be used, we have used an arc-cosine kernel in between two RBF kernels as the kernel combination in each layer. After obtaining the optimal kernel value, the entire datasets were used for feature extraction. The feature selection was done by using univariate feature selection method available in scikit-learn[5] library (Pedregosa et al., 2011). Based on ranking produced by the selection method, top five percent of features were selected, since empirically it was giving a consistent performance. Final classification were carried out by core vector machine with arc-cosine kernel. In all the experiments, we have recorded the standard metric measures such as precision, recall, F1-Score and accuracy.

5.1. Performance evaluation

The performance of proposed method has been evaluated on both normalized and unnormalized form of all the datasets except Ijcnn1 and Usps. The dataset name suffixed with the word *scaled* indicates normalized dataset.Three well known techniques such as *Standard Scalar, Robust Scalar* and *MinMax scalar* (Pedregosa et al., 2011) were used for normalizing data samples and the variation in prediction accuracy has been shown in Table 4. We have experimented with up to 3 layers of feature extraction, for evaluating the performance of multilayer computation. The L1, L2 and L3 columns in Table 4 represent the prediction accuracy of each normalization techniques with one , two and three layers of feature extraction respectively. The prediction accuracy of proposed method on all the dataset without using any normalization techniques has been shown in Table 5. The variation in accuracy with different layers of feature extraction has been illustrated by the graph shown in Fig. 2.

The classification report including average precision, recall, F-measures and the prediction accuracy of proposed method and deep core vector machine on both normalized and unnormalized dataset have been shown in Table 6. In this table, the classification report of proposed method represent the best among those layer

[4] DCVM has been implemented by combining the features from both the packages libCVM-2.2 (Tsang, Kocsor, & Kwok, 2011) and libSVM-2.91.

[5] Available at http://scikit-learn.org/stable/modules/feature_selection.html.

Table 4
Layer wise prediction accuracy of the proposed method with different normalization methods .

Sl. no	Dataset	Standard scaler			Robust scaler			MinMax scaler		
		L1	L2	L3	L1	L2	L3	L1	L2	L3
1	Australian_scale	86.522	86.522	**86.957**	80.435	81.305	85.478	81.304	81.304	82.609
2	Breast-Cancer_scale	97.880	**98.233**	98.233	98.233	97.879	98.233	97.527	97.527	97.527
3	Diabetes_scale	72.656	73.438	73.047	71.875	73.437	**78.516**	74.609	76.563	77.73
4	Letter_scale	96.74	96.79	**97.73**	89.58	90.22	90.52	95.02	95.22	95.36
5	Optdigit_scale	97.6628	97.7184	97.997	96.605	96.995	97.496	99.054	99.109	**99.221**
6	pendigit_scale	99.399	99.5426	99.571	99.285	99.371	99.399	99.399	99.543	**99.628**
7	satimage_scale	92.7	93.2	**93.5**	93.05	93.05	93.2	90.85	91.0	90.8

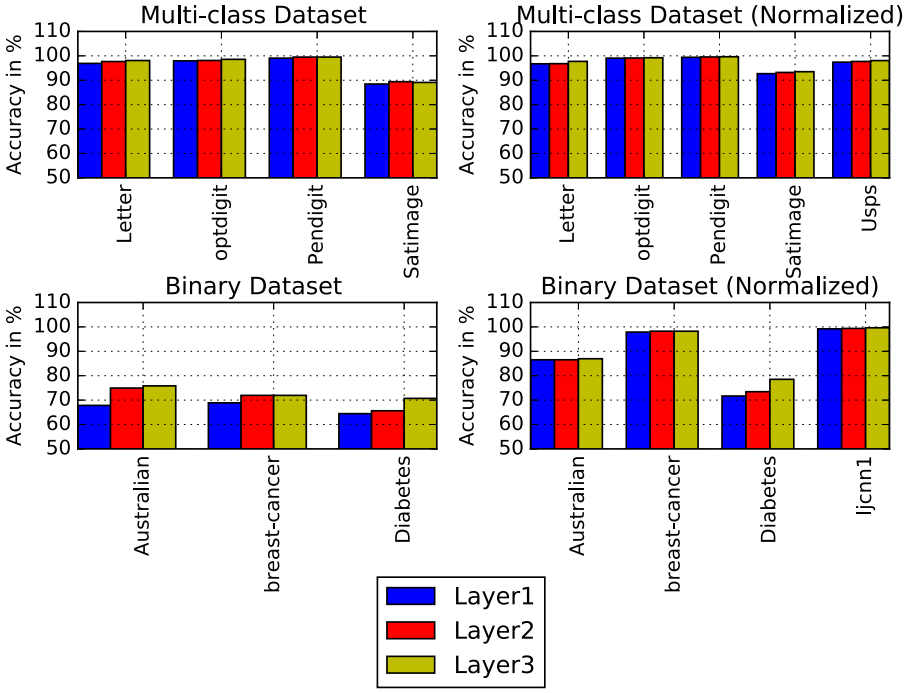

Fig. 2. A comparison in accuracy with different number of feature extraction layers.

Table 5
The layer wise prediction accuracy with out using normalization method.

Sl.No	Dataset	Prediction accuracy		
		L1	L2	L3
1	Australian	67.83	74.94	**75.831**
2	Breast-Cancer	68.87	**71.937**	71.937
3	Diabetes	64.453	65.625	**70.703**
4	Letter	96.9	97.67	**98.082**
5	Optdigit	97.941	98.108	**98.564**
6	pendigit	99.057	**99.493**	99.493
7	satimage	88.45	**89.4**	89.1
8	ijcnn1 (Scaled to [−1 1])	99.199	99.370	**99.637**
9	Usps (scaled to [−1 1])	97.409	97.708	**98.057**

wise performances shown in above tables. The bold face values indicate the highest accuracy on each dataset.

From Table 6 and Fig. 2, following observations have been made.

- In all the dataset, proposed method possess best result in average precision, recall and F1-measures.
- Proposed algorithm consistently improves the accuracy of single layer deep core vector machines.
- In most of the datasets, the proposed algorithm achieves higher accuracies as the number of layers increased.
- As in the case of DCVM, our method also attains higher accuracies on normalized datasets.
- Proposed method also shows its potential in the context of multilayer unsupervised feature extraction, as in the case of deep learning models.

This approach was carried out on a CPU machine configured with AMD Opteron 6376 @ 2.3 GHZ / 16 core processor and 16GB RAM. Exploitation of the parallelization possible in computation using GPU or distributed computing platform would have reduced the training time of this approach. This is a potentially possible dimension for the empirical exploration.

Table 6
The generalization performance in terms of Average Precision, Recall, F-core and overall prediction accuracy of Deep CVM and proposed method.

Sl.No	Dataset	DCVM				Proposed method			
		Preci	Reca	F1-sc	Acc	Preci	Reca	F1-Sc	Acc
1	Australian	.754	.52	.753	75.211	.773	.758	.765	**75.831**
2	Breast-Cancer	.784	.675	.569	67.49	.793	.696	.741	**71.937**
3	Diabetes	.671	.684	.673	68.35	.694	.707	.697	**70.703**
4	Letter	.969	.969	.969	96.9	.981	.981	.981	**98.082**
5	Optdigit	.977	.977	.977	97.663	.987	.985	.986	**98.564**
6	pendigit	.984	.984	.984	98.371	.996	.995	.995	**99.493**
7	satimage	.855	.858	.852	85.8	.891	.894	.892	**89.4**
Scaled dataset									
8	Ausralian_scale	.857	.857	.857	85.65	.870	.870	.869	**86.957**
9	Breast-cancer_scale	.961	.961	.961	96.11	.983	.982	.982	**98.233**
10	Diabetes_scale	.760	.766	.757	76.56	.784	.785	.779	**78.516**
11	ljcnn_scale	.990	.991	.990	99.001	.993	.997	.995	**99.637**
12	Letter_scale	.967	.967	.967	96.66	.981	.976	.978	**97.73**
13	Optdigit_scale	.978	.977	.977	97.718	.992	.992	.992	**99.221**
14	Pendigit_scale	.982	.982	.982	98.19	.996	.996	.996	**99.628**
15	Satimage_scale	.920	.921	.920	92.15	.935	.935	.934	**93.5**
16	Usps_scale	.952	.952	.952	95.217	.981	.981	.981	**98.057**

6. Conclusions

This paper proposes a deep core vector machine with multiple layers of feature extraction. Each feature extraction layer is modeled with an unsupervised multiple kernel learning framework having both single and multilayer kernels. This deep kernel learning model addresses the limitations such as fixed kernel computation problem and scalability issues of existing multilayer kernel machines. The convex combination of both single and multilayer kernels using the unsupervised MKL formulation resolves fixed kernel computation. This unsupervised MKL framework is then used to revamp the Kernel PCA for feature extraction. The scalability problem is addressed by using deep core vector machine as the final classifier. The empirical results show that the proposed method considerably outperform conventional core vector machine with arc-cosine kernel. The improvement in accuracy as the num-

ber layers increased shows the potential of multiple layers of feature extraction. The proposed learning architecture modeled with multiple layers of unsupervised feature extraction followed by a scalable final classifier possess a deep learning perceptiveness in kernel machines.

6.1. Future works

The theoretical and empirical analysis of this paper list out some insightful future directions to explore. a) The generalization performance of our model is highly influenced by the structure of multilayer MKL framework. The number of layers and appropriate base kernels contribute this multilayer MKL structure. A theoretical study on the optimal structure of multilayer MKL model is a potential future direction. b) A study on feasibility of deep multilayer kernel learning approach on structured output prediction task that involves very complex decision function is another interesting research problem. c) The multiple kernel learning based feature extraction involves extensive amount of matrix manipulation. The scalability aspects in this context is another potential future direction. d) The deep learning algorithms are found to be effective for memory related models such as Recurrent neural network, Memory network, Turing neural network etc. A study on the formulation of recurrent kernels that mimics the above models is another explorable direction.

References

Afzal, A. L., & Asharaf, S. (2017). Deep kernel learning in core vector machines. *Pattern Analysis and Applications*, 1–9.

Asharaf, S., Murty, M. N., & Shevade, S. K. (2006). Cluster based core vector machine. In *Sixth international conference on data mining (icdm'06)* (pp. 1038–1042). IEEE.

Asharaf, S., Murty, M. N., & Shevade, S. K. (2007). Multiclass core vector machine. In *Proceedings of the 24th international conference on machine learning* (pp. 41–48). ACM.

Bach, F. R., Lanckriet, G. R., & Jordan, M. I. (2004). Multiple kernel learning, conic duality, and the smo algorithm. In *Proceedings of the twenty-first international conference on machine learning* (p. 6). ACM.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence, 35*(8), 1798–1828.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems, 19*, 153.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning, 2*(1), 1–127.

Blake, C., & Merz, C. J. (1998). Uci repository of machine learning databases [http://www. ics. uci. edu/∼mlearn/mlrepository. html]. irvine, ca: university of california. *Department of Information and computer science, 55*.

Chang, C.-C., & Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST), 2*(3), 27.

Cho, Y., & Saul, L. K. (2009). Kernel methods for deep learning. In *Advances in neural information processing systems* (pp. 342–350).

Cho, Y., & Saul, L. K. (2011). Analysis and extension of arc-cosine kernels for large margin classification. arXiv preprint arXiv:1112.3712.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning, 20*(3), 273–297.

Deng, J., Zhang, Z., Marchi, E., & Schuller, B. (2013). Sparse autoencoder-based feature transfer learning for speech emotion recognition. In *Affective computing and intelligent interaction (acii), 2013 humaine association conference on* (pp. 511–516). IEEE.

Du, Y., Wang, W., & Wang, L. (2015). Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1110–1118).

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine, 29*(6), 82–97.

Hinton, G. E. (2007). Learning multiple layers of representation. *Trends in cognitive sciences, 11*(10), 428–434.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation, 18*(7), 1527–1554.

Ji, Y., & Sun, S. (2013). Multitask multiclass support vector machines: Model and experiments. *Pattern Recognition, 46*(3), 914–924.

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A convolutional neural network for modelling sentences. arXiv preprint arXiv:1404.2188,.

Lawrence, S., Giles, C. L., Tsoi, A. C., & Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks, 8*(1), 98–113.

Lee, H., Ekanadham, C., & Ng, A. Y. (2008). Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems* (pp. 873–880).

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning* (pp. 609–616). ACM.

Miao, Y., Metze, F., & Rawat, S. (2013). Deep maxout networks for low-resource speech recognition. In *Automatic speech recognition and understanding (asru), 2013 ieee workshop on* (pp. 398–403). IEEE.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research, 12*, 2825–2830.

Ranzato, M., Poultney, C., Chopra, S., & LeCun, Y. (2006). Efficient learning of sparse representations with an energy-based model. In *Proceedings of the 19th international conference on neural information processing systems* (pp. 1137–1144). MIT Press.

Rebai, I., BenAyed, Y., & Mahdi, W. (2016). Deep multilayer multiple kernel learning. *Neural Computing and Applications, 27*(8), 2305–2314.

Rosipal, R., Girolami, M., Trejo, L. J., & Cichocki, A. (2001). Kernel pca for feature extraction and de-noising in nonlinear regression. *Neural Computing & Applications, 10*(3), 231–243.

Salakhutdinov, R., & Hinton, G. E. (2009). Deep boltzmann machines.. In *Aistats: 1* (p. 3).

Tsang, I., Kocsor, A., & Kwok, J. (2011). Libcvm toolkit. https://github.com/aydindemircioglu/libCVM.

Tsang, I.-H., Kwok, J.-Y., & Zurada, J. M. (2006). Generalized core vector machines. *IEEE Transactions on Neural Networks, 17*(5), 1126–1140.

Tsang, I. W., Kocsor, A., & Kwok, J. T. (2007). Simpler core vector machines with enclosing balls. In *Proceedings of the 24th international conference on machine learning* (pp. 911–918). ACM.

Tsang, I. W., Kwok, J. T., & Cheung, P.-M. (2005). Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research, 6*(Apr), 363–392.

Wilson, A. G., Hu, Z., Salakhutdinov, R., & Xing, E. P. (2016). Deep kernel learning. In *Proceedings of the 19th international conference on artificial intelligence and statistics* (pp. 370–378).

Zhong, S.-h., Liu, Y., & Liu, Y. (2011). Bilinear deep learning for image classification. In *Proceedings of the 19th acm international conference on multimedia* (pp. 343–352). ACM.

Zhuang, J., Wang, J., Hoi, S. C., & Lan, X. (2011). Unsupervised multiple kernel learning. *Journal of Machine Learning Research-Proceedings Track, 20*, 129–144.

BIO-DATA

1. **Name** : Afzal A.L.
2. **Age and Date of Birth** : 40 years, 31/05/1978
3. **Address for correspondence** : Salma Bhavan
   : Thundathil P O - 695581
   : kariavattom ,
   : Thiruvananthapuram, Kerala
   : Phone +919447246553
4. **e-mail address** : afzal.res15@iiitmk.ac.in, afzal.a.l@gmail.com
5. **Educational Qualifications**

Sl. No.	Examinations Passed	Institution at which studied	Year of Passing	Grade
1	B. Tech (Computer Science and Engineering)	Calicut University	2000	62%
2	M. Tech (Computer and Information Technology)	Manonmaniam Sundaranar University	2010	71.28%

6. **Experience**

Sl. No.	Particulars	Period	Institution
1	Assistant Professor in Computer Science and Engineering	July 2003 – Till date	Co-operative Academy of Professional Education (CAPE)