A Study on Deep Multi-layer Kernel Machines

A Project Report

submitted by

AKHIL P M

in partial fulfillment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF MATHEMATICS
INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY
Thiruvananthapuram - 695547

May 2016

CERTIFICATE

This is to certify that the thesis titled '**A Study on Deep Multi-layer Kernel Machines**', submitted by **Akhil P M**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, for the award of the degree of **MASTER OF TECHNOLOGY**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Sumitra S
Supervisor
Department of Mathematics
IIST

K.S S Moosath
Head of Department
Department of Mathematics
IIST

Place: Thiruvananthapuram
May, 2016

DECLARATION

I declare that this thesis titled '**A Study on Deep Multi-layer Kernel Machines**' submitted in fulfillment of the Degree of MASTER OF TECHNOLOGY is a record of original work carried out by me under the supervision of **Dr. Sumitra. S**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgements have been made wherever the findings of others have been cited.

Akhil P M
SC14M044

Place: Thiruvananthapuram
May, 2016

Abstract

Deep learning is one of the prominent class of algorithms among the representation learning techniques used nowadays, producing state of the art results in many complex pattern recognition tasks. These algorithms learn a set of abstract high-level features in a hierarchical fashion with a layerwise model. Theoretical studies indicates that a deep architecture can represent highly varying decision functions in a compact way, which if modelled with a shallow architecture may require exponentially huge number of datapoints and computational units. With the recent rise in computing power using multi-core CPUs and GPUs, fast and efficient training of deep learning algorithms became possible. But almost all of the deep learning algorithms are formulated on top of neural network based models, which is declining the popularity of kernel methods for learning representations.

In this thesis, we study multi-layer kernels, a kernel machine equivalent to deep learning algorithms, in supervised and unsupervised learning settings. In unsupervised settings, multi-layer kernels are used for feature learning. We use the formulation of Multi-layer Kernel Machines(MKMs) with KPCA algorithm, which is trained in a greedy layerwise fashion. MKMs are modified by taking a linear combination of multiple kernels in each layer, and such models are called Multi-layer Multiple Kernel Learning(ML-MKL) models. In MKL, the optimal kernel weights are found based on an unsupervised formulation.

In supervised learning settings, we use multi-layer kernels for discriminant analysis and structured output prediction problems. Discriminant analysis with kernel FDA algorithm is studied on shape classification problems, which is giving results comparable with state of the art deep learning algorithms. Multi-layer models score high when representing highly complex decision functions in a compact way. Structured output spaces need very complex decision functions in many cases. This was the primary motive for studying multi-layer kernels in structured output spaces.

Acknowledgements

Though only my name appears on the cover of this thesis, the work presented in this report would not have been possible without the constant support and encouragement of many great people.

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Sumitra S for the continuous support throughout the work and for her patience, motivation, enthusiasm and immense knowledge. I thank her, especially for showing me the beauty of great mathematical concepts, with the right mix of theory and applications. I also thank her for correcting this thesis and making it to the present form.

I owe my deepest gratitude to Dr. Asharaf S for constantly motivating me and the 'anytime' support he offered, even in his busy schedules. The hours-long discussion we used to have often was very insightful. The way he analyzed the results and gave suggestions was very helpful for me to understand the problems with my result-oriented rush.

I would like to thank Prof. Nicholas Sabu and Dr. Deepak T.G for spending their valuable time to help me with the optimization algorithms and statistical methods.

I am very thankful to Shiju S. Nair for his continuous support in multiple roles ranging from concept clearing to implementation hacks. I would like to appreciate Harsha and Govindraj for their generous support in making my understanding clear about convex functions and their optimization techniques.

I am very grateful to Prof. Raju K. George, who sourced me a quality workstation on which much of the work has been done.

I take this as an opportunity to thank all my friends in IIST who made this two years of life very memorable.

Last but not the least, I would like to thank my parents and my sister for their care, love and support throughout my life.

Contents

List of Figures

List of Tables

Abbreviations

CRF	Conditional Random Field
DBN	Deep Belif Networks
KFDA	Kernel Fisher Discriminant Analysis
MKL	Multiple Kernel Learning
MKM	Multi-layer Kernel Machines
ML-MKL	Multi-layer Multiple Kernel Learning
MR	Margin Rescaling
NNet	Neural Network with one hidden layer
PCA	Principal Component Analysis
RKHS	Reproducing Kernel Hilbert Space
SMO	Sequential Minimal Optimization
SNE	Stochastic Neighbur Embedding
SR	Slack Rescaling
SVM	Support Vector Machines
tSNE	t-distributed Stochastic Neighbur Embedding

Chapter 1

Introduction

Representation Learning is one of the key area in machine learning, where intense research works are being done spanning over decades. The choice of the representation has a considerable influence in the performance of learning algorithms ([Bengio, 2013] et al.). Representation Learning is also considered as an effective mechanism for transferring the prior knowledge of humans to the machine learning system. [Bengio, 2007] et al. broadly classified the architecture of the representation learning system into deep and shallow architectures. Shallow architectures often assumes strong task specific priors, whereas deep architectures can support broad priors. Algorithms like SVM, single layer neural network etc. come under shallow architectures, and algorithms like deep belif networks, convolutional neural networks etc. belong to deep architectures.

Deep Learning is a rapidly developing area among representation learning algorithms, which makes use of depth as a key criteria for producing good representations (hence they belong to deep architectures). *In general, Deep Learning is a set of algorithms in machine learning that attempts to model high-level abstractions in data by using model architectures composed of multiple non-linear transformations*([Bengio, 2009] et al., [Bengio, 2013] et al.). Deep Learning methods are preferred over shallow ones in many complex learning tasks such as computer vision, speech recognition etc. due to: *the wide range of functions that can be parameterized by composing weakly nonlinear transformations, the broad range of prior knowledge they can transfer to the learning system, the invariance being modelled*

by such systems against local changes in the input and their ability to learn more abstract concepts in an hierarchical fashion([Cho] et al., [Bengio, 2007] et al.).

Though the idea of using such deep architectures for learning representations was known to machine learning researchers, two main factors deterred the wide use of deep learning algorithms: (i) the gradient-based methods often gets trapped in local minima which results in poor generalization capacity, (ii) training the networks with more than 2 or 3 layers was a big computational challenge. The *greedy layerwise unsupervised pre-training* proposed by [Hinton, 2006] et al. was a breakthrough in machine learning research as it tackled the difficulty in training such deep models. Using this method a hierarchy of features can be learned one level at a time, using unsupervised techniques([Bengio, 2013] et al.). This greedy layerwise unsupervised pre-training is used to initialize the weights of the network (unsupervised pre-training stage) and then the weights are fine-tuned with the given label information(supervised fine-tuning stage).

The recent rise in computing power alleviated the second problem upto a considerable level. Efficient training methods are devised for learning networks with millions of parameters both on multi-core CPUs and GPUs([Raina] et al., [Dean] et al.). Scalable deep learning models like [Graphlab] etc. are also developed in distributed cloud based environments.

Armed with these tools deep learning witnessed a sudden boost since 2006, producing state-of-the-art results in several complex machine learning tasks like speech recognition([Hinton 2012] et al.), visual object recognition([Alex] et al., [Graham] et al., [Wan] et al.), natural language processing([Collobert] et al.) etc. However, most of the deep learning algorithms are developed on top of neural network based models([LeCun] et al., [Hinton, 2006] et al.).

Many researchers had extensively studied the limitations of kernel machines in such problems. The analysis shown in [Bengio, 2007] et al. claims that the number of training examples required for a kernel machine with Gaussian kernel may grow linearly with the number of variations (change in the directions) of the target function. [Cho] et.al explored the concept of *kernel methods based Deep Learning* by proposing Multi-layer Kernel Machines(MKMs). The architecture of MKMs

consists multiple layers with each layer performing unsupervised feature extraction using kernel PCA proposed by [Smola] et al. followed by supervised feature selection by ranking features based on their mutual information with class labels. They also proposed a multi-layer kernel function, which can mimic the computations of neural network based architecture.

Inspired by the work of [Cho] et al., multi-layer kernel machines were explored by many researchers ([Zhuang] et al., [Ilyes] et al.). Most of these researchers used multiple kernel learning (MKL) concepts in the layered architecture. Some of these works are using deep layerwise kernel models for automating the kernel learning task in SVMs. But there is less work done combining feature learning with MKMs and MKL. MKL in single layer for unsupervised feature extraction was studied by [Zhuang, 2011] et al. Much of the work done in this thesis is also focussed on MKL and MKMs for unsupervised feature learning to build a deep learning architecture for kernel machines.

1.1 Thesis Outline

The contents of this thesis are organized as six chapters.

The chapter 2 discusses the MKMs in detail, with empirical study on popular deep learning datasets. MKMs with mixed kernels are also tested on these datasets. Finally, some visualization (using tSNE algorithm) of the features learned by the model are also provided.

The chapter 3 describes the study of using MKL in MKMs. The formulation of the unsupervised MKL is given first, followed by empirical study and exploratory analysis on MKL in each layer of MKMs.

The chapter 4 is devoted to discriminant analysis with multi-layer kernels. It includes a brief summary of KFDA algorithm and the results from empirical study.

The chapter 5 contains a study of multi-layer kernels on structured output spaces. Struct SVM, a large margin generalization of SVM to structured output spaces is chosen as the candidate algorithm for study, and analysis is done on multi-label and multi-class classification problems.

The chapter 6 gives the conclusion of this thesis work, along with some potential future directions to explore.

1.2 Contributions

The main contributions of this thesis can be summarized as follows:

- An in depth study is made on multi-layer kernel machines with single and mixed kernels.
- MKL concepts are introduced in MKMs by taking a combination of kernels in each layer, following an unsupervised learning paradigm. The contribution of each kernel in the MKL at every layer is analyzed. A comparative study is done with existing deep learning algiritms.
- A study on the discriminating power of multi-layer kernels is done with KFDA algorithm. Empirical results are compared with the performance of existing deep architectures.
- A comparative study between multi-layer and single layer kernels are done on structured output spaces. The problem instances chosen for the study is multi-label and multi-class classification problems.

Chapter 2

Multi-layer Kernel Machines

Multi-layer Kernel Machines (MKMs) were first introduced by [Cho] et al. In their framework, arc-cosine kernel which itself is having a layered architecture was used. The architecture of MKMs consists of L layers and in each layer the data is subjected to unsupervised dimensionality reduction methods, viz. kernel PCA([Smola] et al.), followed by a supervised feature selection.
The MKM machines use multilayer kernel function, whose description is given in the next section. The rest of the chapter is organized as follows: section 2.2 contains the results of empirical study using MKMs, section 2.3 describes the visualization technique used to plot the distribution of digits (from [MNIST] dataset) in a low-dimensional space, section 2.4 contains the results of empirical study on MKMs with mixed kernels and concluding remarks are given in section 2.5.

2.1 Multi-layer Kernel

Corresponding to each Reproducing kernel Hilbert spaces (RKHS) $\mathcal{F}$ there exists a unique reproducing kernel $k : \mathcal{X} \times \mathcal{X} \to \mathcal{F}$, where $k(x, y) = \phi(x) \cdot \phi(y), x, y \in \mathcal{X}$ where $\mathcal{X}$ is a normed space, $\phi(x), \phi(y) \in \mathcal{F}$ and $\cdot$ is the inner product defined on $\mathcal{F}$.

Iteratively applying the mapping $\phi(\cdot)$ on the inputs x and y and then taking their inner product, we can obtain an L-layer kernel function as

$$k^{(L)}(x,y) = \underbrace{\phi(\phi(\ldots\phi(x)))}_{\text{L times}} \cdot \underbrace{\phi(\phi(\ldots\phi(y)))}_{\text{L times}}$$

Thus a L layer kernel machine consists of L RKHS's.

2.1.1 Multi-layer Composition of Polynomial and Gaussian Kernels

A polynomial kernel of degree d is defined as follows

$$k(x,y) = (x \cdot y)^d$$

Consider a two layer composition of polynomial kernel which can be computed as

$$\begin{aligned}\phi(\phi(x)) \cdot \phi(\phi(y)) &= \Big(\phi(x) \cdot \phi(y)\Big)^d \\ &= (x \cdot y)^{dd} = (x \cdot y)^{d^2}\end{aligned}$$

Thus the higher order compositions of polynomial kernels are simply polynomials of higher degree than the one from which it is constructed.
The two layer composition for gaussian kernels $k(x,y) = e^{-\lambda\|x-y\|^2}$ is given by

$$\begin{aligned}\phi(\phi(x)) \cdot \phi(\phi(y)) &= e^{-\lambda\|\phi(x)-\phi(y)\|^2} \\ &= e^{-2\lambda(1-k(x,y))}\end{aligned}$$

2.1.2 Multi-layer Composition of Arc-cosine Kernels

Let x, y be two inputs in $\mathbb{R}^d$. Define θ as the angle between them.

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

Then the kernel function computed by the arc-cosine kernel is

$$k_n(x,y) = \frac{1}{\pi} \|x\|^n \|y\|^n J_n(\theta) \tag{2.1}$$

where n is called the *degree of the kernel* and

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

$J_n(\theta)$ for n=0, 1, 2 is computed as shown below.

$$J_0(\theta) = \pi - \theta$$

$$J_1(\theta) = sin\theta + (\pi-\theta)cos\theta$$

$$J_2(\theta) = 3sin\theta cos\theta + (\pi-\theta)(1+2cos^2\theta)$$

for n=0, it takes the simple form

$$k_0(x,y) = 1 - \frac{1}{\pi} cos^{-1}\left(\frac{x\cdot y}{\|x\|\,\|y\|}\right)$$

hence the name arc-cosine kernel is given.The kernel function computed by arc-cosine kernel is related to the computation of multi-layer threshold networks as shown below.

Consider a single layer neural network with weights W_{ij} that connects the j^{th} input unit to the i^{th} output unit(see figure 2.1). The network maps input x to output $f(x)$ by applying a non-linear map

$$f(x) = g(W\cdot x)$$

where the non-linearity is described by the network‘s activation function

$$g_n(z) = \Theta(z) z^n$$

with

$$\Theta(z) = \frac{1}{2}(1 + sign(z))$$

this activation function is called one-sided polynomial activation function, whose graph for different n values is also shown in figure 2.1.

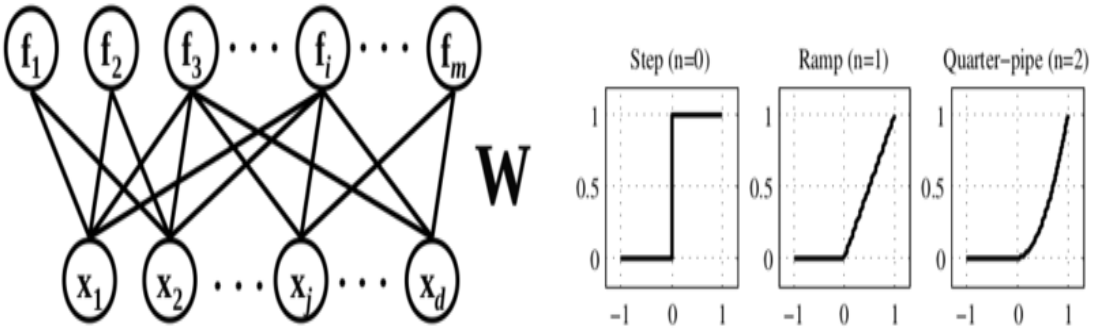

FIGURE 2.1: Single layer network and thresholded activation functions([Cho] et al.)

Let $f(x)$ and $f(y)$ be the outputs corresponding to inputs x and y. Then the inner product of $f(x)$ and $f(y)$ is

$$f(x) \cdot f(y) = \sum_{i=1}^{m} \Theta(w_i \cdot x)\Theta(w_i \cdot y)(w_i \cdot x)^n (w_i \cdot y)^n$$

Here w_i is the i^{th} row of weight matrix W and m is the no of output units. Assume W_{ij} are Gaussian distributed with zero mean and unit variance and the network has an infinite number of output units. Then

$$lim_{m\to\infty} \frac{2}{m} f(x) \cdot f(y) = k_n(x, y)$$

where

$$k_n(x, y) = 2 \int dw \frac{e^{-\frac{\|w\|^2}{2}}}{(2\pi)^{d/2}} \Theta(w \cdot x)\Theta(w \cdot y)(w \cdot x)^n (w \cdot y)^n \tag{2.2}$$

The kernel function obtained in equation 2.2 can be converted into an alternate form as in equation 2.1 with the derivation shown in [Cho] et al.

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

$$k_n^{(L+1)}(x,y) = \frac{1}{\Pi}\left[k^{(L)}(x,x)\ k^{(L)}(y,y)\right]^{\frac{n}{2}} J_n(\theta_n^{(L)})$$

where $\theta_n^{(L)}$ is the angle between images of x and y in the feature space after L layer composition

$$\theta_n^{(L)} = \cos^{-1}\left(k^{(L)}(x,y)\left[k^{(L)}(x,x)\ k^{(L)}(y,y)\right]^{\frac{-1}{2}}\right)$$

In the above formulation, we have assumed that the arc-cosine kernels have the same degree n at every layers of recursion. We can also use kernels of different degree at different layers. The intuition behind the multi-layer composition in case of arc-cosine kernels is, if the base kernel $k(x,y) = \phi(x) \cdot \phi(y)$ can mimic the computation of a single-layer network, then the iterated mapping in $k^{(L)}(x,y)$ can mimic the computation of multi-layer network([Cho] et al.).

2.1.3 MKM Architecture

The architecture of MKMs for solving classification tasks is similar to that of neural network based deep learning machines, with unsupervised feature extraction(using Kernel PCA, [Smola] et al.) followed by supervised feature selection in each layer. Figure 2.2 shows the architecture of an MKM consisting of L layers of non-linear transformations.

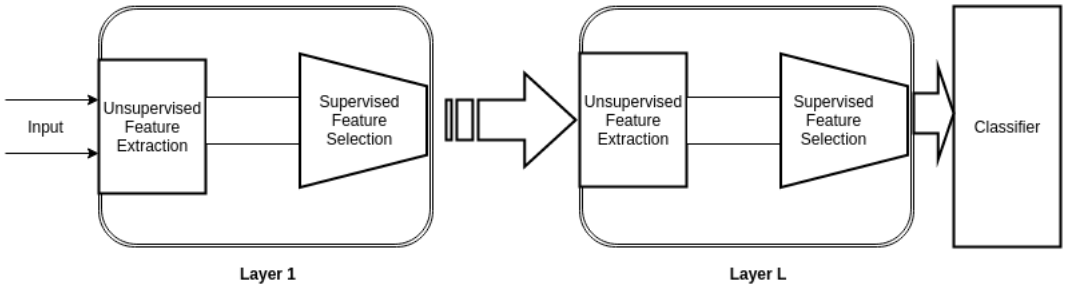

FIGURE 2.2: An MKM with L layers of transformations. Each layer consists of unsupervised feature extraction(using kernel PCA) followed by supervised feature selection.

The supervised feature selection allows us to retain only those set of features that are most informative for the pattern recognition task. The number of features to be passed to the next layer determines the width of that layer. The optimal layer width is computed by ranking the features based on their importance and selecting the required set of feature with the help of a lightweight classifier (detailed procedure is given in the experiments section). This procedure determines the architecture of the network in a greedy, layer-by-layer fashion. In their implementation [Cho] et al. used exhaustive search in the range 10 to 300 to determine the optimal set of features. The output from the final layer can be passed to any classifier. Though any kernel can be used for the kernel PCA based feature extraction, [Cho] et al. emphasized on using arc-cosine kernels due to their similarity with deep learning architecture of neural networks and the inclusion of multiple layers is significant only in case of arc-cosine kernels.

2.2 Experiments on Arc-cosine Kernel MKMs

Empirical study was conducted on four datasets, in which three were created from [MNIST] dataset of handwritten digits by adding noise in the background and one was a binary classification problem on shape images. The detailed experimental set up and a short description about datasets used is given below.

2.2.1 Experimental Set-Up

In the training phase, for each layer we set apart 10000 datapoints for training the model and 2000 datapoints for cross validating kernel parameters. For the kernel PCA we chose 3000 datapoints from the training set of 10000 images randomly. In each layer after extracting features with KPCA, we train a lightweight classifier like kNN and test the performance on the held out 2000 images. The best kernel parameter was chosen based on this performance values. With that parameter value, we did feature extraction on the entire dataset and then passed it to feature selection module. The feature selection was done by using univariate feature selection method available in [scikit-learn] library. This method produces a ranking

of features with a univariate statistical test. Based on this rank we can choose the required number of important features. We chose top 5 percent features based on this rank, since empirically it was giving a consistant performance.
In the final classification stage SVMs with arc-cosine kernels are used. The metric used for comparing the performance is, percentage loss in test dataset. Percentage loss is estimated as

$$\text{loss in percentage} = \left(1 - \frac{\#\text{correct classifications}}{\#\text{datapoints}}\right) \times 100$$

2.2.2 Mnist-back-rand Dataset

The *mnist-back-rand* dataset was created by filling the image background with random pixel values. Each pixel value of the background was generated uniformly between 0 and 255. Image size is 28×28 and the dataset contains 12000 training and 50000 testing images.

2.2.3 Mnist-back-image Dataset

The *mnist-back-image* dataset was generated by filling the image background with random image patches. The patches were extracted randomly from a set of 20 images downloaded from the internet. The dataset contains contains 12000 training and 50000 testing images, each of size 28×28.

2.2.4 Mnist-rot-back-image Dataset

The *mnist-rot-back-image* is a rotated variant of *mnist-back-image* where the rotation angle is generated uniformly between 0 and 2π. Image size and number of samples are also the same as *mnist-back-image* dataset.

2.2.5 Rectangles-image Dataset

For *rectangles-image* dataset, the classification task is to identify whether a rectangle contained in an image has larger width or length. The dataset is constructed by uniformly sampling the height and width of the rectangles. Then random image

patches are added in the background, where image patches are extracted from one of the 20 images used by *mnist-back-image*. Each image is of size 28×28 and each dataset contains 12000 training images and 50000 testing images.
Figure 2.3 contains some sample images from first three dataset and Figure 2.4 contains the samples from *rectangles-image* dataset.

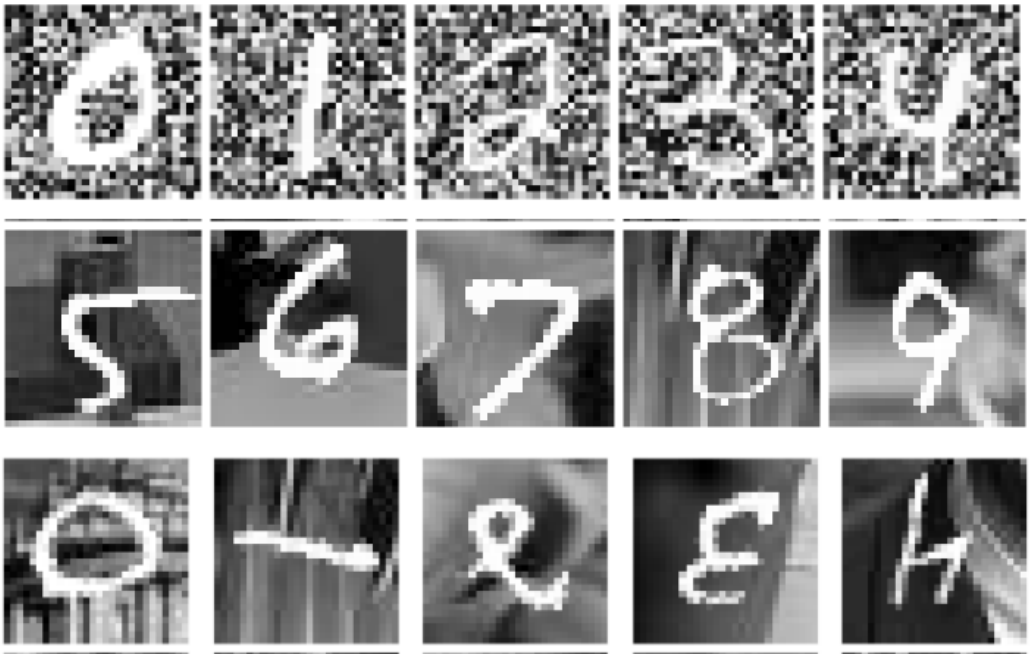

FIGURE 2.3: sample images from *mnist-back-rand*(first row), *mnist-back-image*(second row) and *mnist-rot-back-image*(third row) datasets.

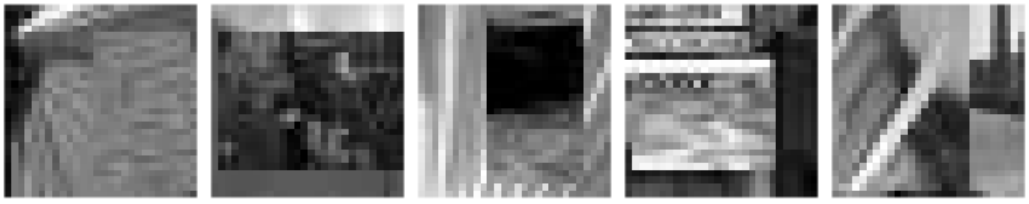

FIGURE 2.4: sample images from *rectangles-image* dataset.

2.2.6 Results and Analysis

Table 2.1 also contains best results obtained from other models([Hugo] et al.) like SVM with RBF kernel(SVM_{RBF}), SVM with polynomial kenel(SVM_{Poly}), single hidden layer feed-forward neural network(NNet), Deep Belif Networks(DBN) with 1 hidden layer(DBN-1), DBN with 3 hidden layer(DBN-3) and 3 hidden layer

Dataset	Loss in Percentage						
	SVM_{RBF}	SVM_{Poly}	NNet	DBN-3	SAA-3	DBN-1	MKMs
back-rand	14.58	16.62	20.04	**6.73**	11.28	9.80	10.55
back-image	22.61	24.01	27.41	16.31	23.00	**16.15**	21.39
rot-back-image	55.18	56.41	62.16	**47.39**	51.93	52.21	51.61
rect-image	24.04	24.05	33.20	23.69	24.05	**22.50**	23.01

TABLE 2.1: Experimental Results of MKMs with Arc-cosine Kernels

Stacked Autoassociator Network(SAA-3). The first three models comes under shallow architectures and the remaining are deep architectures. From the table, it can be observed that MKMs outperforms all the remaining models except Deep Belif Networks(DBN). Compared to DBN the architecture, parameter tuning and optimization are fairly simple in MKMs.

The change in the performance of the classifier when each adding layer to the model was also analyzed. Figures 2.5 and 2.6 shows the variations in the classifier performance when model complexity was increased by adding more layers(on *mnist-back-rand* and *mnist-back-image* datasets respectively). The results indicates that in some cases better representation can be obtained even with a less complex model (eg: in case of *mnist-back-image* dataset, where the performance degrades when model complexity increases).

2.3 Visualizing Features using tSNE

In order to visualize the features learned by the MKM, t-distributed Stochastic Neighbor Embedding(tSNE, proposed by [Maaten] et al.) was used. tSNE is a tool for visualizing high-dimensional data. It converts the similarities between datapoints to joint probabilities and tries to minimize the Kullback-Leibler divergence(KL divergence) between the joint probabilities of the low-dimensional embedding and the high dimensional data([Maaten] et al.). tSNE is a variant of stochastic neighbor embedding that is much easier to optimize and produce

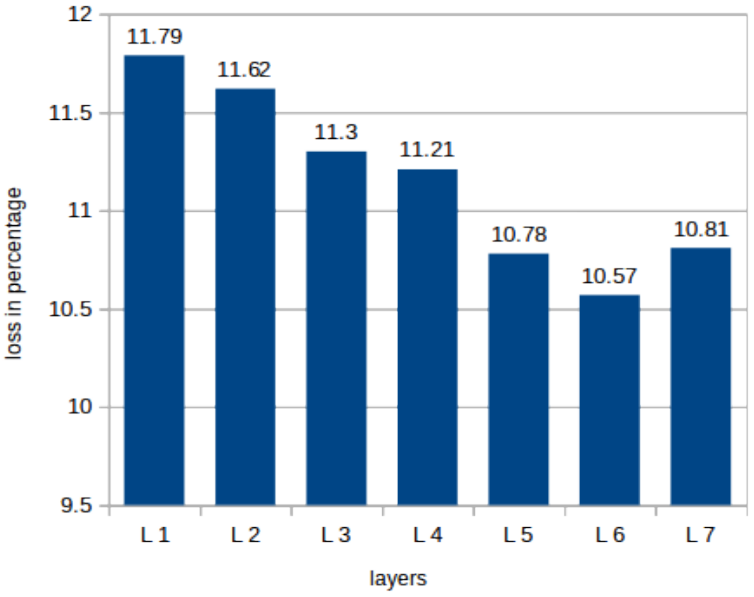

FIGURE 2.5: Change in classifier performance on *mnist-back-rand* dataset when increasing the number of layers.

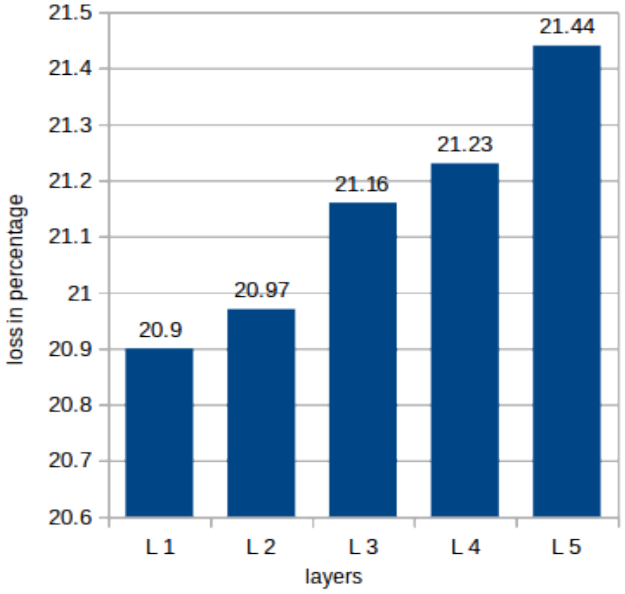

FIGURE 2.6: Change in classifier performance on *mnist-back-image* dataset when increasing the number of layers.

significantly better visualizations by reducing the tendency of the points to get accumulated at the center of the map.

Suppose $X = \{x_1, x_2, \ldots, x_n\}$ are the high-dimensional datapoints, and $Y = \{y_1, y_2, \ldots, y_n\}$ the corresponding low-dimensional map points. In SNE, we first convert high-dimensional representation of datapoints into pairwise similarities that can be interpreted as conditional probabilities. Intuitively, the similarity of a datapoint x_i to x_j is interpreted as the probability of x_i picking up x_j as its neighbor with a Gaussian centered at x_i(denoted as $p_{j/i}$). Mathematically this is formulated as

$$p_{j/i} = \frac{exp(-\left\|x_i - x_j\right\|^2 / 2\sigma_i^2}{\sum_{k \neq i} exp(-\left\|x_i - x_k\right\|^2 / 2\sigma_i^2)}$$

where σ_i is the variance of the Gaussian centered at x_i. For every x_i we calculate corresponding σ_i which will vary according to the density of datapoints around x_i(if the region around x_i is dense σ_i will be low and vice versa).

Similarly for the low-dimensional representations y_i and y_j corresponding to x_i and x_j respectively, we compute the conditional probability(denoted as $q_{j/i}$) by setting the variance of the Gaussian to $\frac{1}{\sqrt{2}}$. Thus

$$q_{j/i} = \frac{exp(-\left\|y_i - y_j\right\|^2}{\sum_{k \neq i} exp(-\left\|y_i - y_k\right\|^2)}$$

Since we are only interested in modelling pairwise similarity, we set $p_{i/i} = q_{i/i} = 0$. The crux of SNE as stated by [Maaten] et al. is "if the map points y_i and y_j correctly model the similarity between high-dimensional datapoints x_i and x_j then the conditional probabilities $p_{j/i}$ and $q_{j/i}$ will be equal". In this case, the cost function can be formulated to minimize the difference between conditional probabilities $p_{j/i}$ and $q_{j/i}$ using KL divergence. The cost function C is defined as

$$C = \sum_i \mathrm{KL}(\mathrm{P}_i \big|\big| \mathrm{Q}_i) = \sum_i \sum_j p_{j/i} log \frac{p_{j/i}}{q_{j/i}} \tag{2.3}$$

where P_i is the conditional probability distribution over all other datapoints given datapoint x_i and Q_i is the conditional probability distribution over all other map points given the map point y_i.

The minimization of the cost function in 2.3 is performed by using gradient descent method. The gradient is computed as

$$\frac{\partial C}{\partial y_i} = 2\sum_j (p_{j/i} - q_{j/i} + p_{i/j} - q_{i/j})(y_i - y_j)$$

Though SNE can produce good quality visualizations, it has the following limitations.

- Since the KL divergence is not symmetric, different types of error in the pairwise distances in low-dimensional map are not weighted equally(there is a large cost for using widely separated map points to represent nearby datapoints, but the cost is small for using nearby map points to represent widely separated datapoints).
- crowding of map points in the center of the map.

tSNE is a variant of SNE which improves the visualization quality by alleviating the above mentioned problems.

- It uses a symmetric version of the SNE cost function by employing a joint probability distribution instead of the conditional probability distribution used by SNE. This also results in simpler gradients.
- It uses student-t distribution to compute the similarity between map points. tSNE employs a heavy-tailed distribution for the map points to alleviate both the crowding problem and optimization problems of SNE.

In the symmetric version of the tSNE the cost function is computed as shown below.

$$C = \mathrm{KL}(\mathrm{P}\big|\big|\mathrm{Q}) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

where P and Q are the joint probability distribution functions in the high-dimensional and low-dimensional space respectively. Here also we set p_{ii} and q_{ii} to zero. The symmetry in the cost function is achieved due to the fact that $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$ $\forall i, j$. The joint probability in high-dimensional space(p_{ij}) and low-dimensional

space(q_{ij}) are computed as

$$p_{ij} = \frac{exp(-\left\|x_i - x_j\right\|^2 / 2\sigma^2}{\sum_{k \neq l} exp(-\left\|x_k - x_l\right\|^2 / 2\sigma^2)}$$

$$q_{ij} = \frac{exp(-\left\|y_i - y_j\right\|^2}{\sum_{k \neq l} exp(-\left\|y_k - y_l\right\|^2)}$$

The gradient of the symmetric SNE has a fairly simple form

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)$$

Symmetric SNE still faces problems when a datapoint x_i is an outlier; when x_i is an outlier $\left\|x_i - x_j\right\|^2$ is large for all x_j with x_i, hence p_{ij} are extremely low, so the location of its low-dimensional map point y_i has very little effect on the cost function. This problem is addressed in tSNE by defining $p_{ij} = \frac{p_{j/i} + p_{i/j}}{2n}$, which ensures that $\sum_j p_{ij} \geq \frac{1}{2n} \; \forall \; x_i$, hence each datapoint x_i makes a significant contribution to the cost function. The crowding problem is addressed in tSNE by using a student t-distribution with one degree of freedom as the heavy-tailed distribution in the low-dimensional map. Thus the joint probability q_{ij} is computed as

$$q_{ij} = \frac{(1 + \left\|y_i - y_j\right\|^2)^{-}1}{\sum_{k \neq l} (1 + \left\|y_k - y_l\right\|^2)^{-}1}$$

Then, finally the gradient of the cost function is obtained as

$$\frac{\partial C}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \left\|y_i - y_j\right\|^2)^{-}1)$$

tSNE was employed in our experiments to understand the changes in the data distribution before and after the feature learning process. Figure 2.7 shows the tSNE embedding of the raw data from *mnist-back-rand* dataset and figure 2.8 shows the same embedding applied on the features produced by MKM with arc-cosine kernels. The tSNE embedding of features learned by MKMs are crowded near the center, hence it was difficult to interpret the separability.

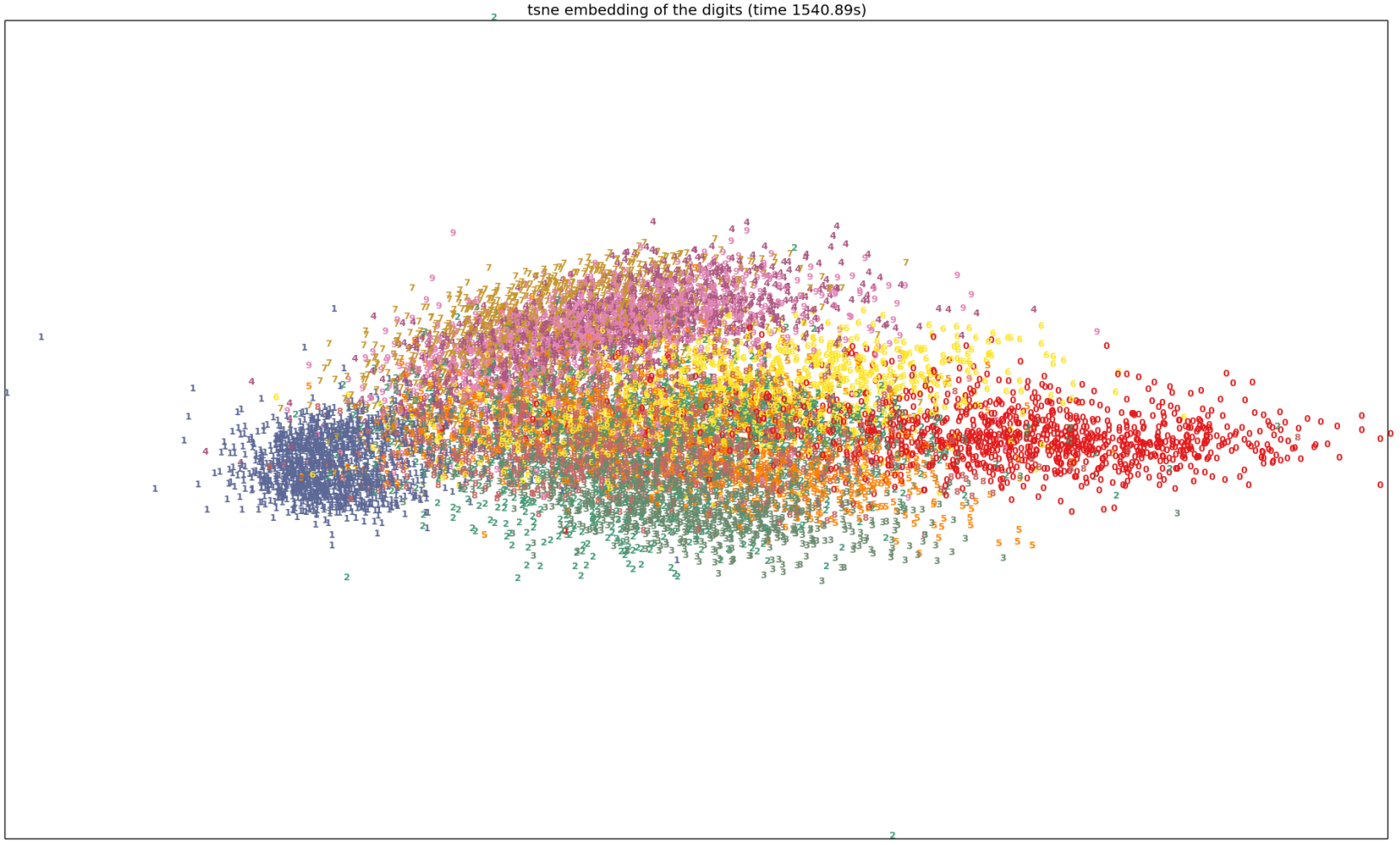

FIGURE 2.7: tSNE embedding of raw data from *mnist-back-rand* dataset.

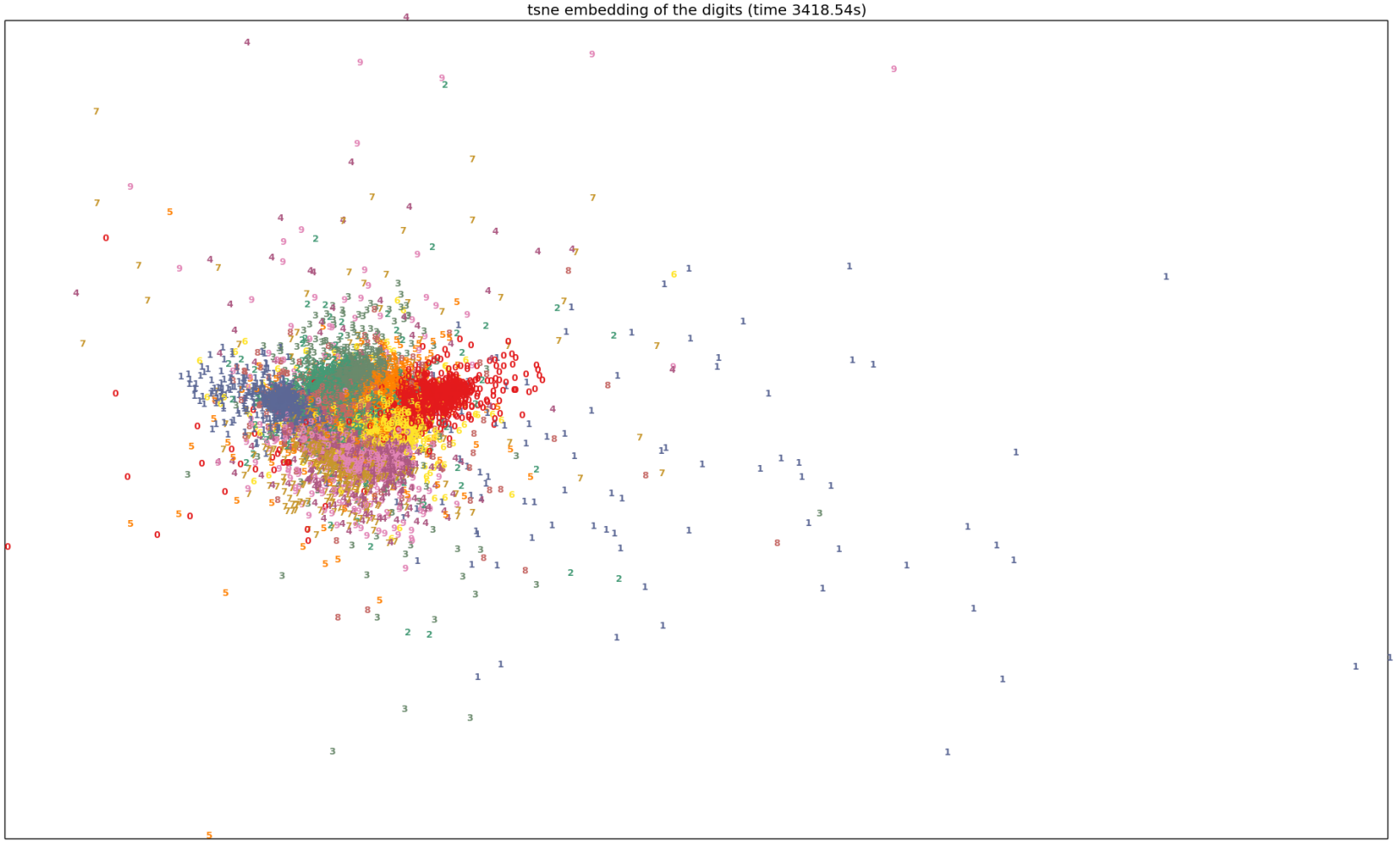

FIGURE 2.8: tSNE embedding of features obtained by MKM from *mnist-back-rand* dataset.

Dataset	Loss in Percentage						
	SVM_{RBF}	SVM_{Poly}	NNet	DBN-3	SAA-3	DBN-1	$MKMs_{(mix)}$
back-rand	14.58	16.62	20.04	**6.73**	11.28	9.80	9.54
back-image	22.61	24.01	27.41	16.31	23.00	**16.15**	20.94
rot-back-image	55.18	56.41	62.16	**47.39**	51.93	52.21	54.03
rect-image	24.04	24.05	33.20	23.69	24.05	**22.50**	25.04

TABLE 2.2: Experimental Results of MKMs with Mixed Kernels

2.4 MKMs with Mixed Kernels

We tried different kernel functions in different layers of MKMs (Gaussian, arc-cosine, polynomial etc.) while performing KPCA. It had been seen that mixing a layer of Gaussian or polynomial kernel in between successive layers of arc-cosine kernel improves the result. This further supported the belief that, with more similarity information (by using more kernels) in hand, we can build better representations. Table 2.2 shows the result obtained with the mixed kernel models. Because of the improved performance, we used Multi-layer Multiple Kernel Machines(ML-MKL) model for our analysis, whose discussion is given in the next chapter.

As in MKMs with arc-cosine kernel, we chose top 5 percent features based on univariate test score for the mixed kernel MKMs. The classifier used in the output layer was an SVM with arc-cosine kernel. The results indicates that MKMs with mixed kernels improved the classifier performance for *mnist-back-rand* and *mnist-back-image* datasets, whereas the classification accuracy declined in the case of *mnist-rot-back-image* and *rectangles-image* dataset. The tSNE embedding of the features produced by MKMs with mixed kernels is shown in figure 2.9 (for *mnist-back-rand* dataset). The best result in this dataset was obtained from a model consisting of three layers with a Gaussian kernel in the middle layer. The visualization indicates that, MKMs with mixed kernels has good separability between

different classes.

2.5 Conclusion

In this chapter we analysed the MKMs framework which has the characteristics of deep learning algorithms by employing kernel methods. The empirical study indicated that MKMs are performing comparable with that of popular deep learning algorithms like DBN, SAA etc. We also analysed the performnace of MKMs with mixed kernels. The mixing produces better results when other kernels are sandwiched between two arc-cosine kernels.

The visualization of features learned by MKMs are difficult to interpret due to crowding of datapoints near the center, but for MKMs with mixed kernels the visualization indicates that the separation is good enough between different classes, even though the samples from same class is distributed in different locations.

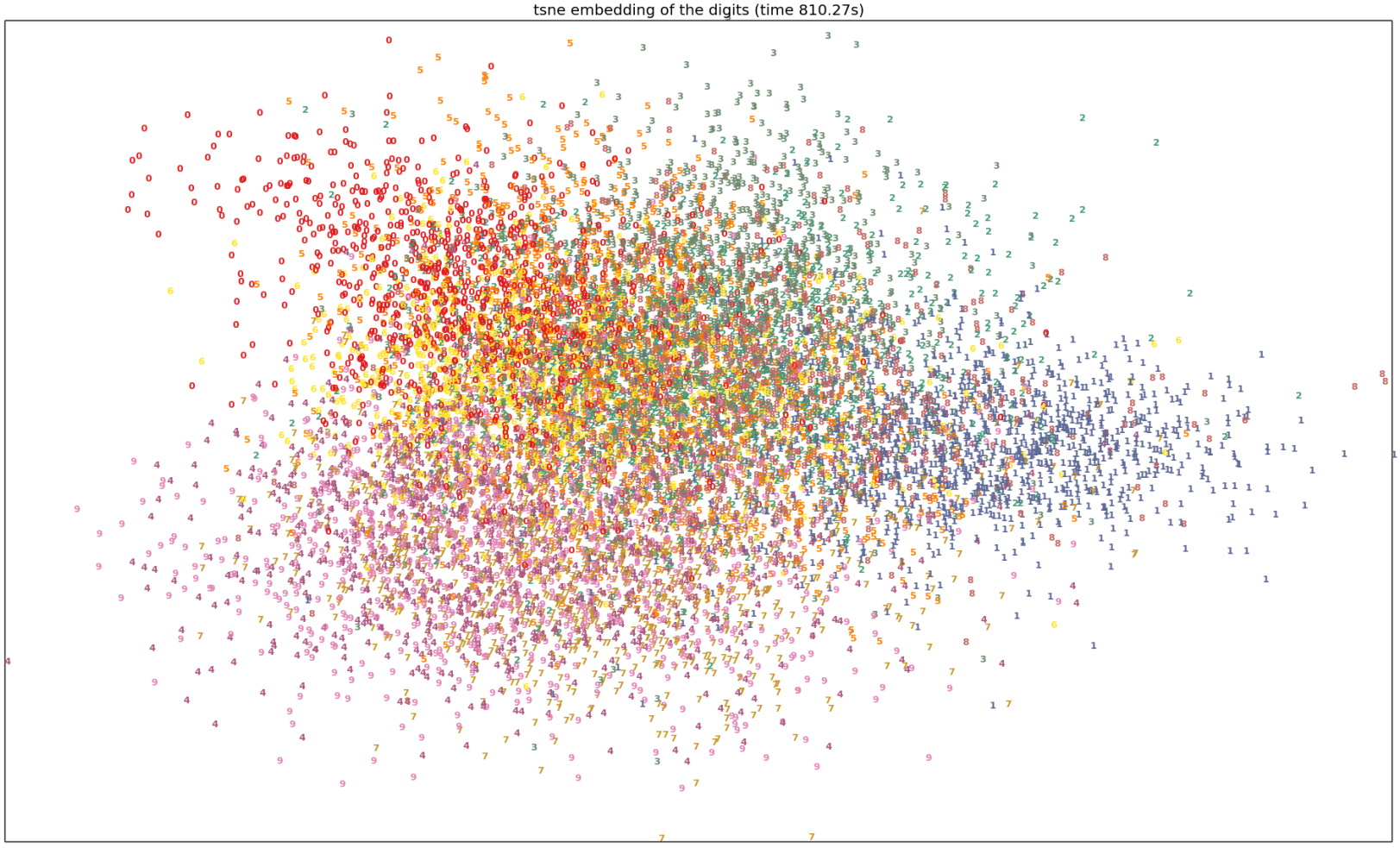

FIGURE 2.9: tSNE embedding of features obtained by MKM with mixed kernels from *mnist-back-rand* dataset.

Chapter 3

Multi-layer Multiple Kernel Learning

In this chapter we discuss the Multi-layer Multiple Kernel Learning (ML-MKL) framework developed using an unsupervised MKL algorithm. The organization of this chapter is as follows; section 3.1 gives the discussion of traditional MKL algorithm in a supervised learning settings, in section 3.2 the unsupervised MKL formulation is introduced, in section 3.3 the related works in ML-MKL domain is discussed, in section 3.4 the proposed ML-MKL algorithm is discussed along with the experimental results and section 3.5 concludes this chapter.

3.1 Multiple Kernel Learning

Multiple Kernel Learning(MKL) aims at learning a convex combination of a set of predefined base kernels for choosing an optimum kernel([Gert] et al.). The primary aim of MKL algorithm is to automate the process of choosing the optimum kernel for the learning task.

Typically multiple kernel learning is formulated in a supervised learning settings. Suppose we are given n datapoints $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$ is the i^{th} input data vector and y_i is the label of the class to which x_i belongs to.

Then the MKL learning task is formulated as the following optimization problem.

$$\min_{k\in\mathcal{K}} \min_{f\in\mathcal{H}_k} \lambda \|f\|^2_{\mathcal{H}_k} + \sum_{i=1}^{n} l(y_i, f(x_i)) \tag{3.1}$$

where $l(\cdot)$ denotes the loss function(commonly used loss function is the hinge loss, defined as $l(t) = \max(0, 1-t)$), $\mathcal{H}_k$ is the RKHS corresponding to the kernel k, $\mathcal{K}$ denotes the optimization domain of the candidate kernels, and $\lambda > 0$ is the regularization parameter.

The optimizaion domain $\mathcal{K}$ is the convex combination of a set of predefined base kernels, defined as follows

$$\mathcal{K} = \left\{ k(\cdot\ ,\cdot) = \sum_{t=1}^{m} \mu_t k_t(\cdot\ ,\cdot) :\ \sum_{t=1}^{m} \mu_t = 1, \mu_t \geq 0 \right\}$$

where k_t is the t^{th} base kernel and μ_t is the weight associated to the t^{th} base kernel. The decision function $f(x)$ can be computed as a linear combination of kernel evaluations on all training samples;

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x)$$

where α_i's are the coefficients. As per the definition of kernel k in MKL, the decision function of the conventional MKL is expressed as

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha_i \sum_{t=1}^{m} \mu_t k_t(x_i, x) \\ &= \sum_{i=1}^{n} \sum_{t=1}^{m} \alpha_i \mu_t k_t(x_i, x) \end{aligned}$$

In this basic formulation, the optimization task has to identify both the optimal kernel k from the domain $\mathcal{K}$ and the optimal decision function f from the RKHS $\mathcal{H}_k$ simultaneously. In order to eliminate this problem [Corinna] et al. proposed a two stage kernel learning algorithm which separates the kernel learning task from the decision function learning.

3.2 Unsupervised MKL

In the KPCA based feature extraction stages, we were using only one kernel for the task. With the intuition that by using multiple kernels at each layer we would get more similarity information, we computed a convex combination of multiple kernels following the work in [Zhuang, 2011] et al. Since we are using kernels for unsupervised feature extraction, traditional MKLs following supervised paradigm cannot be used here.

The goal of an unsupervised multiple kernel learning task is to find an optimal linear combination of the m kernel functions as, i.e, $k^*(\cdot\ ,\cdot) \in \mathcal{K}$. In order to determine the optimality of a linear combination of kernels, we used the following quality criteria[Zhuang, 2011] et al.:

- A good kernel should enable each training instances to be well reconstructed from the localized bases weighted by the kernel values. Formulating this requirment mathematically, for each x_i we expect the optimal kernel should minimize the approximation error $\left\| x_i - \sum_j k_{ij} x_j \right\|^2$, where $k_{ij} = k(x_i, x_j)$.
- A good kernel should induce kernel values that are coincided with the local geometry of the training data. This is equivalent to finding the optimal kernel that minimizes the distortion over all trainig data, computed as $\sum_{i,j} k_{ij} \left\| x_i - x_j \right\|^2$.

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, the optimization problem of unsupervised MKL can be formulated as follows.

$$\min_{k \in \mathcal{K}} \frac{1}{2} \sum_{i=1}^{n} \left\| x_i - \sum_{x_j \in B_i} k_{ij} x_j \right\|^2 + \gamma * \sum_{i=1}^{n} \sum_{x_j \in B_i} k_{ij} \left\| x_i - x_j \right\|^2$$

where γ is a tuning parameter, which controls the tradeoff between the coding error and the locality distortion. Converting to matrix notations the above problem

becomes

$$\min_{\mu\in\Delta,D} \frac{1}{2} \|X(I - K \circ D)\|_F^2 + \gamma * \text{ tr } K \circ D \circ M(11^T) \tag{3.2}$$

$$\text{subject to } D \in \{0,1\}^{n\times n}$$

$$\|d_i\|_1 = N_B, i = 1, 2, \ldots, n$$

$$\Delta = \left\{\mu : \mu^T \mathbf{1} = 1, \mu \geq 0\right\} \text{ and}$$

$$[K]_{i,j} = \sum_{t=1}^{m} \mu_t k^t(x_i, x_j), 1 \leq i, j \leq n$$

The matrix $D \in \{0,1\}^{n\times n}$ contains information about local bases of each x_i as a column vector. In particular, each column vector $d_i \in \{0,1\}^n$ in D has a 1 at those points j where, $x_j \in B_i$ and zero elsewhere(or $B_i = \{x_j : d_j \neq 0\}$). The matrix M is defined as

$$[M]_{ij} = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$$

In equation 3.2 the notation '$\circ$' denotes elementwise multiplication of two matrices, $\|\cdot\|_F^2$ denotes the Frobenius norm of a matrix and 'tr' denotes the trace of a matrix. In their implementation([Zhuang, 2011]) Zhuang et.al solved the optimization problem in two stages in an alternating fashion, 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 we are using a many layer architecture, the alternating optimization strategy is too costly; so we chose to do the optimization across μ only by choosing D beforehand. Specifically, the matrix D is computed beforehand by taking k nearest neighbours of x_i from the training set and putting a one in those positions for d_i. Rest of the positions are filled with zeros. The resulting optimization problem will be

$$\min_{\mu\in\Delta} \frac{1}{2} \|X(I - K \circ D)\|_F^2 + \gamma * \text{ tr } K \circ D \circ M(11^T) \tag{3.3}$$

$$\text{subject to } D \in \{0,1\}^{n\times n}, \|d_i\|_1 = N_B, i = 1, 2, \ldots, n$$

$$\Delta = \left\{\mu : \mu^T \mathbf{1} = 1, \mu \geq 0\right\}$$

The objective function can be formulated as a convex quadratic programming problem w.r.t to kernel weights μ as shown below(derivation of the objective function $J(\mu)$ is shown in Appendix A).

$$J(\mu) = \mu^T \left(\sum_{t=1}^{m} \sum_{i=1}^{n} k_{t,i} k_{t,i}^T \circ d_i d_i^T \circ P \right)^T \mu + z^T \mu \tag{3.4}$$

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 $k_{t,i} = \Big[k^t(x_i, x_1), \ldots, k^t(x_i, x_n)\Big]^T$ is the i^{th} column of the t^{th} kernel matrix. p and v are columns of P and M corresponding to x_i respectively.

3.3 Related Works

[Zhuang] et al. explored the idea of ML-MKL, in which at each layer they took a non-linear combination of kernels in the previous layer, forming a layered structure. They called such a model as deep multiple kernel learning framework. The architecture of their model is shown in figure 3.1. The kernel function obtained at layer l is having the following domain.

$$\mathcal{K}^{(l)} = \left\{ k^{(l)}(\cdot,\cdot) = g^{(l)}\Big(\big[k_1^{(l-1)}, \ldots, k_m^{(l)}(\cdot,\cdot)\big]\Big) \right\}$$

where $g^{(l)}$ is a function to combine multiple $(l-1)$ level kernels, which must ensure the resulting combination is a valid kernel. Though this combination can be applied for any number of layers, optimization problem is difficult to solve beyond two layers. Hence [Zhuang] et al. considered only 2-layer MKL in their empirical study.

In the 2-layer MKL, [Zhuang] et al. defined the kernel domain by using an RBF kernel for the combination function.

$$\mathcal{K}^{(2)} = \left\{ k^{(2)}(x_i, x_j; \boldsymbol{\mu}) = exp\Big(\sum_{t=1}^{m} \mu_t k_t^{(1)}(x_i, x_j) \Big) \right\} : \boldsymbol{\mu} \in \mathbb{R}_+^m$$

μ_t is the weight of the $t^{(th)}$ antecedent layer kernel $k_t^{(1)}$(superscript (1) is used here because the antecedent layer is layer 1 in a 2-layer architecture). In order to

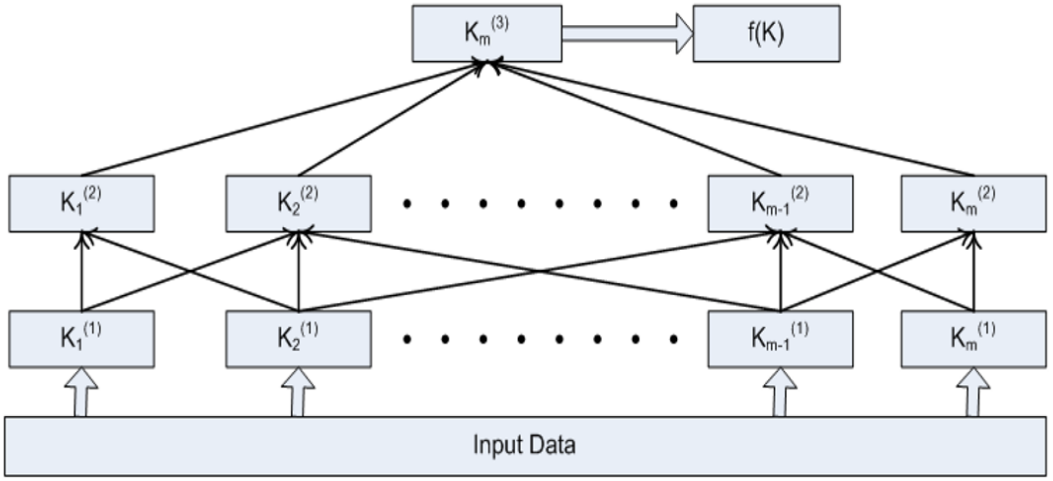

FIGURE 3.1: Architecture of ML-MKL model proposed by [Zhuang] et al.

prevent the kernel weight being too large, the kernel weights are also introduced into the optimization objective as a regularization term. Then the optimization objective becomes

$$\min_{k\in\mathcal{K}} \min_{f\in\mathcal{H}_k} \|f\|_{\mathcal{H}_k}^2 + \mathcal{C}\sum_{i=1}^{n} l(y_i, f(x_i)) + \sum_{t=1}^{m} \mu_t \tag{3.5}$$

solving the Lagrangian, we will get the dual objective function as

$$\min_{\boldsymbol{\mu}} \max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j k^{(2)}(x_i, x_j; \boldsymbol{\mu}) + \sum_{t=1}^{m} \mu_t$$

$$\text{s.t } 0 \le \alpha_i \le \mathcal{C}, \sum_{i=1}^{n} \alpha_i y_i = 0, \mu_t \ge 0, t = 1, \ldots, m$$

where $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$ is a vector of dual variables and $\mu = [\mu_1, \mu_2, \ldots, \mu_m]^T$. The final decision function of the 2-layer MKL is given as

$$f(x; \boldsymbol{\alpha}, \boldsymbol{\mu}) = \sum_{i=1}^{n} \alpha_i y_i k^{(2)}(x_i, x; \boldsymbol{\mu}) + b$$

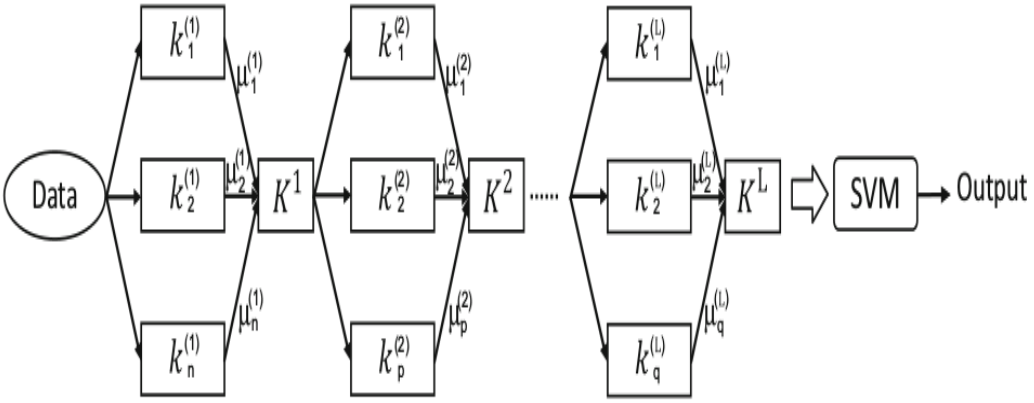

FIGURE 3.2: Architecture of ML-MKL model proposed by [Ilyes] et al. The kernels in each layer are linearly combined and the resulting Gram matrix is passed to the next layer as input.

where b is the bias term. Rewriting the optimization objective in terms of α and μ, we have

$$\mathcal{J}(\alpha, \mu) = \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k^{(2)}(x_i, x_j; \boldsymbol{\mu}) - \sum_{i=1}^{n} \alpha_i - \sum_{t=1}^{m} \mu_t$$

The optimization problem is solved in two stages

- by fixing μ and solve for α.
- by fixing α and solve for μ.

Since $k^{(2)}$ is positive definite $\mathcal{J}(\alpha, \mu)$ is convex over α, thus an SVM solver can be used to solve the optimization over α. The optimization over μ is solved using gradient ascent. In order to address the challenge of choosing the optimal set of base kernels, they proposed to choose base kernels iteratively inside $k^{(2)}$.
[Ilyes] et al. used backpropagation algorithm for designing the ML-MKL framework. The architecture of the model studied by them is shown in figure 3.2. They defined the kernel domain at level l as

$$\mathcal{K}^{(l)} = \left\{ K^{(l)}(K^{(l-1)}; \mu^{(l)}) = \sum_{t=1}^{m} \mu_t^{(l)} k_t^{(l)}(K^{(l-1)}) \right\}$$

$$\text{s.t } \mu_t^{(l)} \geq 0, l = 1, \ldots, L$$

where $k_t^{(l)}$ is the k^{th} base kernel at layer l and $\mu_t^{(l)}$ denotes the weight of the k^{th} base kernel at layer l. L is the total number of layers. The feature extracted in the antecedent layer are combined linearly and passed to the next layer as input. The kernel weights are obtained by minimizing the mean squre error(denotes as $\mathcal{E}$) of the predicted outputs.

$$\mathcal{E} = \frac{1}{2n} \sum_{i=1}^{n} \|f(x_i) - y_i\|^2$$

where $f(x_i)$ is the value of the decision function for the input x_i, which is computed as

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K^{(l)}(K^{(l-1)}; \mu^{(l)}) + b$$

where b is the bias term. The weights $\mu^{(l)}$ in each layer are obtained using gradient descent algoritm.

$$\mu^{(l)} := \mu^{(l)} + \eta \nabla \mathcal{E} \; l = 1, \ldots, L$$

$$\nabla \mathcal{E} = \left\{ \frac{\partial \mathcal{E}}{\partial \mu_1^{(1)}}, \cdots, \frac{\partial \mathcal{E}}{\partial \mu_m^{(1)}}; \cdots; \frac{\partial \mathcal{E}}{\partial \mu_1^{(L)}}, \cdots, \frac{\partial \mathcal{E}}{\partial \mu_m^{(L)}} \right\}$$

Here η is the learning rate parameter. The error obtained in the final layer are propagated back to all layers using backpropagation algorithm to update the kernel weight parameters $\mu^{(l)}$ in each layer.

3.4 Multi-layer Multiple Kernel Learning

The architectue of the proposed ML-MKL framework is shown in figure 3.3. It consists of many layers and in each layer the kernel PCA based feature extraction is performed using the combination of a set of predefined kernels. The dimensionality of the features thus obtained are reduced by using supervised feature selection techniques. The final output can be given to any classifier. Algorithm 1 summarizes the proposed ML-MKL algorithm.

The experimental setup used here was the same as described in chapter 2. Table 3.1 lists the results obtained with the proposed ML-MKL framework. The classifier used was SVM with arc-cosine kernel. For *mnist-back-rand* dataset, the best result

Algorithm 1: ML-MKL Algorithm

Input: data X, true labels y, no. of layers L, base kernels for each layer $K_{base}^{(l)} = \{k_1^{(l)}, k_2^{(l)}, k_m^{(l)}\}$, N_B, γ;

Output: kernel weights μ^l for each layer, predicted labels;

1.Initialize $[M]_{ij} = x_i^T x_i + x_j^T x_j - 2x_i^T x_j$, $\boldsymbol{D} = d_1, d_2, \ldots, d_n$ as row vectors, where $d_i = \{1$ if $x_j \in B_i$ else $0 \forall x_j \in X\}$, $\mu = \frac{1}{m}$, $\boldsymbol{P} = X^T X$;

2.**for** *each layer* l **do**

a. $\boldsymbol{W} = \sum_{t=1}^{m} \sum_{i=1}^{n} k_{t,i}^{(l)} k_{t,i}^{(l)^T} \circ d_i d_i^T \circ P$

b. $[\boldsymbol{z}]_t^l = \sum_{i=1}^{n} (2\gamma v_i \circ d_i - 2p_i \circ d_i)^T k_{t,i}^{(l)}$

c. $\boldsymbol{\mu}^{*^l} = \mu^{l^T} W \mu^l + z^{l^T} \mu^l$

d. $\boldsymbol{K}_{new} = \sum_{t=1}^{m} \mu_t^l * K_t^{(l)}$

e. extract principal components with $\boldsymbol{K}_{new}$

f. select most informative features for layer $l(X_{new})$

g. $\boldsymbol{P} = X_{new}^T X_{new}$

end

3.Give the final set of features to any classifier;

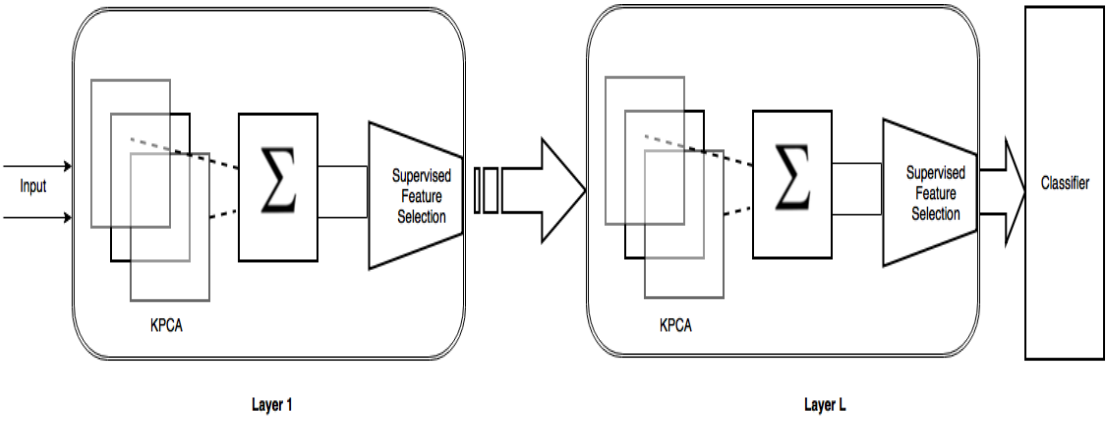

FIGURE 3.3: An ML-MKL with L layers of transformations. Each layer consists of many kernels for feature extraction using kernel PCA and a supervised feature selection module.

was obtained with a model consists of 4 layers and in each layer 7 kernels were used. In particular, each layer consisted of a mixture of one arc-cosine kernel and 6 gaussian kernels. For *mnist-back-image* dataset, the best ML-MKL model obtained had 2 layers and 5 kernels in each layer. In each layer a mixture of one arc-cosine kernel and 4 polynomial kernels were used. In the case of *mnist-rot-back-image* dataset the best result was fetched by a model having only one layer with 4 arc-cosine kernels in it. For *rectangles-image* dataset the best result was shown by a one layer ML-MKL model with one arc-cosine kernel and 6 Gaussian kernels.

Dataset	Loss in Percentage						
	SVM_{RBF}	SVM_{Poly}	NNet	DBN-3	SAA-3	DBN-1	**ML-MKL**
back-rand	14.58	16.62	20.04	**6.73**	11.28	9.80	8.43±0.088
back-image	22.61	24.01	27.41	16.31	23.00	**16.15**	20.92±0.092
rot-back-image	55.18	56.41	62.16	**47.39**	51.93	52.21	51.21±0.811
rect-image	24.04	24.05	33.20	23.69	24.05	**22.50**	22.88±0.124

TABLE 3.1: Experimental Results of ML-MKL.

Figures 3.4 and 3.5 illustrates the variation in classifier performance on *mnist-back-rand* and *mnist-back-image* datasets respectively when layers were added iteratively to the ML-MKL model. The value shown for each layer was the best error rate obtained after tuning the kernel parameters. The parameters were chosen greedily for each layer (with the expectation that subsequent layers would learn more valuable features from the current one) and no fine-tuning was performed with respect to the entire architecture.

Figure 3.6 shows the tSNE embedding of the features learned by ML-MKL algorithm. The visualization indicates that classes are well separated.

Tables 3.2 and 3.3 shows the kernel weights of each kernel in the mixture at every layer for *mnist-back-rand* and *mnist-back-image* datasets respectively. In both cases k_1 is an acr-cosine kernel, and the remaining are Gaussian kernels for *mnist-back-rand* dataset and polynomial kernel for textitmnist-back-image dataset. The results in the table indicates that, the contribution of individual kernels was highly varying in each layer (no single kernel had complete dominance over all layers in the feature learning process).

In order to evaluate the contribution of each kernel, exploratory analysis was carried out to monitor the performance of individual kernels in each layer. The individual kernels performance were compared with the combined kernel's performance. Tables 3.4 and 3.5 summarizes the results of this exploratory analysis on *mnist-back-rand* and *mnist-back-image* datasets respectively (here K_{conv} is the

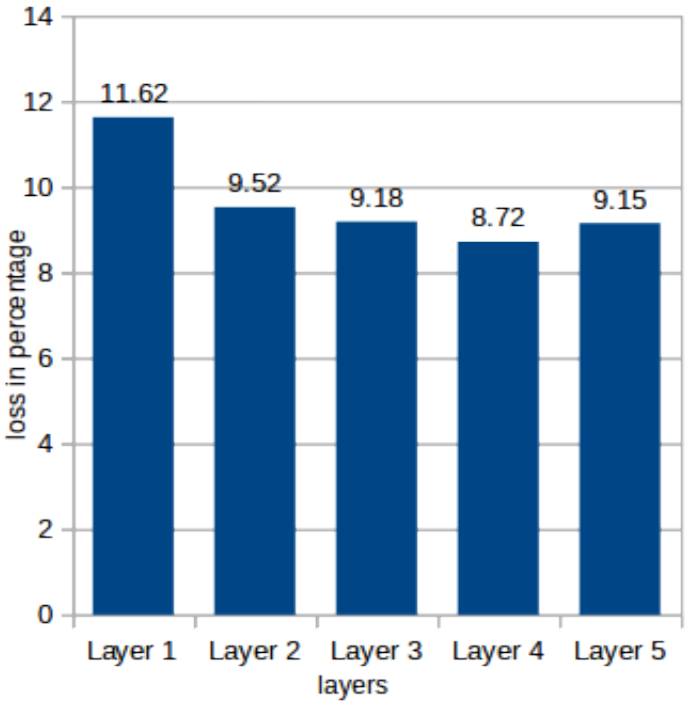

FIGURE 3.4: Change in classifier performance on *mnist-back-rand* dataset when adding layers iteratively.

Layers	**Kernel Weights**						
	k_1	k_2	k_3	k_4	k_5	k_6	k_7
Layer 1	0.2007	0.1331	0.1331	0.1332	0.1332	0.1333	0.1333
Layer 2	0.2711	0.1160	0.1181	0.1203	0.1225	0.1248	0.1271
Layer 3	0.1764	0.0999	0.1125	0.1266	0.1426	0.1607	0.1811
Layer 4	0.0598	0.0524	0.0747	0.1071	0.1547	0.2245	0.3269
Layer 5	0.0514	0.0493	0.0726	0.1069	0.1569	0.2295	0.3334

TABLE 3.2: Kernel weights in each layer for the *mnist-back-rand* dataset

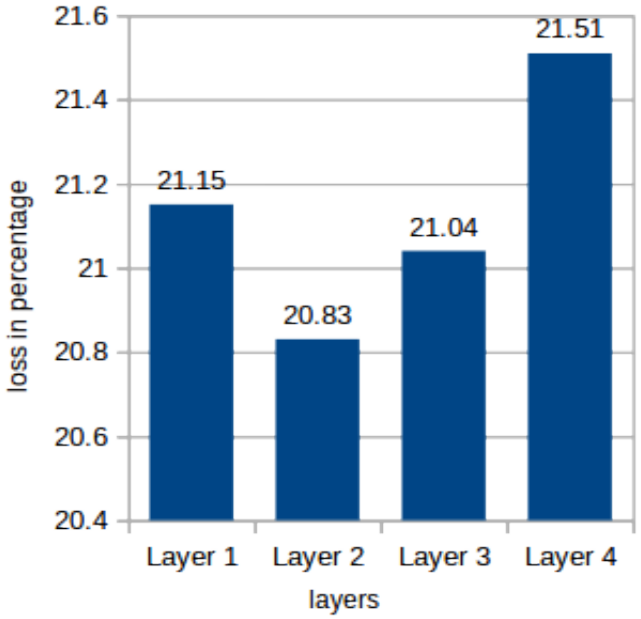

FIGURE 3.5: Change in classifier performance on *mnist-back-image* dataset when adding layers iteratively.

Layers	**Kernel Weights**				
	k_1	k_2	k_3	k_4	k_5
Layer 1	0.2445	0.1887	0.1888	0.1889	0.1891
Layer 2	0.3421	0.1603	0.1630	0.1659	0.1688
Layer 3	0.2035	0.1615	0.1843	0.2104	0.2403
Layer 4	0.0843	0.1409	0.1877	0.2508	0.3362

TABLE 3.3: Kernel weights in each layer for the *mnist-back-image* dataset

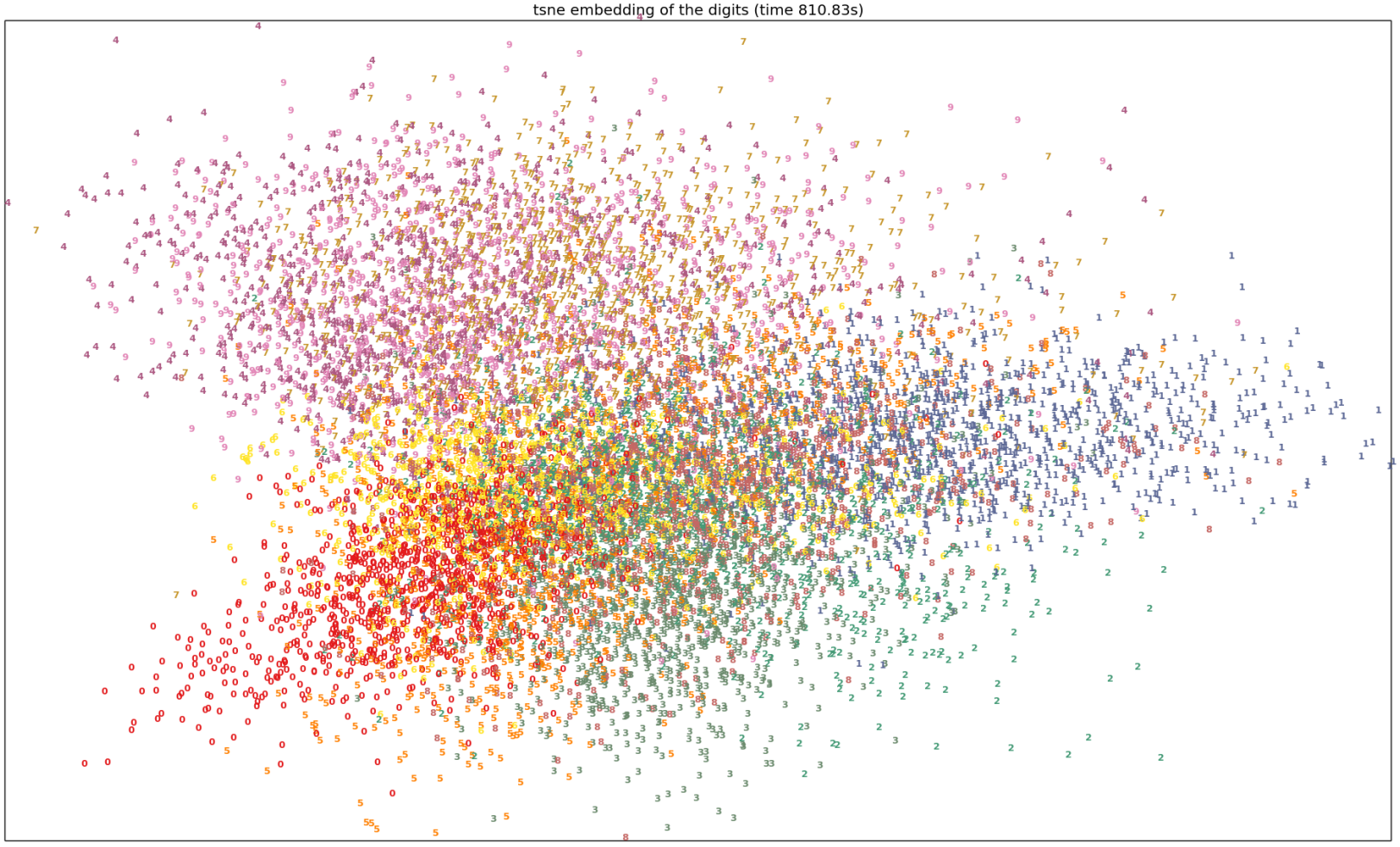

FIGURE 3.6: tSNE embedding of features obtained by ML-MKL algorithm for the *mnist-back-rand* dataset.

Layers	**Loss in Percentage**							
	k_1	k_2	k_3	k_4	k_5	k_6	k_7	K_{conv}
Layer 1	11.46	9.92	10.06	10.04	10.1	9.87	10.06	11.62
Layer 2	10.36	9.87	9.90	9.89	9.92	9.82	9.81	9.52
Layer 3	9.46	11.07	10.33	9.85	9.70	9.56	9.01	9.18
Layer 4	9.26	9.06	8.95	8.96	8.93	8.72	9.00	**8.72**
Layer 5	9.18	9.07	9.02	9.20	9.22	9.36	9.47	9.15

TABLE 3.4: Individual kernels performance evaluation for *mnist-back-rand* dataset

Layers	**Loss in Percentage**					
	k_1	k_2	k_3	k_4	k_5	K_{conv}
Layer 1	20.91	21.82	21.82	21.83	21.77	21.15
Layer 2	21.27	21.00	21.09	21.04	21.04	**20.83**
Layer 3	20.95	21.18	21.02	20.99	21.05	21.03
Layer 4	21.06	21.36	21.42	21.64	21.66	21.51

TABLE 3.5: Individual kernels performance evaluation for *mnist-back-image* dataset

result of combined kernel). The results clearly indicates that the combination was not always improving the performance in all layers. However, the best result in both datasets were obtained by a combined kernel.

3.5 Conclusion

In this chapter we explored the concept of multiple kernel learning in MKMs. A linear combination of multiple kernels formulated purely from unlabelled data is used in each layer of MKMs. The learning process of the proposed ML-MKL

algorithm employs a greedy layerwise training for each layer. Empirical results indicates that using (unsupervised) MKL in MKMs improves the classifier performance. In our experimental analysis the classification accuracy of ML-MKL models was better than learning machines with shallow architectures and was comparable with existing deep architectures.

Exploratory analysis performed on the features learned by the ML-MKL model reveals much more interesting facts about the model. The contribution of kernels in each layer is measured in terms of individual kernel performance and the weights assigned to that kernel. However this information is insufficient to determine the optimal structural complexity required for modelling a problem.

Chapter 4

Kernel FDA with Multi-layer Kernels

In this chapter we study the discriminating power of multi-layer kernels with kernel Fisher Discriminant Analysis(KFDA). The analysis in this section was done on binary classification problems. This chapter is organized as follows: section 4.1 gives a brief introduction of kernel Fisher discriminant Analysis, section 4.2 contains the results of empirical study on *rectangles-image* and *convex* datasets(both are binary classification problems studied extensively in deep learning literatures), and section 4.3 gives the conclusion.

4.1 Kernel Fisher Discriminant Analysis

The working principle of discriminant analysis is to find a set of features that discriminates the classes very well([Sebastian] et al.). Fisher Discriminant Analysis(FDA) was originally proposed for learning a set of discriminating features in the input space. Kernel FDA is a non-linear generalization of FDA, in which the discriminating features are learned in feature space.

Let $X_1 = \{x_1^1, \ldots, x_{n_1}^1\}$ and $X_2 = \{x_1^2, \ldots, x_{n_1}^2\}$ be data samples from two classes (class 1 and class 2) and the union of two, denoted as $X = X_1 \cup X_2$ as the training set. KFDA find the directions f which maximizes the cost function

$$\mathcal{J}(f) = \frac{f^T S_B^\phi f}{f^T S_W^\phi f} \tag{4.1}$$

where $f \in \mathcal{F}$ and S_B^ϕ and S_W^ϕ are the between and within class scatter matrices respectively

$$S_B^\phi = (m_1^\phi - m_2^\phi)(m_1^\phi - m_2^\phi)^T$$

$$S_W^\phi = \sum_{i=1,2} \sum_{x \in X_i} (\phi(x) - m_i^\phi)(\phi(x) - m_i^\phi)^T$$

where $m_i^\phi = \frac{1}{n_i} \sum_{j=1}^{n_i} \phi(x_j^i)$. Intuitively maximizing $\mathcal{J}(f)$ is equivalent to finding a direction w which maximizes the separation of the two classes while minimizing the within class variance([Sebastian] et al.). We need to transform the formulation in 4.1 in terms of kernel function $k(x, y) = \phi(x) \cdot \phi(y)$ in order to use kernels. According to RKHS theory, any solution to the Tikhnov regularization $f \in \mathcal{F}$ must lie in the span of the feature map($\phi(\cdot)$) corresponding to training examples. Thus it can be represented as

$$f = \sum_{i=1}^{n} \alpha_i \phi(x_i) \tag{4.2}$$

combining 4.2 and the definition of m_i^ϕ we have

$$f^T m_i^\phi = \frac{1}{n_i} \sum_{j=1}^{n} \sum_{k=1}^{n_i} \alpha_j k(x_j, x_k^i) = \alpha^T M_i$$

where $(M_i)_j = \frac{1}{n_i} \sum_{k=1}^{n_i} k(x_j, x_k^i)$. Define $M = (M_1 - M_2)(M_1 - M_2)^T$. The we have

$$f^T S_B^\phi f = \alpha^T M \alpha \tag{4.3}$$

using similar transformations we have

$$f^T S_W^\phi f = \alpha^T N \alpha \tag{4.4}$$

where $N = \sum_{i=1,2} K_i (I - \mathbf{1}_{n_i}) K_i^T$, K_i is an $n \times n_i$ matrix with entries $(K_i)_{nm} = k(x_n, x_m^i)$(this is the kernel matrix for class i), I is the identity matrix and $\mathbf{1}_{n_i}$ is the matrix with with all entries $\frac{1}{n_i}$. The derivation of this compact forms M and

N are shown in Appendix B.

Combining (4.3) and (4.4) we will get an objective function in terms of α.

$$\mathcal{J}(\alpha) = \frac{\alpha^T M \alpha}{\alpha^T N \alpha}$$

This problem can be solved by finding the leading eigen vectors of $N^{-1}M$. The projection of a new pattern x onto f is given by

$$f \cdot \phi(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x)$$

The estimation of $N \in \mathbb{R}^{n \times n}$ from a sample of size n poses an ill-posed problem(since the sample size is not high enough to get an exact covariance structure in $\mathbb{R}^{n \times n}$). This problem is solved by replacing N with N_μ as

$$N_\mu = N + \mu I$$

where μ is a large positive constant and I is the identity matrix. This has two possible benefits

- It makes the problem numerically more stable as for large μ, N_μ will become positive definite.
- It decreases the bias in sample based estimation of eigenvalues.

4.2 Experiments

Empirical study was conducted on two binary classification datasets namely *rectangles-image* dataset and *convex* dataset. A short description about *rectangles-image* dataset is given in chapter 2.

4.2.1 Convex Dataset

The *convex* dataset consists of a single convex region in an image. The dataset was constructed by taking the intersection of a number of half-planes whose location

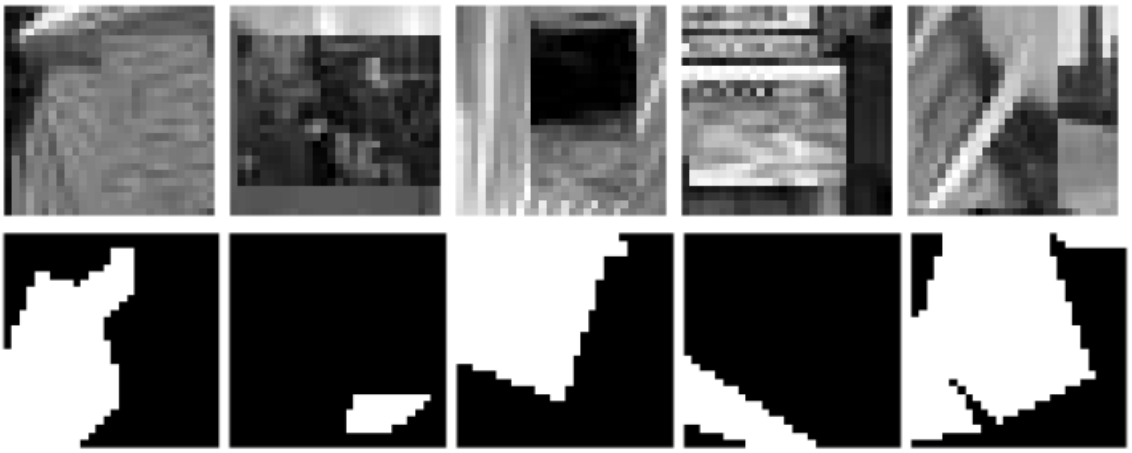

FIGURE 4.1: Sample images from *rectangles-image*(first row) and *convex*(second row) datasets.

Dataset	**Loss in Percentage**						
	SVM_{RBF}	SVM_{Poly}	NNet	DBN-3	SAA-3	DBN-1	**KFDA**
rect-image	24.04	24.05	33.20	23.69	24.05	22.50	**21.96**
convex	19.13	19.82	32.25	19.92	**18.41**	18.63	19.02

TABLE 4.1: Experimental Results of KFDA with multi-layer kernels.

and orientation were chosen uniformly at random. The classification task was to identify whether the shape enclosed in the image is convex or not. This dataset consists of 12000 training and 50000 testing samples of size 28×28.

Figure 4.1 shows some sample images from *rectangles-image* and *convex* datasets. In the experiments, KFDA with multi-layer arc-cosine kernels were used for feature extraction and kNN classifier was used for the classification. Table 4.1 shows the results of the empirical study.

For *rectangles-image* dataset, the best result was obtained for a five layer KFDA with kernel degree values in each layer was given by [0,3,3,3,3]. For *convex* dataset the best result was obtained from a model having 20 layers with degree parameter equal to 1 in each layer. The variations in classifier performance as the number of layers were increased is shown in tables 4.2 and 4.3 for *rectangles-image* and *convex* datasets respectively. In table 4.3, $1 \times n$ indicates that an arc-cosine kernel

Kernel Parameters	**Loss in Percentage**
0	23.12
0,3	22.54
0,3,3	22.39
0,3,3,3	22.15
0,3,3,3,3	21.96
0,3,3,3,3,3	22.01

TABLE 4.2: Change in classifier performance while increasing number of layers for *rectangles-image* dataset

Kernel Parameters	**Loss in Percentage**
1	21.94
1×3	21.68
1×6	21.46
1×9	19.78
1×12	19.52
1×15	19.38
1×18	19.30
1×21	19.02

TABLE 4.3: Change in classifier performance while increasing number of layers for *convex* dataset

of n layers is used with kernel parameter is equal to '1' in each layer.

4.3 Conclusion

In this chapter we experimented on KFDA with multi-layer arc-cosine kernels. The result obtained are very promising. On *rectangles-image* dataset, the classifier performed even better than a DBN based model. On *convex* dataset, its performance was better than all shallow models and was comparable with that of deep models. One of the striking observation from these results is that, better performance is obtained when using either a highly non-linear arc-cosine kernel(degree > 1) or a multi-layer arc-cosine kernel with very large number of layers (above 10).

Chapter 5

Multi-layer Kernels in Structured Output Spaces

This chapter focuses on evaluating multi-layer arc-cosine kernels on structured output prediction problems. The contents of this chapter are organized as follows; section 5.1 gives a brief description about the large margin formulation of pattern recognition problem on structured output spaces, section 5.2 talks about the result of empirical study on multi-class and multi-label classification problems and section 5.3 concludes the chapter.

5.1 SVM in Structured Output Spaces

Typically machine learning algorithms are designed to produce flat real valued outputs; in the case of classification problems the output is a class label, for regression the output is a real number. For structured output learning algorithms, the output space has structured and interdependent variables, which is usually stored and processed as multi-dimensional arrays. For example, in the case of natural language parsing the output is a parse tree, in the case of image segmentation the output is the four 2D cordinates of bounding box surrounding the object. Two popular algorithms which works well in this domain are Conditional Random Fields(CRF) proposed by [Lafferty] et al. and Structural SVMs proposed by [Tsochantaridis] et al. In this project, we did our study on structural SVMs.

Structural SVMs are first introduced by [Tsochantaridis] et al. in 2005, and studied by [Joachims] et al. for simplifying the optimization problem while dealing with exponentially huge number of constraints. In particular, the number of constraints in the formulation of StructSVM is equal to the cardinality of the output space(which is exponential or even infinite). So decomposition methods like SMO which process each constraints explicitly is not suitable for these kind of problems. [Joachims] et al. proved that the optimization problem can be solved efficiently using cutting plane algorithm proposed by [Kelley]. They also proved that the number of iterations are independent of the number of training examples and provided an upper bound on the number of iterations.

5.1.1 StructSVM : Formulation

Structured output prediction describes the problem of learning a function

$$h : \mathcal{X} \longrightarrow \mathcal{Y}$$

where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space(structured). To learn h, we assume that a training sample of input-output pairs

$$S = ((x_1, y_1), \ldots, (x_n, y_n)) \in (\mathcal{X} \times \mathcal{Y})^n$$

is available and drawn i.i.d from a joint distribution $\mathrm{P}(\mathcal{X}, \mathcal{Y})$. Following empirical risk minimization principle, we will find an $h \in \mathcal{H}$ that minimizes the empirical risk

$$R_s^{\triangle} = \frac{1}{n} \sum_{i=1}^{n} \triangle(y_i, h(x_i))$$

Here $\triangle(y, \overline{y})$ denotes the **loss** associated with predicting $\overline{y}$ when y is the correct output. The formulation assumes that the loss function is arbitrary and should satisfy the follwing requirments.

$$\triangle(y, \overline{y}) = \begin{cases} > 0 \text{ for } y \neq \overline{y} \\ = 0 \text{ for } y = \overline{y} \end{cases}$$

StructSVM selects an $h \in \mathcal{H}$ that minimizes a regularized empirical risk on S. The general idea here is to learn a discriminant function $f : \mathcal{X} \times \mathcal{Y} \longrightarrow \mathbb{R}$ over input-output pairs from which one derives a prediction by maximizing f over all $y \in \mathcal{Y}$ for a given input x.

$$h_w(x) = \arg\max_{y \in \mathcal{Y}} f_w(x, y)$$

We assume that $f_w(x, y)$ is linear in some combined feature space relating x and y, denoted as $\Psi(x, y)$.

$$f_w(x, y) = (w \cdot \Psi(x, y))$$

Here $w \in \mathbb{R}^N$ is the parameter vector. Intuitively we can think of $f_w(x, y)$ as a compatibility function that measures how well the output y matches the given input x([Joachims] et al.). This combined feature representation is required in the formulation, since we assumed that the sample S is drawn from a joint distribution $\mathrm{P}(\mathcal{X}, \mathcal{Y})$. Depending upon the structure of the output space, $\Psi(x, y)$ is defined separately for different problem instances.

5.1.2 Margin Rescaling(MR) Formulation

In order to take the loss into consideration, we modify the soft-margin formulation used in SVMs. The soft-margin formulation is given by

$$\min_{w, \xi \geq 0} \quad \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i$$

$$\text{s.t } \forall i, \forall \overline{y} \in \mathcal{Y} \setminus y_i : w^T[\Psi(x_i, y_i) - \Psi(x_i, \overline{y})] \geq 1 - \xi_i, \; \xi_i \geq 0$$

Here ξ_i is the slack variable and C is the regularization parameter.

$$\xi_i = \max\{0, \max_{y \in \mathcal{Y} y_i}(1 - w^T[\Psi(x_i, y_i) - \Psi(x_i, \overline{y})])\}$$

As we have mentioned previously, this optimization problem is intractable for decomposition methods like SMO, since we have $\mathcal{O}(n|\mathcal{Y}|)$ constraints in the formulation. In Margin Rescaling formulation, the margin is adjusted according to

the loss. In particular, we adjust the position of the hinge by keeping its slope fixed. The loss in MR formulation is computed as

$$\triangle_{MR}(y, h_w(x)) = \max_{\overline{y} \in \mathcal{Y}} \{\triangle(y, \overline{y}) - (w^T[\Psi(x, y) - \Psi(x, \overline{y})])\} \ \geq \triangle(y, h_w(x))$$

and slack is obtained as $\xi = \max\{0, \triangle_{MR}(y, h_w(x))\}$. This leads to the following formulation

$$\min_{w, \xi \geq 0} \quad \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i$$

$$\text{s.t } \forall \overline{y_1} \in \mathcal{Y} : \ w^T[\Psi(x_1, y_1) - \Psi(x_1, \overline{y_1})] \geq \triangle(y_1, \overline{y_1}) - \xi_1$$

$$\vdots$$

$$\text{s.t } \forall \overline{y_n} \in \mathcal{Y} : \ w^T[\Psi(x_n, y_n) - \Psi(x_n, \overline{y_n})] \geq \triangle(y_n, \overline{y_n}) - \xi_n$$

Intuitively, the constraints ensures that the score of the correct label $w^T\Psi(x_i, y_i)$ must be greater than all other scores $w^T\Psi(x_i, \overline{y_i})$, $\forall \overline{y_i} \in \mathcal{Y} \backslash y_i$ by a required margin. In MR formulation, the margin is $\triangle(y_i, \overline{y_i})$.

5.1.3 Slack Rescaling(SR) Formulation

In Slack Rescaling formulation, the slack variables are rescaled according to the loss. In particular, the slope of the hinge loss function is adjusted while keeping its position fixed. In SR formulation the margin is 1. The loss in SR formulation is computed as

$$\triangle_{SR}(y, h_w(x)) = \max_{\overline{y} \in \mathcal{Y}} \{\triangle(y, \overline{y})(1 - (w^T[\Psi(x, y) - \Psi(x, \overline{y}))\}$$

and slack is obtained as $\xi = \max\{0, \triangle_{SR}(y, h_w(x))\}$. This leads to the following formulation

$$\min_{w, \xi \geq 0} \quad \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i$$

$$\text{s.t } \forall \overline{y_1} \in \mathcal{Y} : \ w^T[\Psi(x_1, y_1) - \Psi(x_1, \overline{y_1})] \geq 1 - \frac{\xi_1}{\triangle(y_1, \overline{y_1})}$$

$$\vdots$$

$$\text{s.t } \forall \overline{y_n} \in \mathcal{Y} : \; w^T[\Psi(x_n, y_n) - \Psi(x_n, \overline{y_n})] \geq 1 - \frac{\xi_n}{\triangle(y_n, \overline{y_n})}$$

Both of the above formulation has n slack variables, hence it is called n-slack formulation. These formulations can be converted into 1-slack formulation by summing up all the slack variables([Joachims] et al.). n-slack formumations have $\mathcal{O}(n|\mathcal{Y}|)$ constraints.

The solution space of this problem is a compact polyhedral convex set. The cutting plane algorithm finds the most violating constraint corresponding to each training example and add it to the working set. After each addition to the working set, we find a solution across all the constraints in working set. This effectively shrinks the size of the version space in a speedy manner. As the iteration continues, the number of constraint violations decreases and the algorithm converges. Every single cut in the convex set corresponds to a constraint violation. Instead of doing a step by step updation, the cutting plane algorithm cuts down a portion of the version space which results in faster convergence.

5.2 Experiments

Empirical study was conducted on multi-class and multi-label classification problems. For multi-class problems the loss function used was the absolute difference between labels, and for multi-label problems loss function was the hamming distance between labels expressed in binary form. Since the output space was finite, we used exhaustive search over $\mathcal{Y}$ in both cases, while finding the most violated constraints. Multi-label and multi-class classification problems are the simplest problem instances that can be studied using StructSVM, since their output space is finite.

The combined feature map $\Psi(x, y)$ was constructed as follows. Let $x \in \mathbb{R}^d$ and k be the number of classes. Suppose x is represented as $1 : x_1, \ldots, d : x_d$. Then for multi-class problems $\Psi(x, y)$ was obtained by shifting the indices by $(y - 1) \times d$ positions; i.e.,

$$\Psi(x, y) = (y - 1) \times d + 1 : x_1, \ldots, (y - 1) \times d + d : x_d$$

Dataset	Arc-Cosine Kernel	Other Kernel(best)
Scene Segentation (multilabel - 6 class)	30.35	30.60
Vehicle Dataset (multiclass - 4 class)	26.48	24.90
Iris Dataset (multiclass - 3 class)	1.67	3.33
Breast Cancer Wiscosin (binary)	0.98	0.98
Synthetic Data (multiclass - 7 class)	33.85	32.55

TABLE 5.1: Performance comparison of multi-layer arc-cosine kernel to other kernels in StructSVM framework.

For multi-label classification problems, we took the binary representation of y and from that we extracted all bit positions that are ON. Then $\Psi(x, y)$ is computed by applying the same shifting to all the extracted indices.
Implementation was done using [SVM$^{\text{struct}}$] library, by modifying its API functions for multi-label and multi-class problems. Table 5.1 lists the results of empirical study (value shown is the loss in percentage). The synthetic dataset was a multi-class problem instance available in [SVM$^{\text{struct}}$] library. Here the comparison was made between multi-layer (arc-cosine)kernel machines and commonly used single layer kernel machines.

5.3 Conclusion

In this chapter, we studied multi-layer kernels in structured output spaces. The experimental study was done on multi-label and multi-class problem instances. The results are competetive with single layer kernel machines. Multi-layer architectures are found to be effective in complex pattern recognition tasks. Hence the discriminating power of these multi-layer kernels must be tested in more complex structured output spaces on problems like natural language parsing, protein sequence alignment prediction etc.

Chapter 6

Conclusion and Future Works

This thesis presents a study of deep multi-layer kernels on supervised and unsupervised learning algorithms. In the supervised learning settings, multi-layer kernels are studied on structured output prediction problems and discriminant analysis methods. In the unsupervised learning settings, multi-layer kernels are used for feature learning task. In particular, we used MKMs and MKL for building a feature learning framework with kernel machines whose architecture and training is equivalent to existing deep learning algorithms.

In the unsupervised feature learning model with MKMs we experimented with single kernel (arc-cosine kernel) and mixed kernels. In some cases, the mixed kernel version was found to be superior than their single kernel counterparts. Features learned by the MKMs were visualized with tSNE algorithm to understand the separability of different classes.

Instead of using single kernel in each layer, a convex combination of multiple kernels were tried out using an unsupervised MKL formulation. The training of this model was done in a greedy layer-by-layer fashion. Experimental results indicates that, this ML-MKL model is superior to single-kernel MKMs. Though we had conducted empirical study on object recognition datasets like *cifar10* and text classification datasets like *20-newsgroups*, the results were not very impressive. In the case of *20-newsgroups* classification, multiple layers might be making the similarity information in the kernel matrix more noisy, since the performance was degrading after each layer is added. The empirical studies on supervised learning

settings also gave fruitful results. The KFDA algorithm with multi-layer kernels showed competitive performance with state of the art deep learning algorithms. Structured output learning algorithms also works well when multi-layer kernels were used.

6.1 Future Works

The empirical study conducted in this thesis is giving many insights, from which we can list out some potential future directions to explore.

The complexity of the model in terms of the number of layers and number of kernels in each layer are set by using cross-validation techniques in our experiments. The optimal number of layers and optimal number of kernels in each layer characterizes the structure of ML-MKL. A thoretical study on the optimal structure of ML-MKL model is a future extension for this work.

The training of ML-MKL framework in the current implementation is performed in a greedy layer-by-layer fashion, by iteratively adding layers. This can be thought of as unsupervised pre-training done in typical deep learning algorithms. Then we can devise a supervised fine-tuning mechanism for this model, by taking class labels into account, which may change either the kernel parameters or weights.

The deep learning algorithms are found to be particularly effective for learning complex decision functions. Structured output prediction tasks like parse tree prediction, protein segment alignment prediction etc. need very complex decision functions. Thus, the deep multi-layer kernel machines can be studied in such domains.

The results from KFDA with multi-layer kernels is having some interesting aspects. The algorithm is performing well, either when using a highly non-linear arc-cosine kernel or when very large number of layers are used in the kernel function. The obscurity behind the need of such complex kernels for discriminant analysis can be explored further.

Appendix A

Derivation of the objective function $J(\mu)$

The objective function to be minimized is given by

$$\min_{k \in K_{conv}} \frac{1}{2} \sum_{i=1}^{n} \left\| x_i - \sum_{x_j \in B_i} k_{ij} x_j \right\|^2 + \gamma * \sum_{i=1}^{n} \sum_{x_j \in B_i} k_{ij} \left\| x_i - x_j \right\|^2$$

Expanding the norm on the first part of the sum

$$\begin{aligned} \min_{k \in K_{conv}} \frac{1}{2} \sum_{i=1}^{n} \Bigg(\left\| x_i \right\|^2 - 2 * \sum_{x_j \in B_i} k_{ij} (x_i \cdot x_j) + \\ k_i k_i^T \circ d_i d_i^T \circ X^T X \Bigg) + \gamma * \sum_{i=1}^{n} \sum_{x_j \in B_i} k_{ij} \left\| x_i - x_j \right\|^2 \end{aligned} \tag{A.1}$$

The notation '$\circ$' denotes elementwise multiplication of two vectors. Here the summation $\sum_{i=1}^{n} \left\| x_i \right\|^2$ can be discarded, since it is independent of the optimization parameters. Substituting $X^T X = P$,

$$\sum_{x_j \in B_i} k_{ij} (x_i \cdot x_j) = k_i \circ d_i \circ p_i$$

and

$$\sum_{x_j \in B_i} k_{ij} \left\| x_i - x_j \right\|^2 = k_i \circ d_i \circ v_i$$

in A.1 we will get the simplified objective function

$$\min_{k \in K_{conv}} \sum_{i=1}^{n} \Big(k_i k_i^T \circ d_i d_i^T \circ P + 2\big(\gamma * k_i \circ v_i \circ d_i - k_i \circ p_i \circ d_i\big) \Big) \tag{A.2}$$

Here p_i and v_i are columns of P and M corresponding to x_i respectively. Substituting $k_i = \sum_{t=1}^{m} \mu_t k_{t,i}$ in A.2 we will get

$$\min_{\mu \in \Delta} \mu^T \left(\sum_{t=1}^{m} \sum_{i=1}^{n} k_{t,i} k_{t,i}^T \circ d_i d_i^T \circ P \right)^T \mu + z^T \mu$$

which is the objective function $J(\mu)$. Here $[z]_t = \sum_{i=1}^{n} (2\gamma v_i \circ d_i - 2 p_i \circ d_i)^T k_{t,i}$ and $k_{t,i} = \Big[k^t(x_i, x_1), \ldots, k^t(x_i, x_n) \Big]^T$ is the i^{th} column of the t^{th} kernel matrix.

Appendix B

Derivation of cost function $\mathcal{J}(\alpha)$ in KFDA

The cost function is given as

$$\mathcal{J}(f) = \frac{f^T S_B^\phi f}{f^T S_W^\phi f} \tag{B.1}$$

We have

$$f^T m_i^\phi = \frac{1}{n_i} \sum_{j=1}^{n} \sum_{k=1}^{n_i} \alpha_j k(x_j, x_k^i) = \alpha^T M_i \tag{B.2}$$

applying B.2 in the numerator of B.1 we get

$$\begin{aligned} f^T S_B^\phi f &= f^T (m_1^\phi - m_2^\phi)(m_1^\phi - m_2^\phi)^T f \\ &= (f^T m_1^\phi - f^T m_2^\phi) \cdot (f^T m_1^\phi - f^T m_2^\phi) \\ &= (\alpha^T M_1 - \alpha^T M_2) \cdot (\alpha^T M_1 - \alpha^T M_2) \\ &= \alpha^T (M_1 - M_2)(M_1 - M_2)^T \alpha \\ &= \alpha^T M \alpha \end{aligned}$$

where $M = (M_1 - M_2)(M_1 - M_2)^T$. Applying $f = \sum_{i=1}^{n} \alpha_i \phi(x_i)$ in the denominator of B.1

$$f^T S_W^\phi f = (\sum_{i=1}^{n} \alpha_i \phi(x_i))^T \sum_{j=1,2} \sum_{x \in X_j} (\phi(x) - m_j^\phi)(\phi(x) - m_j^\phi)^T (\sum_{i=1}^{n} \alpha_i \phi(x_i)) \tag{B.3}$$

To simplify the notations, define

$$P_{ij} = \sum_{x \in X_i} \phi(x_j) \cdot \phi(x)$$

Then

$$\sum_{i=1}^{n} \sum_{j=1,2} \sum_{x \in X_j} \alpha_i (\phi(x_i) \cdot \phi(x)) = \alpha^T P_1 + \alpha^T P_2 \tag{B.4}$$

Applying B.4 in B.3 we get

$$\begin{aligned}
f^T S_W^\phi f &= (\alpha^T P_1 - \alpha^T M_1) \cdot (\alpha^T P_1 - \alpha^T M_1) + (\alpha^T P_2 - \alpha^T M_2) \cdot (\alpha^T P_2 - \alpha^T M_2) \\
&= \alpha^T (P_1 - M_1)(P_1 - M_1)^T \alpha + \alpha^T (P_2 - M_2)(P_2 - M_2)^T \alpha \\
&= \alpha^T K_1 (I - \mathbf{1}_{n_1}) K_1^T \alpha + \alpha^T K_2 (I - \mathbf{1}_{n_2}) K_2^T \alpha \\
&= \alpha^T \Big(\sum_{i=1,2} K_i (I - \mathbf{1}_{n_i}) K_i^T \Big) \alpha \\
&= \alpha^T N \alpha
\end{aligned}$$

where $K_j = \sum_{i=1}^{n} \sum_{x \in X_j} k(x_i, x)$, I is the identity matrix, $\mathbf{1}_{n_j}$ is the matrix with with all entries $\frac{1}{n_j}$ and $N = \sum_{i=1,2} K_i (I - \mathbf{1}_{n_i}) K_i^T$. Thus the cost function becomes

$$\mathcal{J}(f) = \frac{f^T S_B^\phi f}{f^T S_W^\phi f} = \frac{\alpha^T M \alpha}{\alpha^T N \alpha}$$

Bibliography

[Cho] Y. Cho, L.K. Saul, Kernel Methods for Deep Learning. *Advances in Neural Information Processing Systems(NIPS)* Volume 22, 342 – 350, 2009.

[Bengio, 2007] Yoshua Bengio and Yann LeCun, Scaling learning algorithms towards AI. in Bottou, L. and Chapelle, O. and DeCoste, D. and Weston, J. (Eds) *Large-Scale Kernel Machines*, MIT Press, 2007.

[Bengio, 2013] Yoshua Bengio, Aaron Courville and Pascal Vincent, Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence(TPAMI)* Volume 35, 1798–1828, August 2013.

[Bengio, 2009] Yoshua Bengio, Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning* Volume 2, 1 – 127, January 2009.

[Graham] Benjamin Graham, Fractional Max-pooling. *arxiv:cs/arXiv:1412.6071*, 2014.

[Wan] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun and Rob Fergus, Regularization of Neural Networks using DropConnect. *JMLR Proceedings* Volume 28, 1058 – 1066, 2013.

[Hinton 2012] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath and Brian Kingsbury, Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine* Volume 29, 82 – 97, 2012.

[Alex] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems(NIPS)*, 1106 – 1114, 2012.

[Collobert] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *International Conference on Machine Learning(ICML)*, 2008.

[Hinton, 2006] Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh, A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[LeCun] Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner, Gradient based learning applied to document recognition. *Proceedings of the IEEE*, 22782324, November 1998.

[Raina] Rajat Raina, Anand Madhavan and Andrew Y. Ng, Large-scale Deep Unsupervised Learning using Graphics Processors. *International Conference on Machine Learning*, 2009.

[Dean] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, MarcAurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang and Andrew Y. Ng, Large Scale Distributed Deep Networks. *Advances in Neural Information Processing Systems*, 1223 – 1231, 2012.

[Graphlab] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin and J. Hellerstein, GraphLab: A New Framework For Parallel Machine Learning. *26th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2010.

[Hugo] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio, Online companion for the paper An empirical evaluation of deep architectures on problems with many factors of variation. [Online]. Available: `http://www.iro.umontreal.ca/lisa/twiki/bin/view.cgi/Public/DeepVsShallowComparisonICML2007` 2014.

[Smola] Bernhard Schölkopf, Alexander Smola and Klaus-Robert Müller, Nonlinear Component Analysis As a Kernel Eigenvalue Problem. *Journal of Neural Computation* Volume 10, 1299–1319, 1998.

[Zhuang, 2011] J. Zhuang, Jialei Wang, Steven C.H Hoi, Xiangyang Lan, Unsupervised Multiple Kernel Learning. *Journal of Machine Learning Research(JMLR)* Volume 20, 129–144, 2011.

[Gert] Gert R. G. Lanckriet, Nello Cristianini, Peter L. Bartlett, Laurent El Ghaoui, and Michael I. Jordan, Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, Volume 5, 27–72, 2004.

[scikit-learn] 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(JMLR)* Volume 12, 2825–2830, 2011.

[MNIST] Y. LeCun and C. Cortes, The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

[Maaten] L.J.P. van der Maaten and G.E. Hinton, Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research(JMLR)*, volume 9, 2579–2605, 2008.

[Tsochantaridis] I. Tsochantaridis, T. Joachims, T. Hoffman and Y. Altun, Large Margin Methods for Structured and Interdependent Output SPaces. *Journal of Machine Learning Research(JMLR) 6*, 1453–1484, 2005.

[Joachims] T. Joachims, T. Finely, C. John Nu, Cutting-Plane Training on Structural SVMs. *Journal of Machine Learning 77(1)*, 27–59, 2009.

[Kelley] J.E Kelley, The Cutting-plane Method for Solving Convex Programs. *Journal of the Society for Industrial and Applied Mathematics(SIAM) 8*, 703–712, 1960.

[Zhuang] Jinfeng Zhuang, Ivor W. Tsang and Steven C.H. Hoi, Two-layer Multiple Kernel Learning. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp 909–917, 2011.

[Ilyes] Ilyes Rebai, Yassine BenAyed and Walid Mahdi, Deep Multilayer Multiple Kernel Learning. *Journal of Neural Computing and Applications*, pp 1–10, 2015.

[Corinna] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh, Two-stage learning kernel algorithms. *International Conference on Machine Learning*, pp 239–246, 2010.

[Sebastian] Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, and Klaus-Robert Müller, Fisher Discriminant Analysis With Kernels. *Journel of Neural Networks for Signal Processing*, pp 41–48, 1999.

[Lafferty] J. Lafferty, A. McCallum, and F. Pereira, Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning*, 2001.

[SVM$^{\text{struct}}$] Thorsten Joachims, SVM$^{\text{struct}}$: Support Vector Machine for Complex Outputs. `http://www.cs.cornell.edu/People/tj/svm_light/svm_struct.html`

Source Code

All the code-works done as part of this project is publicly hosted in github. The source code is available in the following repository

`https://github.com/akhilpm/Masters-Project/`

List of papers based on thesis

1. Akhil P M, Asharaf S, Sumitra S, *Unsupervisd MKL in Multi-layer Kernel Machines*(paper under preparation).