MULTILAYER KERNEL MACHINES

Siyu Gao
Department of Mathematics
NYU Shanghai
sg5578@nyu.edu

May 19, 2021

ABSTRACT

In this project, we reproduce and generate the Multilayer Kernel Machines(MKM) based on Cho and Saul's paper [4]. MKM is proposed as a combined algorithm of Kernel PCA and Feature Selection with a final Classifier. We not only reproduce the MKM following the way in original paper, but also change the design of MKMs and tested these new designs on different kernels and classifiers. Experiments are carried mainly on MNIST dataset for classification problem. It turns out that the original MKM algorithm is very sensitive to hyper-parameter of feature selection which can significantly undermine the accuracy. For general MKMs, linear and polynomial kernels performs quite well. For MKMs with Arc-cosine Kerenl, lower order with less layer always outperform others,which is consistent with results in original paper. Also, we found that model KernelPCA+KNN with arc-cosine kernel at order 0 with only 1 layer achieves the best stable performance.

Keywords kernel machine · kernel methods · deep learning · arc-cosine kernel

1 Introduction

Deep learning has been proved to be powerful in achieving higher level data abstraction and consolidating both unsupervised and supervised learning algorithms[13]. Models built in deep architectures learn complex mappings by transforming their inputs through multilayers of non-linear processing [4].The deep and non-local characters give models ability to learn the nature of highly variable data from fewer samples. Large parameters and hidden layers appeared in the model can facilitate the abstraction required for the generalization and incorporating prior knowledge in the training structurally and systematically [12]. There are already a lot of successful applications of deep learning in the fields of image recognition [17], speech recognition [9],text mining and etc.

Due to the non-linearity of real world datasets, kernel methods which build a bridge between linearity and non-linearity have been applied together with a lot of well known supervised and unsupervised learning paradigms, for example, support vector machines (SVM) and Principle Component Analysis (PCA). In the kernel induced (possibly high-dimensionaly) feature space, the distribution of data becomes more amenable especially for the classification problem [13]. Also, since kernel tricks only require the inner product, the complicated procedure to find the feature map and compute the mapping vector in the feature space could be simplified.[8].Unlike the neural networks, kernel machines also guarantee structural risk minimization and global optimal solution. Such advantages motivated researchers to study the combination of deep learning and kernel machines-deep kernel machines. There are many directions in studying deep kernel machines including kernel machines as the final classifier of deep networks, kernelization in deep neural networks(NN) [11] and build kernel machines with deep structure[13].In this project, we will study a deep kernel machine of the last type and it's called Multilayer Kernel Machines (MKM) [4].

Multilayer Kernel Machines (MKM) is proposed together with the family of Arc-cosine Kernels in Cho and Saul's paper [4]. It combines both supervised and unsupervised algorithm to perform classification tasks. MKM has kernel PCA and feature selection procedure in each layer and a classifier in the final output layer. In Cho and Saul's paper, they mostly focus on the MKM using arc-cosine kernel,mutual information feature selection methods and

LMNN [16] as the final classifier to demonstrate the property of arc-cosine kernel, but they don't study the design of MKM in details generate the MKM framework on experiments with other kernels or classifiers. Therefore, in this project, we break MKM desgin into 3 go-forward parts (kernel+ classifier, kernel+ feature selection + classifier, kernel + metric learning + classifier) and test each of them. Plus, not only for arc-coseine kernel and LMNN classifier used in original paper, other kernel and classifier are also tested in each part.

Here is an outline for the following sections. In Section 2, we will give the needed background on Kernel PCA, Arc-cosine Kernel and Large Margin Nearest Neighbor (LMNN) to facilitate the understanding of experiments. In Section 3, some works studying same topic on MKM will be presented to compare with the MKM in this project. In Section 4, we will introduce the original MKM in details including the its algorithm, the detailed implementation of layers, feature selection and classification. In Section 5, we show the experiments details and results under different design of MKM and different ingredients. In the result summary, I will try to explain the underlying reasons of results. Lastly, in Secion 6,we will discuss the problems of current experiments and future works.
The code of this project can be found in this Github link.

2 Background

Arc-Cosine Kernel Arc-Cosine Kernel is a family of kernel functions for computing the similarity between $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$. Here is the definition:
Definition: For $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, the **arc-cosine** function via the integral representation is :

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

To better understand the influence of inputs, it can be rewrite as

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

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

For Arc-Cosine Kernel with order equals to 0,1,2, we have

$$\begin{aligned} J_0(\theta) &= \pi - \theta \\ J_1(\theta) &= \sin\theta + (\pi-\theta)\cos\theta \\ J_2(\theta) &= 3\sin\theta\cos\theta + (\pi-\theta)(1+2\cos^2\theta) \end{aligned}$$

that we can compute the kernel at these order directly and also iteratively derive the higher orders.
Arc-cosine kernel can be viewed as the inner product between mappings of an single-layer threshold networks with width tends to infinity [4]. It mimic the computation of large width neural networks. The composition of it's feature map gives it a multilayer kernels that theoretically mimic multilayer random featured fully connected threshold networks with certain activation functions.

Large Margin Nearest Neighbor (LMNN) LMNN is one of the so-called linear metric learning methods to learn a distance measure M [16] being the distance between two data points $\vec{x_i}$ and $\vec{x_j}$ that equals to

$$d(\vec{x_i}, \vec{x_j}) = (\vec{x_i} - \vec{x_j})^T M (\vec{x_i} - \vec{x_j})$$

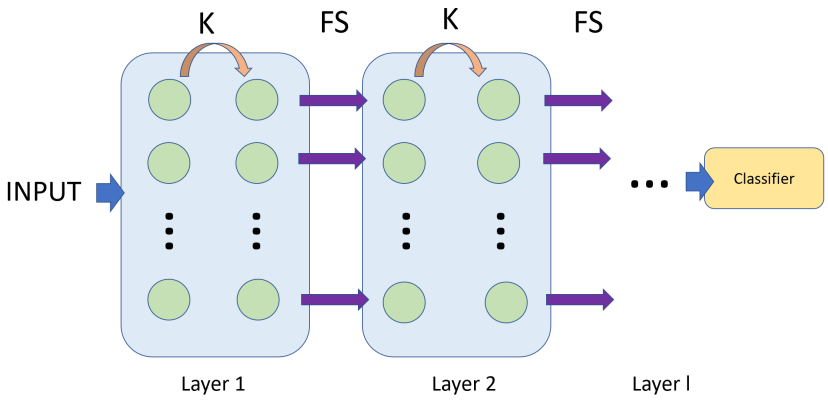
There are two types of points in the algorithm: target neighbors and imposters.The target neighbors are the data points that should become nearest neighbors under the learned metric. They share the same label with $\vec{x_i}$. While imposters are data points who are also nearest neighbor of $\vec{x_i}$ but not in the same class. Basically,for every data point $\vec{x_i}$, we want the target neighbors to be close and the impostors to be far away. The final optimization problem is given as following:

$$\min_{\mathbf{M}} \sum_{i,j \in N_i} d(\vec{x}_i, \vec{x}_j) + \lambda \sum_{i,j,l} \xi_{ijl}$$
$$\forall_{i,j \in N_i, l, y_l \neq y_i}$$
$$d(\vec{x}_i, \vec{x}_j) + 1 - d(\vec{x}_i, \vec{x}_l) \leq \xi_{ijl}$$
$$\xi_{ijl} \geq 0$$
$$\mathbf{M} \succeq 0$$

3 Related Works

Many researchers have noticed the connection between kernel methods and deep learning. As mentioned in a deep kernel machines survey [13], a Deep Core Vector Machines are built based on arc-cosine kernel to exhibit the potential of scaling the deep kernel-based SVM[1].Also, other application based on arc-cosine kernel machines on signal processing [9], speech recognition[3], Genome-Based Prediction[6] and etc. have flourished in the recent 10 years.
More and more people are studying the relation between deep learning and kernel methods[2]. Many people are trying to build models share both advantages of deep learning and kernel methods in fields including kernelization on deep neural networks[7, 11] and deep multilayer kernel learning[18, 15]. Generally, the design of MKM share the similar ideas with them.

4 MKM Framework



- **Multilayer Kernel Mathines Algorithms**
  A MKM is constructed by input, middle layer and output layer. We call the first middle layer as layer 1 and the last middle layer as layer l. Each middle layer can be broken into two part: K-part and FS-part. K-part refers to the Kernel PCA operations and FS part refers to feature selection methods.

  In the training process, in each middle layer, data are fitted to K-part in an unsupervised style and then fitted to FS-part in a supervised way. From the nature of K-part and FS-part, we know that the number of features for each data point will be reduced every time transferred by a middle layer. When l middle layers are done, the resutling data will have the same number of data points but with the number of features far less than the original one. Then, these resulting data will be fed to the final classifier and we will get a trained classifier C.

  In the test process, we can not simply fit the training data to the trained classifier C in the previous training process since the test dataset has the different feature dimension compared with that of C. Yet,to test the MKM trained above, we need to thoroughly bookkeep the middle layer operation done in the training process and reuse such process in the test process. That is we need to write down the components and feature label used of each layer in the training and take this ingredients to retrain the test set i.e. process the feature extraction. The resulting test data could be finally predicted by C and calculate the accuracy score.

- **Feature Selection Algorithm**
  Kernel PCA (K-part) background have been provided in the Section 2, so we only discuss Feature Selection

Algorithm (i.e. FS-part) here. In each middle layer, after getting the data after K-part, we calculate the mutual information between each feature with each label and give a mutual information score for each label. The higher score implies that the label contains more information about the label and should be kept. We choose the highest w features and send them to the next layer.

5 Experiments and Main Results

Following the MKM framework we discussed above, we implement different types of multilayer machines. Based on the MKM structure, we divide the realization of functions into three parts: (1) PCA or Kernel PCA layers to reduce the dimension of feature; (2) Feature selection methods to prune away the uninformative feature that could be redundant to the training and model generalization; (3) Final Classifier with a linear metric learning method. During the implementation, we do the test by changing the three realization methods and control other settings at the same time. For example, as the original paper has shown that MKMs work well with Kernel PCA layer using Arc-cosine kernel, we will change the types of kernel or the parameter settings in the Arc-cosine kernel family to see how performance will vary under such changes. Similar test methods are applied to other realization methods,too.

In the below experiments, datasets are setted by default as MNIST if no special datasets are mentioned. The size of training set and test set are 3000 and 1000.(It may be too small to train the model. however, due to the computer memory limits, it's the biggest size we could choose without a ValueError or MemoryError). To avoid the cheat on test set, we divide 10000 mnist samples into two 5000 parts. The 3000 training data are sampled from the former samples and 1000 test data are sampled from the latter one. In addition, we use the accuracy score to measure the performance. For each performance score, we test the same settings 5 times on sampled datasets from the same sampling pool and take the average of score as the final accuracy. Lastly, the benchmark we choose here is SVM+Arc-cosine Kernel at it's best performance order (order = 0) tested on MNIST. As it shown in Figure 1 Right Plot, it has the accuracy around 0.810.

5.1 PCA + KNN

It could be strange to test multilayer machines without kernel in the MKM experiments, but this multilayer PCA machine combined with classifier KNN is tested to distinguish kernel's contribution to the training and the contributions from multilayer structures. In the practice, it's also an important step for us to build the structure for a real MKM.

Without the feature selection, this model simply use PCA repeatedly for 1 layers and trained a classifier

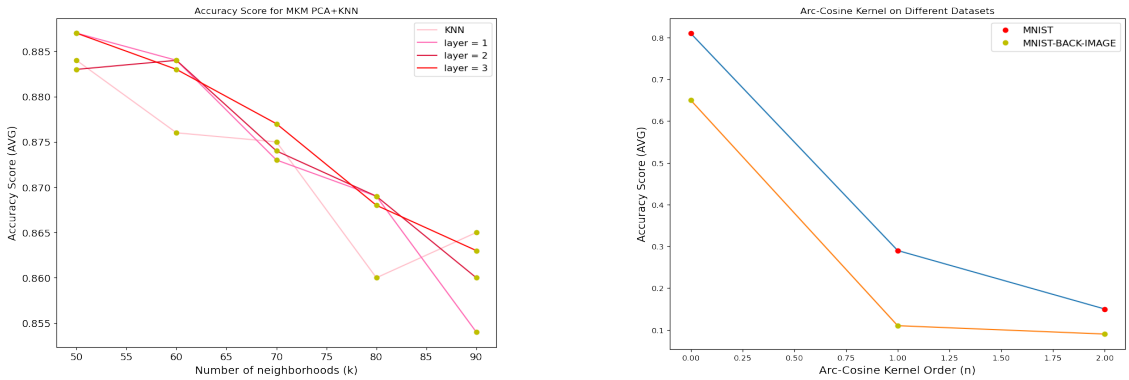


Figure 1: Left Plot: Showing the performance of PCA+KNN model vary under different layers and k-neighbors. The best performance is achieved when k=50 at with 1 or 2 layers having accuracy 0.885-0.889.
Right Plot: Showing the performance of SVM+Arc-cosine Kernel at different orders on both MNIST and MNIST-back-image. The best performance on MNIST achieved at 0-order Arc-cosine around 0.810 and on MNIST-back-image achieved at 0-order Arc-cosine around 0.650.

(here is KNN) using the resulting smaller number of features. From the figure below, we can see that it doesn't vary a lot between 1 layer to 3 layers. The best performance of PCA+KNN achieved between 0.885-0.889. Compared with the bench mark using SVM combined with Arc-cosine kernel here, it has a slightly improvement. Also, it achieves better result than using KNN without PCA but the improvements is not stable under different k-neighbors setting. The repeated layer of PCA doesn't doesn't benefit much from the multilayer architecture in this case but it indeed improve

the naive KNN.

5.2 MKM: KPCA + KNN

Now, we change the PCA used above to kernel PCA. Here, we use different kernels and different k-neighbors to build the model.For kernels, we test linear, polynomial,gaussian and Arc-cosine kernels.

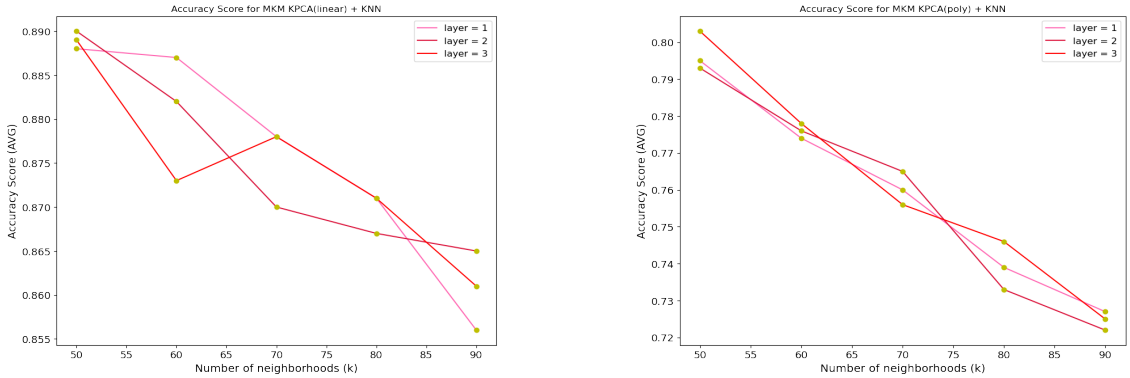


Figure 2: Left Plot: Showing the performance of KPCA(linear kernel)+KNN model vary under different layers and k-neighbors. The best performance is achieved when k=50 at with 2 layers having accuracy 0.890.
Right Plot: Showing the performance of KPCA(polynomial kernel)+KNN model vary under different layers and k-neighbors. The best performance is achieved when k=50 at with 3 layers having accuracy 0.806.

Compared with benchmarck SVM+Arc-cosine(0), KPAC(linear)+KNN outperforms the benchmark at k ranging from 50 to 90 which suggests that it's a quite stable outperforming results. However, KPCA(polynomial)+KNN only achieve the benchmark at the best result at k=50. As shown in the Figure 2, both models don't show significant performance difference under single layer or multiple layers. Though in different extents, they both show the trend that performance will decrease when k-neighbor is getting larger.

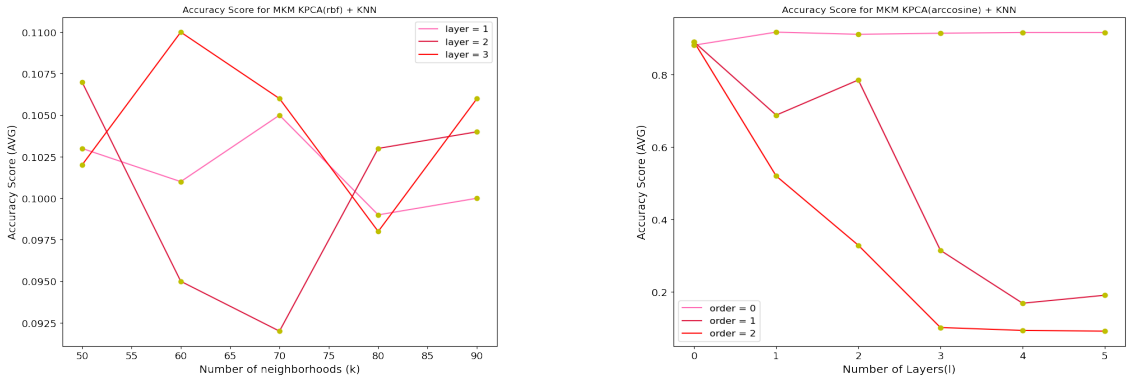


Figure 3: Left Plot: Showing the performance of KPCA(gaussian kernel)+KNN model vary under different layers and k-neighbors. It shows very pool performance.
Right Plot: Showing the performance of KPCA(arccosine kernel)+KNN model vary under different layers and orders. The best performance is achieved at 1 layer with order 0 having accuracy 0.913.

Compared with benchmark SVM+Arc-cosine(0), KPCA(arccosine)+KNN at order 0 outperforms the benchmark with accuracy at 0.913. As shown in Figure 3 Right Plot, it is stable with the layer number changing. Arc-cosine kernel at other orders (1,2) don't have satisfying results and accuracy decline dramatically when layer number is increasing. It's likely that lower order implies better performance. Here, we set k=50 when training, k=10,30,100 are also tried but don't have much influence on results. Suprisingly, gaussian kernel performs very pool in this case.

5.3 MKM: KPCA + LMNN

In this MKM design, we test the model on linear, polynomial, gaussian and Arc-cosine(kernel) with KNN combined with a metric learning algorithms(LMNN).

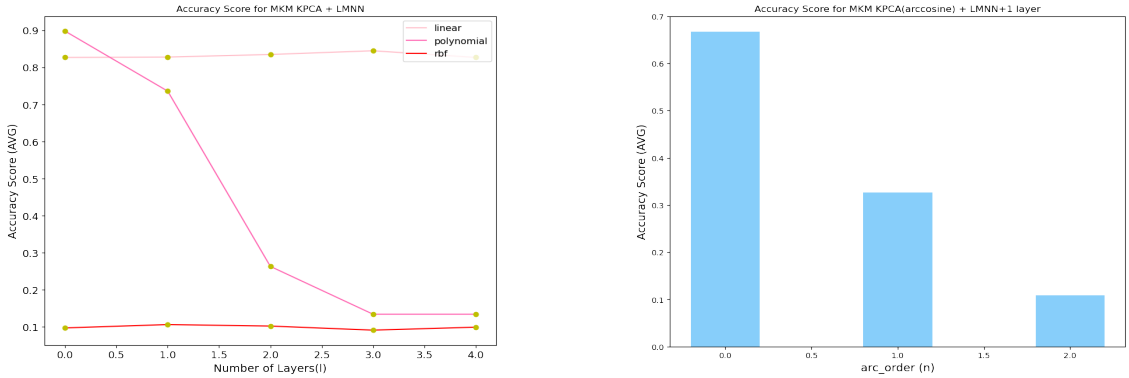


Figure 4: Left Plot: Showing the performance of KPCA(linear,polynomial,gaussian)+LMNN model vary under different layers. KPCA(polynomial) shows the best performance at layer 1 with accuracy 0.898. KPCA(gaussain) still performs the worst.
Right Plot: Showing the performance of KPCA(arccosine kernel)+LMNN model vary under different orders. Here we choose with only 1 layer MKM. The best performance is achieved at 1 layer with order 0 having accuracy 0.667.

For the former three kernels, we test them with LMNN on different layers ranging from 1 to 5.The underlying model KNN are setted to k=50 according to the best performer in Section 5.2. KPCA(polynomial) achieves the best performance at layer 1 with accuracy 0.898 which outperforms our benchmark SVM+Arc-cosine(0).Besides, KPCA(linear) achieves stable performance over all layers at accuracy slightly above 0.8 which is still competitive with the benchmark. KPCA(gaussian) still perform the worst in this case which is consistent with the result in Section 5.2 since LMNN is just a metric learning algorithm used here to enhance the KNN Classifier. If we look along the x-axis, the quick drops on KPCA(polynomial) accuracy score results demonstrate again that higher layers tend to worse performance in MKM.

For the right plot,we choose the best perform layer in the last section used as the layer number to train KPCA(arccosine). And the result is consistent with before experiments that KPCA(arc-cosine) performs the best at order 0.

5.4 MKM: KPCA + FS + KNN/LMNN

In this model design, we reproduce the feature selection methods from Cho and Saul's paper [4] but obtain quite poor result in all cases. All kernels get the accuracy at around 0.100 or even below, while this methods seems to work well in original paper after a very costly hyper-parameter tuned process.

Recall that in the FS-part we mentioned in Section 4, we need to keep the top w features with highest score. This w is determined in Cho and Saul's paper using cross validation but very complicated way. They compute the error rates of a basic KNN(k) classifier with Euclidean distance in the feature space. They compute the error rates a a held-out set of validation examples for k ranging from 1 to 15 and w ranging from 10 to 300, and record the best w to decide w for the next layer. In the practice, we tried to tuned the w ranging from 10 to 30 with step 5 and k ranging from 1 to 3 while there is no improvement on results. Also, w = 100,200,300 are tried but the result is still poor.

In my opinion, this feature selection might be very sensitive to the parameter chosen and also contribute a lot to the performance realization using KPCA(arccosine kernel)+FS+LMNN MKM. And this tought could also be supported by the result in Section 5.3 since the KPCA(arccosine)+LMNN doesn't perform as well as KPCA(other kernels)+LMNN.

5.5 Time Complexity Discussion

Training time is always a core problem in training kernel machines since the computational complexity grows quadratically with data samples size because of the required large kernel matrix [14]. It's yet quite challenging to deal with massive datasets with kernel machines. Plus, optimization in Kernel learning is also super time consuming to learn an optimal kernel function or an optimal hyper-parameter, which is very important to the model(like the design of MKM in Section 5.4)[10].

We encounter the both problems in this project.

- Firstly, since it's kernel needed to be computed in each middle layer of MKMs, the experiments shown here from Section 5.2 to Section 5.4 all took hours to train and we even don't train them over 6 layers. Especially, for MKM, according to the Section 4, we can see kernel matrix need to be computed in both training process and test process (the components kept in the training need to be applied to the kernel matrix again).
- Secondly, the parameters of polynomial kernels and gaussian kernels in Section 5.2 needed to be tuned before training on KNN.

In this case, there are works built based on Arc-cosines kernel but can reduce the training time. Deep Core Vector Machine (DCVM) is built based on Arc-cosine kernel and exhibit potential of scaling the deep kernel based support vector machines [13]. The DCVM exhibit the both the reduced training time and higher accuracy compared with Support Vector Machine (SVM) and Core vector Machine(CVM)[1].

5.6 Main Results and Discussion

Here are some observations from the experiments above.

- In all the MKMs designs above, low layers tend to outperform the high layer and the benchmark SVM+arc-cosine(0).
- In MKMs incoperating with Arc-cosine Kernels, lower Arc-cosine order tends to outperforms the higher orders and sometimes outperforms benchmark SVM+arc-cosine(0).
- LMNN in classifier implemented in the original paper actually reduce the accuracy in this project for arc-cosine kernels.

In summary, KPCA(arc-cosine)+KNN with order 0 layer 1 is the best performer in our project tested on MNIST so far. It achieves the best accuracy score but seems to violate the initial intention in designing a MKM since such model is quite stable with layer changing. In the Cho's following paper to analyze the geometry properties of different orders of arc-cosine kernels[5]. He mentioned that for kernels with order greater than 1, the surface of the corresponding Hilbert Space can be described as manifold while order 0 can't be explained in this way at all. It seems that order 0 better preserve the angle between data points which is quite important for the following KNN Classification.

Another significant observation is the lower accuracy with more layers. In my opinion, it's probably why original authors would add feature selection part to prune away the redundant information. High noises transferred between layers make the final classifier lost.(Interestingly, arc-order = 0 is not influenced by such noise). Unfortunately, we didn't find a successful way to reproduce authors' feature selection methods and further works are needed for future study. Since festure selection indeed make the MKM not robust on the both kernel and dataset generalization.

For the observation on LMNN, in my opinion, it may also related to the remained redundant information mentioned above.

6 Problems and Future Work

Most of problems rise in the experiments. Due to the limitation of computer and time, the training set size(3000) we used is quite small compared with the usual sets and theoretically, it indeed undermine the accuracy. Also, limitations on computers also give me a lot of ValueErrors and MemoryErrors since it's very easy to over the inf value setted and don't have enough space for multiple layer computations. Especially when we try to reproduce the composition version

of kernel, ValueError rises only with 1 composition.

The future works rely on modifying and analyzing the feature selection methods.It is likely to be the most sensitive part in MKMs and understanding how it will influence the training is quite interesting. Beyond the MKMs, as other study direction of deep kernel machines mentioned in Section 1. I would like to try more on ther kernelization of deep neural networks part. Since Arc-cosine kernel is proposed as a powerful linear tool with deep features, I'm wondering if it will have better performance when combined with real deep networks.

References

[1] A. L. Afzal and S. Asharaf. Deep kernel learning in core vector machines. *Pattern Anal. Appl.*, 21(3):721–729, August 2018.

[2] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 541–549. PMLR, 10–15 Jul 2018.

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

[4] Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

[5] Youngmin Cho and Lawrence K. Saul. Analysis and extension of arc-cosine kernels for large margin classification. *CoRR*, abs/1112.3712, 2011.

[6] José Crossa, Johannes W.R. Martini, Daniel Gianola, Paulino Pérez-Rodríguez, Diego Jarquin, Philomin Juliana, Osval Montesinos-López, and Jaime Cuevas. Deep kernel and deep learning for genome-based prediction of single traits in multienvironment breeding trials. *Frontiers in Genetics*, 10:1168, 2019.

[7] Geoffrey E Hinton and Russ R Salakhutdinov. Using deep belief nets to learn covariance kernels for gaussian processes. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008.

[8] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *The Annals of Statistics*, 36(3):1171 – 1220, 2008.

[9] Po-Sen Huang, Haim Avron, Tara N. Sainath, Vikas Sindhwani, and Bhuvana Ramabhadran. Kernel methods match deep neural networks on timit. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 205–209, 2014.

[10] Linh Le, Jie Hao, Ying Xie, and Jennifer Priestley. Deep kernel: Learning kernel function from data using deep neural network. In *2016 IEEE/ACM 3rd International Conference on Big Data Computing Applications and Technologies (BDCAT)*, pages 1–7, 2016.

[11] Julien Mairal, Piotr Koniusz, Zaïd Harchaoui, and Cordelia Schmid. Convolutional kernel networks. *CoRR*, abs/1406.3332, 2014.

[12] Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Kernel analysis of deep networks. *Journal of Machine Learning Research*, 12(78):2563–2581, 2011.

[13] Nair Nikhitha, Afzal al, and Asharaf S. Deep kernel machines: a survey. *Pattern Analysis and Applications*, 24:1–20, 05 2021.

[14] Huan Song, Jayaraman J. Thiagarajan, Prasanna Sattigeri, and Andreas Spanias. Optimizing kernel machines using deep learning, 2017.

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

[16] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(9):207–244, 2009.

[17] Meiyin Wu and Li Chen. Image recognition based on deep learning. In *2015 Chinese Automation Congress (CAC)*, pages 542–546, 2015.

[18] Jinfeng Zhuang, Ivor W. Tsang, and Steven C.H. Hoi. Two-layer multiple kernel learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 909–917, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.