# Neural Generalization of Multiple Kernel Learning

**Ahmad Navid Ghanizadeh[1] · Kamaledin Ghiasi-Shirazi[2]** · **Reza Monsefi[2]** ·
**Mohammadreza Qaraei[3]**

## Abstract

Multiple Kernel Learning (MKL) is a conventional way to learn the kernel function in kernel-based methods. MKL algorithms enhance the performance of kernel methods. However, these methods have a lower complexity compared to deep models and are inferior to them regarding recognition accuracy. Deep learning models can learn complex functions by applying nonlinear transformations to data through several layers. In this paper, we show that a typical MKL algorithm can be interpreted as a one-layer neural network with linear activation functions. By this interpretation, we propose a Neural Generalization of Multiple Kernel Learning (NGMKL), which extends the conventional MKL framework to a multi-layer neural network with nonlinear activation functions. Our experiments show that the proposed method, which has a higher complexity than traditional MKL methods, leads to higher recognition accuracy on several benchmarks.

## 1 Introduction

Kernel methods are a class of machine learning algorithms, which have been widely used in different types of problems [10, 19, 22, 36]. The performance of kernel methods is highly dependent on the type of kernel and the parameters of that kernel. This raises the challenge of

✉ Kamaledin Ghiasi-Shirazi
k.ghiasi@um.ac.ir

Ahmad Navid Ghanizadeh
ahgh00002@uni-saarland.de

Reza Monsefi
monsefi@um.ac.ir

Mohammadreza Qaraei
mohammadreza.mohammadniaqaraei@aalto.fi

1  Department of Computer Science, Saarland University, Saarbrücken, Germany

2  Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

3  Department of Computer Science, Aalto University, Helsinki, Finland

Springer

learning the kernel function. Multiple Kernel Learning (MKL), which learns the combination of a finite set of kernels, is a principled way to address this challenge [1, 4, 9, 17, 34]. In MKL algorithms, a conventional way for kernel learning is to combine the kernel functions linearly. More sophisticated ways for learning the kernel functions are to use nonlinear combinations [7, 35, 39, 40]. However, it is not clear that these methods can obtain better performance compared to a simple linear combination of kernels, and there is still room for improvement [4].

Similar to other kernel-based methods, two well-known properties of typical MKL algorithms are (i) these methods usually converge to a global solution since they can be formulated as a convex optimization problem, and (ii) kernel functions can be used in large-margin classifiers, for example, Support Vector Machines (SVMs), which reduces overfitting and may lead to better generalization. Although these features have been very appealing in the traditional view of machine learning, their importance has been challenged by the remarkable successes of deep learning methods.

Deep learning models pass data through several layers, which produces a rich representation of the data and provides a way to learn complex functions. There are remarkable differences between the framework of deep models and kernel methods. In contrast to kernel methods, deep learning models are not usually trained by a convex optimization problem. While it might be considered a drawback, as argued by [3], sacrificing convexity may be inevitable for learning complex functions. Furthermore, contrary to kernel methods that use hinge loss, which is margin-based, deep learning models use softmax with cross-entropy in the last layer for classification. However, [30] have shown that softmax with cross-entropy loss and hinge loss have very similar characteristics, and both entail a loss when the margin is not observed. Therefore, it can be said that deep learning methods that use softmax with cross-entropy are inherently margin-based. Finally, using stochastic gradient descent (SGD) with mini-batches for optimizing deep learning models provides a way for highly parallel computations via GPUs, which has led to state-of-the-art results in large-scale problems.

In recent years, finding the connection between kernel and deep learning methods has become an active research topic. These works have been done mainly with the purpose of boosting kernel methods [6, 21, 32] or achieving a better understanding of deep learning models [2, 8, 12]. In this paper, we investigate the connection between deep learning and a set of well-known algorithms in kernel methods, namely MKL, for improving kernel-based algorithms. We show that the conventional MKL algorithms can be interpreted as a one-layer neural network without nonlinear activation functions in which kernel values represent the input. From this point of view, we propose a Neural Generalization of Multiple Kernel Learning (NGMKL), which extends the shallow linear neural interpretation of MKL to a multilayer neural network with nonlinear activation functions. This practice leads to an MKL model with higher complexity, which improves the capacity of MKL models to learn more complex functions.

MKL models with deep structures have been investigated previously through Multilayer Multiple Kernel Learning (MLMKL) methods [13–15]. However, the framework of these methods still sticks to kernelized SVMs, which makes them significantly different from neural networks. More precisely, similar to ordinary MKL methods, MLMKL models use SVMs as classifiers, while neural networks utilize softmax for classification. Secondly, these models have some constraints on the parameters, such as the positivity of the combination weights to keep the deep kernel function positive definite. However, we note that the rationality behind imposing positive definiteness over kernel functions is to obtain a convex objective function, which MLMKL will not attain because of its deep structure. As a result, it may not be beneficial to constrain the kernel function to be a Mercer kernel in a deep structure.

Finally, in contrast to typical deep learning models, in MLMKL models, the weights are not usually trained simultaneously, and the training algorithm alternates between learning the kernel combination weights and the classifier's parameters.

In contrast to MKL and MLMKL algorithms, our proposed method utilizes a softmax classifier in the last layer. There are no constraints, such as the positivity of the combination weights, on the parameters of NGMKL, and all the parameters are trained simultaneously. A consequence of the neural interpretation of MKLs is that the framework of deep learning algorithms can be leveraged, which, for instance, provides fast parallel computations via GPUs. Moreover, in the context of SGD, mini-batching allows high throughput since it can process a large number of input examples in many cores at once in GPU. We show that NGMKL outperforms conventional MKL algorithms on commonplace datasets in kernel methods.

The rest of the paper is organized as follows. Section 2 reviews the related work on connecting kernel methods and deep learning. We also review the MLMKL models. In Sect. 3, we describe the conventional formulation of MKL algorithms. In Sect. 4, we show how MKL can be seen in the neural network framework and propose the NGMKL model. The comparison of NGMKL and ordinary MKL algorithms by experiments on several datasets are presented in Sect. 5. Finally, we conclude the paper in Sect. 6.

## 2 Related Work

Several works in the literature investigate the connections between kernel methods and neural networks. In this section, we review three types of these works: (i) Kernel functions that provably imitate the architecture of multilayer neural networks, (ii) Multilayer neural networks constructed by kernel approximation techniques, and (iii) MKL methods with multilayer structures.

Some methods link kernel methods with neural networks by proposing kernel functions that mimic the computation of neural networks. For example, similar to neural networks, it has been demonstrated that the feature map of arc-cosine kernels consists of a linear transformation using an infinite number of weights followed by a nonlinearity [6]. However, the interpreted weights are fixed and have a Gaussian distribution with zero mean and unit variance. To tackle this drawback, [24] proposed to learn the covariance of the interpreted weights by stretching a finite number of trained weights to an infinite number. [25] proposed learning the covariance by an iterative approach similar to restricted Boltzmann machines. In arc-cosine kernels, one can change the nonlinearity of the interpreted neural network and extend the structure to multiple layers by using two distinct hyperparameters. The connections between neural networks with an infinite number of weights and kernel methods are also discussed in the context of Gaussian processes, which has attracted attention for a better understanding of deep learning in recent years [11, 12, 18, 38].

While kernel functions are implicitly seen as an infinite-width neural network in the aforementioned methods, some methods use kernel approximation techniques [26, 37] to explicitly construct one layer of a neural network by a kernel function. For example, convolutional kernel networks that link convolutional neural networks with kernel methods are constructed by consecutively approximating a convolution kernel [20, 21, 23]. In another method called M-DKMO, a neural network is used upon the feature maps of some kernel functions approximated by the Nyström method for kernel learning in an end-to-end fashion [32]. Although this method uses neural networks for learning kernels, it differs from the proposed NGMKL

method. NGMKL reformulates the MKL problem through neural networks, while M-DKMO uses neural networks for learning approximated kernel feature maps.

More similar to our work, some methods proposed MKL problems in which the kernels are combined through a neural-network-like architecture. These methods, which are known as Multilayer Multiple Kernel Learning (MLMKL) models, were first introduced by [40]. In the method of [40], the kernel function is defined as a convex combination of a set of base kernels followed by an exponential nonlinearity, which can be seen as a two-layer MKL problem. [29] extended two-layer MLMKLs to several layers and optimized the combination parameters by backpropagation. [14] utilized unsupervised, semi-supervised, and Laplacian SVM as the objective functions for MLMKL. For reducing computations and improving the scalability of MLMKL, [15] proposed kernel approximation in a greedy layer-wise fashion. To make MLMKL more similar to neural networks, [31] formulated the base kernels as a neural operation and optimized the support vectors of MLMKLs along with the other parameters.

Although MLMKL models combine multiple kernels in a multilayer structure, NGMKL differs since the multilayer neural architecture in MLMKL is only for combining the base kernels. In other words, the kernel combination in MLMKL has a multilayer structure, while in our proposed method, the whole MKL problem, namely kernel combination and classification, is interpreted as a single neural network that can be extended to a deep structure. As a result, firstly, MLMKLs use SVMs for classification, while NGMKLs utilize softmax with cross-entropy, similar to ordinary neural networks. Secondly, MLMKL has some constraints on the parameters, whereas NGMKL is formulated as an unconstrained optimization problem. Finally, in most of the MLMKL models, the parameters for combining the kernels and the parameters of the classifier are learned alternatively. In contrast, in NGMKL, all the parameters are learned simultaneously.

## 3 Multiple Kernel Learning

In a typical kernel-based method, data is represented by a positive definite function called a kernel, which computes the similarity between data. However, a single kernel might not be rich enough to capture the complex underlying patterns of real-world data. In MKL algorithms, data is represented by several kernels, and the aim is to learn the optimal combination of these representations. Using multiple kernels instead of a single kernel increases the learning capacity of kernel-based models. In this section, we describe the conventional framework of MKL models.

Let $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ consist of $n$ pairs of feature vectors $x_i \in \mathbb{R}^d$ and their corresponding labels $y_i \in \{-1, +1\}$. Using a kernel $k(., .) \rightarrow R$, each input x is represented by the vector $k^x = [k(x, x_1), \ldots, k(x, x_n)]^T$. Then a linear function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ parameterized by $\gamma \in \mathbb{R}^n$ could be used on top of $k^x$ to predict the label of x:

$$f(k^x) = \gamma^T k^x \tag{1}$$

By using $s$ different base kernels $k_1, \ldots, k_s$, the data can be represented by vectors $k_1^x, \ldots, k_s^x$ corresponding to these kernels. If these vectors are passed through the function $f$, the final classifier $f_c : \mathbb{R}^n \rightarrow \{-1, 1\}$ is formed by combining the output of these functions:

$$f_c(x) = \text{sign}(\sum_{j=1}^{s} \beta_j f(k_j^x)), \tag{2}$$

where $\beta_j$ can be interpreted as the weight of the kernel $k_j$. In support vector machines, which seek to find a hyperplane with the maximum margin, the MKL problem can be formulated as the following optimization problem [34]:

$$\max_{\beta} \min_{\alpha} \quad \frac{1}{2} \sum_{j=1}^{s} \beta_j (\alpha \odot y)^T G_j (\alpha \odot y) - \alpha^T \mathbf{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} \alpha_i y_i = 0,$$

$$\sum_{j=1}^{S} \beta_j = 0, \tag{3}$$

$$0 \le \alpha \le C,$$

$$\beta_j \ge 0, \quad \forall j = 1, \ldots, s$$

where $G_j = [k_j^{x_1}, \ldots, k_j^{x_n}]$, $y = [y_1, \ldots, y_n]^T$, $\alpha \in \mathbb{R}^n$, $\mathbf{1}$ is a vector of ones, and $C$ is the hyperparameter of the SVM. The decision function obtained by solving Eq. (3) is similar to Eq. (2) in which $\gamma = \alpha \odot y$.

## 4 Proposed Model

In this section, we propose a formulation of the MKL problem as a neural network. The core element of a neural network is a basic module in which a linear operation followed by a nonlinear one are applied to the input:

$$\hat{u} = \phi(\Gamma^T u) \tag{4}$$

where $u \in \mathbb{R}^n$ is a feature vector, $\Gamma \in \mathbb{R}^{n \times n'}$ is a matrix of parameters, $\phi(.)$ is a nonlinear activation function, and $\hat{u} \in \mathbb{R}^{n'}$ is the output of the module.

In MKL, multiplication of each kernelized representation of data, $k^x$, with the parameter vector $\gamma \in \mathbb{R}^n$ (Eq. (1)) can be seen as a basic module in a neural network but without nonlinearity. By applying a nonlinear function $\phi(.)$, Eq. (1) becomes equivalent to this module in a neural network:

$$z = \phi(\gamma^T k^x) \tag{5}$$

Having multiple kernel representations $k_1^x, \ldots, k_s^x$ for data x obtained by different kernels, by Eq. (5), we have parameters $\gamma_1, \ldots, \gamma_s$ and representations $z_1^1, \ldots, z_s^1$ corresponding to these kernels. We concatenate $z_1^1, \ldots, z_s^1$ to form $z^1$ as follows:

$$z^1 = [z_1^1, \ldots, z_s^1]^T \tag{6}$$

Then we apply the basic module of neural networks to $z^1$ repeatedly as follows:

$$z^l = \phi_l((W^{l-1})^T \ldots \phi_1((W^1)^T z^1) \ldots), \tag{7}$$

where $W^1 \in \mathbb{R}^{s \times n_1}$ and $W^i \in \mathbb{R}^{n_{i-1} \times n_i}$ for $i > 1$ are weight matrices, and $\phi_i(.)$ is an activation function. The above equation generalizes the MKL problem to a deep neural network, which we call the Neural Generalization of Multiple Kernel Learning (NGMKL). In particular, in a binary classification problem, by setting $l = 1$ and setting the same parameters for $\gamma_1, \ldots, \gamma_s$, and using a linear activation function as $\phi_1$, the structure of MKL
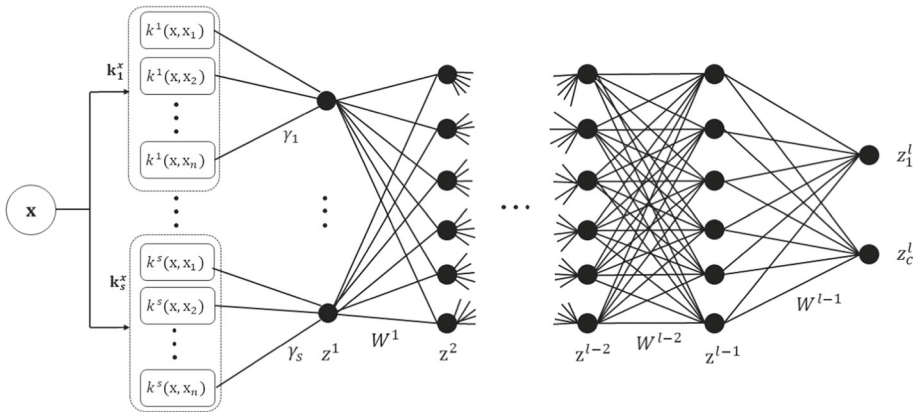
**Fig. 1** Architecture of the proposed network with $L$ layers

problem of Eq. (7) becomes equivalent to the conventional MKL framework. The architecture of NGMKL is shown in Fig. 1.

While conventional MKL frameworks use SVM for classification, in NGMKL, we use softmax for classification. Let $z^l \in \mathbb{R}^c$, where $c$ is the number of classes. The softmax classifier assigns $c$ probabilities to the data as follows:

$$\hat{y}_i = \frac{e^{z_i^l}}{\sum_{j=1}^{c} e^{z_j^l}}, \tag{8}$$

and the label of the data is defined as $k = \underset{i}{\arg\max} \, \hat{y}_i$.

For training NGMKL, we use the cross-entropy loss function. Assume $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ consists of $n$ pairs of data $x_i \in \mathbb{R}^d$ and the corresponding labels $y_i \in \{0, 1\}^c$, where $y_{ik} = 1$ if $k$ is the relevant class for the sample $x_i$, and $y_{ik} = 0$ otherwise. The cross-entropy loss function for a sample $(x, y)$, in which $y_k = 1$ and $y_{i \neq k} = 0$, is defined as follows:

$$J = -z_k^l + \log \left( \sum_{j=1}^{c} e^{z_j^l} \right) \tag{9}$$

We train all the parameters of NGMKL, including $W^1, \ldots, W^l$ and $\gamma_1, \ldots, \gamma_s$, simultaneously, using the backpropagation algorithm. The derivatives of the cross-entropy function with respect to the parameters of a two-layer NGMKL are as follows:

$$\frac{\partial J}{\partial w_{ij}^1} = (\hat{y}_j - y_j)\phi_1(z_i^1) \tag{10}$$

$$\frac{\partial J}{\partial \gamma_i} = w_{i*}^1 (y - \hat{y})\phi_1'(z_i^1) k_x^i \tag{11}$$

where $w_{i*}^1$ is the $i$-th row of $W^1$ and $\hat{y} = [\hat{y}_1, \ldots, \hat{y}_c]^T$.

As seen in Eq. (3), the SVM framework of MKL has several constraints on the parameters. These constraints are for preserving the convexity of the problem and ensuring a large margin. However, in our proposed model, there is no constraint on the parameters, and the goal of large-margin classification is obtained by a combination of softmax with cross-entropy loss

**Table 1** Summary of the datasets used in the experiments from the UCI repository

| Dataset | # Training set | # Test set | Dimensions |
|---|---|---|---|
| a1a | 802 | 803 | 123 |
| a3a | 1592 | 1593 | 123 |
| a4a | 2391 | 2392 | 123 |
| german.numer | 500 | 500 | 24 |
| ionosphere | 175 | 176 | 34 |
| monks1 | 556 | 557 | 6 |
| sonar | 104 | 104 | 60 |
| svmguide3 | 621 | 622 | 21 |
| banana | 400 | 4900 | 2 |
| b.cancer | 200 | 77 | 9 |
| diabetes | 468 | 300 | 8 |
| german | 700 | 300 | 20 |
| image | 1300 | 1010 | 18 |
| ringnorm | 400 | 7000 | 20 |
| f.solar | 666 | 400 | 9 |
| thyroid | 140 | 75 | 5 |
| titanic | 150 | 2051 | 3 |
| twonorm | 400 | 7000 | 20 |
| waveform | 400 | 4600 | 21 |

and weight decay (which penalizes large weights). In Sect. 5, we show that although NGMKL is formulated as a non-convex optimization problem, it can outperform the typical MKL models.

## 5 Experimental Results

In this section, we evaluate NGMKL in several classification problems and compare it with conventional MKL models. We use a two-layer NGMKL and three variants of this model, namely "NGMKL1", "NGMKL2", and "NGMKL3", which are different from each other in the choice of the input kernel. In the following subsections, we describe the datasets we used for the evaluation of the models, the baseline models with which we compare NGMKL, the architecture and the optimization settings of NGMKL, and the three variants of NGMKL which are different in the choice of the input kernel.

### 5.1 Datasets

Since the conventional MKL models are not scalable, we use UCI datasets to evaluate the models. These datasets are described in Table 1. Following [28], each dataset is randomly split into the training and testing data, and this practice is repeated 100 times, except for the image dataset, which has 20 splits.

## 5.2 Baselines

We compare NGMKL with some well-known MKL algorithms:

- MKL-SD: an $\ell_1$-norm regularized MKL solved by subgradient descent methods [27].
- MKL-SILP: an $\ell_1$-norm regularized MKL with a semi-infinite linear programming objective function [33].
- MKL-Hessian: an MKL algorithm solved by second-order optimization methods [5].
- $\ell_p$-MKL: an $\ell_p$-norm regularized MKL in which we set $p = 2$ [16].
- MKBoost: A boosting approach to the MKL problem [39].

## 5.3 NGMKL Setup

We use a two-layer NGMKL in all the experiments. The nonlinearity of the hidden layer is the scaled hyperbolic tangent $f(x) = 1.7159 \tanh(\frac{2}{3}x)$. We use the Stochastic Gradient Descent (SGD) algorithm to train our model in which the batch size is 40, the learning rate is 0.01, and the weight decay is $5 \times 0^{-6}$.

## 5.4 Three Variants of NGMKL

We use three variants of NGMKL named "NGMKL1", "NGMKL2", and "NGMKL3" regarding the methods used for selecting the input kernels from the base kernels. There are 17 base kernels in all the models, including NGMKLs and the baselines: three polynomial kernels ($k(x, y) = (x^T y)^d$) of degrees 1, 2, and 3 as well as 14 Gaussian kernels ($k(x, y) = e^{\frac{-1}{2\sigma^2} \|x-y\|_2^2}$) with kernel widths $\sigma = 2^{-6}, \ldots, 2^7$. The three variants of NGMKL are as follows:

- NGMKL1: All 17 base kernels are used as the input kernels in this model.
- NGMKL2: In this model, $\ell_1$-MKL [33] is used for selecting the input kernels from the base kernels. $\ell_1$-MKL produces sparse weights for combining the base kernels by penalizing the $\ell_1$ norm of the weights.
- NGMKL3: MKBoost-D1 [39] is used for selecting the input kernels among the 17 base kernels. MKBoost-D1, in each trial of the boosting method, selects the kernel with the minimum error and adds it to the set of selected kernels. The selection is with replacement.

## 5.5 Comparison Results

A comparison of three variants of NGMKLs is shown in Table 2. This table compares three variants of NGMKLs, showing that all NGMKL methods generally achieve similar results. However, NGMKL3 achieves better results compared to the other variants of NGMKLs in 9 datasets. The second successful variant is NGMKL2, achieving better results in 6 datasets, and NGMKL1 has a better error rate in the remaining datasets.

Table 3 compares popular MKL algorithms with NGMKL3. Comparing NGMKL3 and the best competing method for each row, we have highlighted statically significant results, using an unpaired $t$ test, in boldface. Results for "monks" and "sonar" datasets are not statistically significant. Overall for 17 datasets, the differences between NGMKL3 and the best other method are statistically significant.

**Table 2** The results of our three proposed methods [mean classification error ± std] in percent on UCI datasets. The best performance in each dataset is highlighted in **bold**

| Dataset | NGMKL1 | NGMKL2 | NGMKL3 |
|---|---|---|---|
| a1a | 18.04 ± 0.00 | **15.94 ± 0.00** | 17.39 ± 0.01 |
| a3a | 17.93 ± 0.00 | 17.56 ± 0.00 | **17.08 ± 0.01** |
| a4a | 18.22 ± 0.00 | 19.01 ± 0.00 | **17.32 ± 0.01** |
| german.numer | **24.57 ± 0.01** | 25.03 ± 0.01 | 25.79 ± 0.02 |
| ionospher | 06.99 ± 0.00 | 05.63 ± 0.00 | **05.09 ± 0.30** |
| monks1 | **04.87 ± 0.09** | 05.16 ± 0.00 | 04.91 ± 0.03 |
| sonar | 21.41 ± 0.08 | 20.78 ± 0.05 | **17.93 ± 0.23** |
| svmguide3 | 18.45 ± 0.00 | **17.73 ± 0.00** | 18.08 ± 0.02 |
| banana | 12.00 ± 0.00 | **11.00 ± 0.33** | 11.06 ± 0.00 |
| b.cancer | 26.49 ± 0.30 | 27.86 ± 0.17 | **26.04 ± 0.22** |
| diabete | 24.40 ± 0.02 | 26.40 ± .0.05 | **24.29 ± 0.03** |
| german | 24.19 ± 0.05 | **23.83 ± 0.05** | 23.91 ± 0.05 |
| image | 03.55 ± 0.00 | 04.06 ± 0.00 | **03.14 ± 0.00** |
| ringnorm | 02.07 ± 0.00 | 01.96 ± 0.00 | **01.57 ± 0.00** |
| f.solar | 35.80 ± 0.04 | **35.57 ± 0.07** | 35.79 ± 0.03 |
| thyroid | 05.53 ± 0.10 | 05.80 ± 0.05 | **04.17 ± 0.06** |
| titanic | 22.34 ± 0.00 | **22.12 ± 0.00** | 22.41 ± 0.01 |
| twonorm | **02.80 ± 0.00** | 03.05 ± 0.00 | 02.90 ± 0.00 |
| waveform | **09.37 ± 0.01** | 10.68 ± 0.04 | 09.55 ± 0.00 |

According to Table 3, all regular MKL algorithms have the same error rate because they have the same formulation but adopt different strategies to speed up the optimization process. However, NGMKL3 is superior to the traditional MKL methods in most datasets (12 out of 17).

In the next step, we evaluate NGMKL3 with different numbers of layers. For simplicity, we only compare with three different layers (L=2,3,4). When we have a higher number of layers, most of the datasets remain unchanged in terms of the error rate (16 out of 19), and only three of them (namely a1a, a3a, and a4a) have a decrease in the classification error rate (Fig. 2). As seen from Table 1, these three datasets have more dimensions than others. Since our proposed method (NGMKL3) can learn complex functions by extending the layers with nonlinear activation functions, the classification error on these three datasets has decreased. However, extending the layers cannot improve accuracy in most datasets, especially when using UCI datasets, since most of them are not complex enough. In addition, the three variants of our proposed methods are not sensitive to the number of layers; therefore, we evaluated our proposed methods in comparison with conventional methods based on two layers.

In Fig. 3, we compare the performance of NGMKL3 with two Artificial Neural Networks (ANN) with two layers. The architecture of ANN, including the layers, neurons within each layer, and the activation functions, is the same as that of NGMKL3. In order to configure and train the ANN, two approaches can be considered. One approach is to use initial weights from MKL and freeze them during training (ANN1), and the second is to use random initial weights and train the whole neural network (ANN2). The reason behind the first approach is to demonstrate the importance of the kernel layer and provide a fair comparison of MKL

**Table 3** The average testing errors [mean classification error ± std] in percent of five popular MKL Algorithms and one of the proposed methods (NGMKL3). The relative ranking of each algorithm on each data set is shown in the parenthesis. Rank 1 refers to the algorithm which achieves the highest accuracy on the given data set

| Dataset | MKL-SD | MKL-SILP | MKL–Hessian | MKboost | $\ell_p$-MKL | NGMKL3 |
|---|---|---|---|---|---|---|
| a1a | 24.53 ± 0.09(4) | 24.53 ± 0.09(4) | 24.56 ± 0.08(5) | 18.78 ± 0.77(2) | 23.93 ± 0.66(3) | **17.39 ± 0.01(1)** |
| a3a | 24.24 ± 0.03(4) | 24.24 ± 0.03(4) | 24.55 ± 0.00(5) | 18.63 ± 0.51(2) | 24.03 ± 0.40(3) | **17.08 ± 0.01(1)** |
| a4a | 24.85 ± 0.01(4) | 24.85 ± 0.01(4) | 24.85 ± 0.01(4) | 18.90 ± 0.52(2) | 24.39 ± 0.33(3) | **17.32 ± 0.01(1)** |
| german.numer | 29.99 ± 0.04(5) | 29.97 ± 0.25(4) | 30.00 ± 0.06(6) | 26.39 ± 1.17(2) | 29.35 ± 0.55(3) | **25.79 ± 0.02(1)** |
| ionospher | 06.94 ± 1.55(5) | 07.51 ± 2.55(6) | 06.71 ± 1.77(4) | 05.74 ± 1.39(3) | 05.63 ± 2.00(2) | **05.09 ± 0.30(1)** |
| monks | 20.90 ± 2.51(5) | 20.68 ± 3.11(4) | 17.41 ± 2.66(3) | 04.78 ± 1.40(1) | 21.56 ± 0.00(6) | 04.91 ± 0.03(2) |
| sonar | 22.48 ± 4.23(5) | 22.09 ± 4.27(4) | 24.13 ± 3.65(6) | 18.17 ± 3.96(2) | 19.76 ± 4.06(3) | 17.93 ± 0.23(1) |
| svmguide3 | 22.56 ± 0.38(4) | 22.58 ± 0.40(5) | 22.83 ± 0.00(6) | 18.42 ± 0.67(2) | 21.84 ± 0.57(3) | **18.08 ± 0.02(1)** |
| banana | 10.71 ± 0.00(2) | **10.38 ± 0.00(1)** | 10.86 ± 0.00(3) | 12.12 ± 0.00(6) | 11.95 ± 0.00(5) | 11.06 ± 0.00(4) |
| b.cancer | 28.38 ± 0.25(4) | 28.31 ± 0.26(3) | 27.53 ± 0.22(2) | 31.45 ± 0.18(5) | 33.51 ± 0.11(6) | **26.04 ± 0.22(1)** |
| diabetes | 24.68 ± 0.07(3) | 24.88 ± 0.06(4) | **23.60 ± 0.02(1)** | 28.65 ± 0.12(6) | 24.93 ± 0.05(5) | 24.29 ± 0.03(2) |
| german | 27.83 ± 0.12(4) | 27.42 ± 0.11(3) | 23.93 ± 0.04(2) | 28.09 ± 0.06(5) | 30.90 ± 0.08(6) | **23.91 ± 0.05(1)** |
| image | 05.34 ± 0.00(5) | 05.34 ± 0.00(5) | **02.05 ± 0.00(1)** | 02.87 ± 0.00(2) | 03.68 ± 0.00(4) | 03.14 ± 0.00(3) |
| ringnorm | 02.11 ± 0.00(3) | 02.13 ± 0.00(4) | 01.71 ± 0.00(2) | 05.16 ± 0.00(6) | 03.037 ± 0.00(5) | **01.57 ± 0.00(1)** |
| f.solar | 32.45 ± 0.02(2) | 32.45 ± 0.02(2) | 35.38 ± 0.02(3) | 36.49 ± 0.03(5) | **32.42 ± 0.03(1)** | 35.79 ± 0.03(4) |
| thyroid | 04.21 ± 0.04(3) | 04.47 ± 0.04(4) | 04.20 ± 0.04(2) | 04.71 ± 0.05(5) | 05.73 ± 0.11(6) | **04.17 ± 0.06(1)** |
| titanic | 22.32 ± 0.00(4) | 22.27 ± 0.00(3) | 22.17 ± 0.00(2) | 22.45 ± 0.03(6 | **22.09 ± 0.00(1)** | 22.41 ± 0.01(5) |
| twonorm | 02.99 ± 0.00(2) | 03.02 ± 0.00(3) | 03.36 ± 0.00(4) | 03.56 ± 0.00(5) | 03.90 ± 0.00(6) | **02.90 ± 0.00(1)** |
| waveform | 11.84 ± 0.00(3) | 11.89 ± 0.00(4) | 11.00 ± 0.00(2) | 12.13 ± 0.00(5) | 14.15 ± 0.00(6) | **09.55 ± 0.00(1)** |
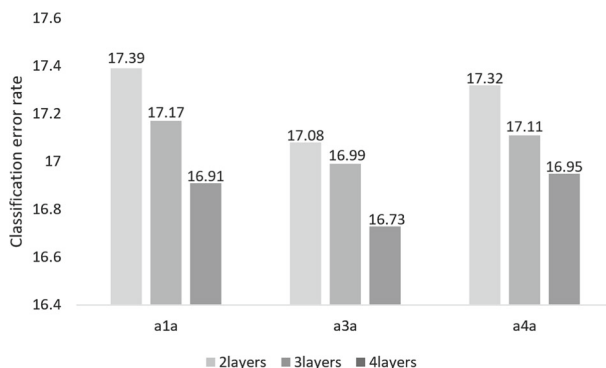
**Fig. 2** Evaluating the performance of NGMKL3 with different numbers of layers



**Fig. 3** Evaluating the performance of NGMKL3 with two variants of Artificial Neural Network (ANN) with two layers

methods with neural networks. As seen in Fig. 3, in all datasets, ANN1 and NGMKL3 have almost the same classification error rate, but ANN2 performs worse than both approaches in terms of recognition accuracy.

In the next experiment, we compare the training classification error of NGMKL3 and ANN2. According to our results, a lower number of iterations is required for the convergence of NGMKL3 compared to ANN2. In other words, by transforming data from the kernel layer, a rich representation of data is formed, allowing us to capture underlying patterns with fewer training iterations. In Fig. 4, we depict the average training error based on the iteration of ANN2 and NGMKL3 for "b.cancer" and "waveform" datasets. In the "b.cancer" dataset, for instance, NGMKL3 achieves a near-zero training error after 300 iterations, while ANN2 reaches this level after 600 iterations.
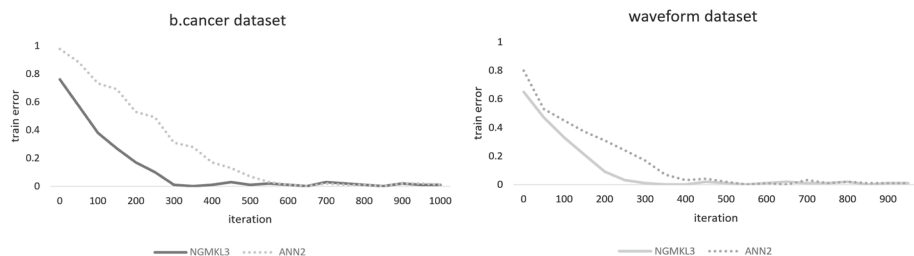
**Fig. 4** The number of iterations to train NGMKL3 and ANN2 on b.cancer (left subfigure) and waveform (right subfigure) datasets

## 6 Conclusion

In this paper, we showed that the framework of the MKL could be interpreted as a neural network with one hidden layer in which the input data is represented by a set of kernels. We proposed NGMKL, a neural generalization of the MKL problem, using this interpretation. Like conventional neural networks, and unlike typical MKL algorithms, NGMKL employs the softmax layer for classification. Furthermore, there are no constraints on the parameters, and all the parameters are trained simultaneously. Moreover, It leverages nonlinear transformations, and the architecture can be extended to multiple layers. We evaluated NGMKL on the UCI datasets and compared it with typical MKL algorithms. The experimental results show that the proposed model can boost the performance of MKL models.

## References

1. Bach FR, Lanckriet GR, Jordan MI (2004) Multiple kernel learning, conic duality, and the smo algorithm. In: Proceedings of the twenty-first international conference on Machine learning, p 6
2. Belkin M, Ma S, Mandal S (2018) To understand deep learning we need to understand kernel learning. arXiv preprint arXiv:1802.01396
3. Bengio Y, LeCun Y et al (2007) Scaling learning algorithms towards AI. Large-scale kernel Mach 34(5):1–41
4. Bucak SS, Jin R, Jain AK (2013) Multiple kernel learning for visual object recognition: a review. IEEE Trans Pattern Anal Mach Intell 36(7):1354–1369
5. Chapelle O, Rakotomamonjy A (2008) Second order optimization of kernel parameters. In: Proceedings of the NIPS Workshop on Kernel Learning: Automatic Selection of Optimal Kernels, vol 19, p 87
6. Cho Y, Saul LK (2010) Large-margin classification in infinite neural networks. Neural Comput 22(10):2678–2697
7. Cortes C, Mohri M, Rostamizadeh A (2009) Learning non-linear combinations of kernels. Adv Neural Inf Process Syst 396–404
8. Daniely A, Frostig R, Singer Y (2016) Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. Adv Neural Inf Process Syst 2253–2261
9. Gönen M, Alpaydın E (2011) Multiple kernel learning algorithms. J Mach Learn Res 12:2211–2268

10. Grauman K, Darrell T (2005) The pyramid match kernel: Discriminative classification with sets of image features. In: Tenth IEEE International conference on computer vision (ICCV'05) Volume 1, IEEE, vol 2, pp 1458–1465
11. Hazan T, Jaakkola T (2015) Steps toward deep kernel methods from infinite neural networks. arXiv preprint arXiv:1508.05133
12. Jacot A, Gabriel F, Hongler C (2018) Neural tangent kernel: convergence and generalization in neural networks. Adv Neural Inf Process Syst 8571–8580
13. Jiu M, Sahbi H (2016) Deep kernel map networks for image annotation. In: 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP), IEEE, pp 1571–1575
14. Jiu M, Sahbi H (2017) Nonlinear deep kernel learning for image annotation. IEEE Trans Image Process 26(4):1820–1832
15. Jiu M, Sahbi H (2019) Deep representation design from deep kernel networks. Pattern Recogn 88:447–457
16. Kloft M, Brefeld U, Sonnenburg S, Zien A (2011) LP-norm multiple kernel learning. J Mach Learn Res 12:953–997
17. Lanckriet GR, Cristianini N, Bartlett P, Ghaoui LE, Jordan MI (2004) Learning the kernel matrix with semidefinite programming. J Mach Learn Res 5:27–72
18. Lee J, Bahri Y, Novak R, Schoenholz SS, Pennington J, Sohl-Dickstein J (2017) Deep neural networks as gaussian processes. arXiv preprint arXiv:1711.00165
19. Longworth C, Gales MJ (2009) Combining derivative and parametric kernels for speaker verification. IEEE Trans Audio Speech Lang Process 17(4):748–757
20. Mairal J (2016) End-to-end kernel learning with supervised convolutional kernel networks. Adv Neural Inf Process Syst 1399–1407
21. Mairal J, Koniusz P, Harchaoui Z, Schmid C (2014) Convolutional kernel networks. Adv Neural Inf Process Syst 2627–2635
22. Miwa M, Sætre R, Miyao Y, Tsujii J (2009) Protein-protein interaction extraction by leveraging multiple kernels and parsers. Int J Med Informat 78(12):e39–e46
23. Mohammadnia-Qaraei MR, Monsefi R, Ghiasi-Shirazi K (2018) Convolutional kernel networks based on a convex combination of cosine kernels. Pattern Recogn Lett 116:127–134
24. Pandey G, Dukkipati A (2014a) Learning by stretching deep networks. In: International conference on machine learning 1719–1727
25. Pandey G, Dukkipati A (2014b) To go deep or wide in learning? In: Artificial Intelligence and Statistics, PMLR, pp 724–732
26. Rahimi A, Recht B (2008) Random features for large-scale kernel machines. Adv Neural Inf Process Syst 1177–1184
27. Rakotomamonjy A, Bach FR, Canu S, Grandvalet Y (2008) Simplemkl. J Mach Learn Res 9:2491–2521
28. Rätsch G, Onoda T, Müller KR (2001) Soft margins for adaboost. Mach Learn 42(3):287–320
29. Rebai I, BenAyed Y, Mahdi W (2016) Deep multilayer multiple kernel learning. Neural Comput Appl 27(8):2305–2314
30. Rosasco L, Vito ED, Caponnetto A, Piana M, Verri A (2004) Are loss functions all the same? Neural Comput 16(5):1063–1076
31. Sahbi H (2019) Totally deep support vector machines. arXiv preprint arXiv:1912.05864
32. Song H, Thiagarajan JJ, Sattigeri P, Spanias A (2018) Optimizing kernel machines using deep learning. IEEE Trans Neural Netw Learn Syst 29(11):5528–5540
33. Sonnenburg S, Rätsch G, Schäfer C (2006a) A general and efficient multiple kernel learning algorithm. Adv Neural Inf Process Syst 1273–1280
34. Sonnenburg S, Rätsch G, Schäfer C, Schölkopf B (2006) Large scale multiple kernel learning. J Mach Learn Res 7:1531–1565
35. Varma M, Babu BR (2009) More generality in efficient multiple kernel learning. In: Proceedings of the 26th annual international conference on machine learning, pp 1065–1072
36. Vedaldi A, Gulshan V, Varma M, Zisserman A (2009) Multiple kernels for object detection. In: 2009 IEEE 12th international conference on computer vision, IEEE, pp 606–613
37. Williams CK, Seeger M (2001) Using the nyström method to speed up kernel machines. Adv Neural Inf Process Syst 682–688
38. Wilson AG, Hu Z, Salakhutdinov R, Xing EP (2016) Deep kernel learning. In: Artificial intelligence and statistics, pp 370–378
39. Xia H, Hoi SC (2012) Mkboost: a framework of multiple kernel boosting. IEEE Trans Knowl Data Eng 25(7):1574–1586
40. Zhuang J, Tsang IW, Hoi SC (2011) Two-layer multiple kernel learning. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics, pp 909–917