Deep Kernel Learning via Random Fourier Features

Jiaxuan Xie, Fanghui Liu, Kaijie Wang, Xiaolin Huang

Department of Automation
Shanghai Jiao Tong University
Shanghai, China 200240

{knxie,lfhsgre}@outlook.com, {xiaolinhuang,kaijie_wang}@sjtu.edu.cn

Abstract

Kernel learning methods are among the most effective learning methods and have been vigorously studied in the past decades. However, when tackling with complicated tasks, classical kernel methods are not flexible or “rich” enough to describe the data and hence could not yield satisfactory performance. In this paper, via Random Fourier Features (RFF), we successfully incorporate the deep architecture into kernel learning, which significantly boosts the flexibility and richness of kernel machines while keeps kernels’ advantage of pairwise handling small data. With RFF, we could establish a deep structure and make every kernel in RFF layers could be trained end-to-end. Since RFF with different distributions could represent different kernels, our model has the capability of finding suitable kernels for each layer, which is much more flexible than traditional kernel-based methods where the kernel is pre-selected. This fact also helps yield a more sophisticated kernel cascade connection in the architecture. On small datasets (less than 1000 samples), for which deep learning is generally not suitable due to overfitting, our method achieves superior performance compared to advanced kernel methods. On large-scale datasets, including non-image and image classification tasks, our method also has competitive performance.

Introduction

Vast quantities of accessible and available labeled data have engendered amazing breakthroughs of deep learning methods on various tasks, such as image classification, speech recognition, machine translation and object tracking. However, when tackling problems where labeled data are insufficient, existing deep approaches could not yield satisfactory performance due to severe overfitting. To overcome this troublesome issue, like meta-learning and one-shot learning, there have been some useful learning methods proposed.

Another considerable category, kernel methods, as an important machine learning tool with solid theoretical analysis, are fully developed to cope with small data problems and have been successfully applied in many fields. Nevertheless, the shallow structure limits the learning capability of kernel methods. Besides, standard kernel methods hypothesize the user-defined kernel effective enough to describe complex non-linear relationships, which is a weakness if we do not know a good data representation in advance.

Motivated by recent progress of deep learning, people realize that making kernel deeper could be a promising way to improve the flexibility of kernel methods (Cho and Saul 2009). (Wilson et al. 2016a). But there are two challenges for constructing a deep kernel structure. First, a layer that calculates a kernel will map inputs in \( \mathbb{R}^d \) into \( \mathbb{R}^{m \times m} \), where \( m \) is the number of samples. Pairwise considering samples is a merit of kernel methods but when \( m \) is large, it becomes challenging to calculate on a big matrix, not to mention to train those matrices in multi-layers. Second, it is difficult to train the kernel used in each layer. In fact, subject to the traditional rule of kernel methods, kernels utilized in each layer are still pre-given and characterized by one or several parameters. Thus, even in shallow model, it is more prevalent to tune the kernel parameters rather than to train them. Let alone the choice of different kernels, which is usually based on model selection methods but it is not suitable for dealing with the case of multiple layers.

To bridge the first challenge, we turn to the well-known kernel trick, i.e., for a kernel \( k \) that satisfies the Mercer’s condition, there is

\[
k(u, v) = \langle \phi(u), \phi(v) \rangle
\]

where \( \phi: \mathcal{X} \rightarrow \mathcal{H} \) is a feature map which projects samples of input space \( \mathcal{X} \) into a reproducing kernel Hilbert space \( \mathcal{H} \). By applying kernel trick, one can successfully control the computation complexity in deep kernel structure. Meanwhile, the composition of nonlinear mapping essentially could yield a more sophisticated and complex kernel as the composition of nonlinear kernels, as illustrated by (Cho and Saul 2009). Take the decompositions of two RBF kernels as an example, the composition becomes:

\[
k^{(2)}(u, v) = \phi^{(2)}(\phi^{(1)}(u)) \cdot \phi^{(2)}(\phi^{(1)}(v)) = e^{-2\lambda} \exp(-2\lambda k(u, v)).
\]

Although the feature mappings \( \phi^{(1)}, \phi^{(2)} \) that corresponding to the features of the inner and outer kernels respectively cannot be explicitly given, the above formulation, via learning feature expressions, shows the possibility of learning approaches composed of multiple kernel layers.

This idea of using features rather than kernels also appears in large-scale kernel learning, where the calculation is kept...
RFFNet is able to learn from both image and non-image data. For different kernels, the distribution could be improved from \( \frac{1}{100\%} \) to \( \frac{1}{5\%} \) on image data “CIFAR-10”, \( \frac{1}{17\%} \) on “monks1” and definiteness condition. For the RBF kernel (Vempati et al. 2010), the Laplacian kernel, and the Cauchy kernel, satisfy the shift-invariant and definiteness condition. For the RBF kernel \( k(u - v) = e^{-||u-v||^2/2} \), the distribution could be explicitly given.

**Random Fourier Features**

Let us first introduce the Random Fourier Feature method (RFF), the foundation of the proposed model. RFF originates from harmonic analysis (Rahimi and Recht 2008) and has become a powerful tool to approximate the feature mapping of a kernel. Here, the kernel should satisfy the following two conditions: i) the kernel should be shift-invariant, i.e., \( k(u, v) = k(u - v) \); ii) the kernel should be positive semi-definite. The two conditions together guarantee that \( k(z) \), where \( z = u - v \), is the Fourier transform of a measure. The Bochner’s theorem provides the key insight behind this transformation:

**Theorem 1 (Bochner (Rudin 1962)).** A continuous kernel \( k(u, v) = k(u - v) \) on \( \mathbb{R}^d \) is positive definite if and only if \( k(\delta) \) is the Fourier transform of a non-negative measure.

Accordingly, with \( \zeta(\omega)(u) = e^{j\omega^\top u} \), \( k(u - v) \) can be regarded as the following expectation over a distribution \( p \), i.e.,

\[
k(u - v) = \int_{\mathbb{R}^d} p(\omega)e^{j\omega^\top(u-v)}d\omega
\]

\[
= E_{\omega}(\zeta(\omega)(u)\zeta(\omega)(v))^*,
\]

In other words, when \( \omega \) is drawn from \( p \), \( \zeta(\omega)(u)\zeta(\omega)(v)^* \) is an unbiased estimate of \( k(u - v) \). If both the probability \( p(\omega) \) and \( k(u - v) \) are real, \( \psi(\omega)(u) = \cos(\omega^\top u) \sin(\omega^\top u)^\top \) gives a mapping so that \( E[\psi(\omega)(u)^\top \psi(\omega)(v)] = k(u, v) \).

**Related advanced kernel learning methods:** Since the proposal of deep neural networks, people have tried to enhance the flexibility of kernel methods in order to deal with complicated data. With deeper architectures, deep kernel learning (DKL) is expected to have better flexibility and the richness of representations is expected to get dramatic boosting from shallow kernel-based methods. In (Cho and Saul 2009), a new family of arc-cosine kernels are proposed via the integral representation to construct a multilayer kernel machine. However, this methodology is present for the specialized kernel and could not be applied to other common kernels. There also exist some works that are related to the Bayesian methodology. For instance, Smola et al. (Schölkopf and Smola 2001) combine Bayesian estimation framework with kernel methods and Wilson et al. (Wilson et al. 2016b) try to learn kernels through the marginal likelihood of a Gaussian process, but these methods usually require an extra feature extraction module such as the MLP for vectors or the deep network for images.
The data points are multiplied by corresponding ω, mapped by $\cos(\cdot)$ and $\sin(\cdot)$ respectively and then be concatenated in the concentrating block and become the inputs of the next layer (we call the parts between the two imaginary lines a RFF module). The final layer is a fully connected layer which produces the corresponding output such as the labels in classification tasks. With supervision, the kernels in different layers are learned by back propagation. Note the dimension $D$ of $\omega$ in each RFF module could be different.

For speeding up kernel calculation, the flowchart is to first choose an appropriate kernel and then apply RFF to approximate it. In this paper, we take the advantage of RFF for approaching kernels to incorporate deep architecture into the kernel learning. In the proposed structure, the distribution of $\omega$ in each layer could be trained and it is expected that via training the distribution we can find suitable kernel for each layer, which is at least theoretically feasible since RFF covers many kernels.

### Random Fourier features neural networks

**Structure**

With random Fourier Features, we can approximate kernels with feature mappings determined by trainable distributions. Therefore, we now could realize the deep kernel structure. Specifically, our deep kernel learning framework via random Fourier features is demonstrated in Fig. 1 and called random Fourier features neural networks (RFFNet). In RFFNet, there are $l$ layers, each of which consists of a RFF module and a concentrating block. A RFF module is the key part for producing features, including linear transformation, $\cos(\cdot)$ and $\sin(\cdot)$ mappings. The last layer is usually a fully connected layer which carries out the linear transform and yields the corresponding output such as the probabilities of the labels in classification tasks.

In RFF module, the input first goes through a linear transformation, where the parameters $\omega_i = [\omega_i^1, \ldots, \omega_i^D]$ are corresponding to a distribution related to a kernel and the important feature part in RFF is expressed as $f = \omega_i^\top x$. Here, for vectors, this operation could be executed by a full-connection layer without bias. For image data, considering a 3-channel image $X$ of size $N_h \times N_w \times 3$ and let $x \in \mathbb{R}^N$, where $N = N_h \cdot N_w \cdot 3$, be the vector that is constructed by stacking the column of $X$ together. And the feature $f$ could be computed by passing the image $X$ from a convolutional layer.

Then the feature $f$ is mapped by the cosine, sine function and stacked in the concentrating block in order to yield the corresponding random Fourier features with regard to $\omega_i$. Besides the stacking, the concentrating block for image data could be also integrated with more functional structure or tool such as the batch normalization (Ioffe and Szegedy 2015), the dropout (Srivastava et al. 2014), the residual connection (He et al. 2016) and the pooling, etc. In the proposed model, we incorporate the batch normalization and pooling operation into the concentrating block. By reducing internal covariate shift, batch normalization tremendously speeds up the training process, enables higher learning rates and regularizes the model to some extent. And the RFFNet also ben-
efits from the pooling operation due to its augmentation for the noise tolerance and reduction of the parameters to be trained.

Through one layer which contains of a RFF module and a concentrating block, data are projected into a finite-dimensional feature space and then could be treated as the input of the next layer. For instance, the input sample $x \in \mathbb{R}^d$ is transformed into the feature $s_1 \in \mathbb{R}^{2D}$, where $D$ is the number of RFFs. As illustrated in [4], with a larger $D$, the empirical average gives a better estimation to the expectation, i.e., the kernel is approximated better. By stacking layers one by one as described above, we are able to build up an end-to-end deep kernel learning architecture for different tasks.

**Training**

We apply end-to-end training to find suitable model which essentially means that we are training each layer among all the shift-invariant and positive definite kernels. Consider training samples $\{x_1, x_2, \ldots, x_n\}$ with labels $\{y_1, \ldots, y_n\}$, we solve the following problem to establish the discriminative function

$$
\min \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \frac{\lambda}{2} ||f||^2, \quad (7)
$$

where $f$ is the prediction function, $\lambda$ is the penalizing parameter to control the complexity, and $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a smooth loss function.

The training procedure follows the standard way of backpropagation. The particular operation in RFFNet is the sin and cos. Representing $W = \{\omega_1, \ldots, \omega_l\}$, $\omega_i = [\omega_i^1, \ldots, \omega_i^{D_l}]$ as the parameters in the RFF modules and $w$ as the parameters in the last linear layer, utilizing $E_i$ for loss of $x_1$ and $s_i$ for the output of the $j^{th}$ RFF layers, we could update $\omega_i^m$ by the chain rule. For example, we have

$$
\frac{\partial E_i}{\partial \omega_i^m} = \frac{\partial E_i}{\partial y_i} \cdot \frac{\partial y_i}{\partial s_i^m} \cdot \frac{\partial s_i^m}{\partial \omega_i^m} + \frac{\partial y_i}{\partial s_i^{m+D_l}} \cdot \frac{\partial s_i^{m+D_l}}{\partial \omega_i^m}, \quad (8)
$$

where $D_l$ is the dimension of $\omega_i$, $y_i = w^T s_i + b$.

$$
\frac{\partial y_i}{\partial s_i^m} = w^T, \quad \frac{\partial y_i}{\partial s_i^{m+D_l}} = w_{m+D_l},
$$

$$
\frac{\partial s_i^m}{\partial \omega_i^m} = -s_{i-1} \sin[\omega_i^m]^T s_{i-1}], \quad (10)
$$

$$
\frac{\partial s_i^{m+D_l}}{\partial \omega_i^m} = s_{i-1} \cos[\omega_i^m]^T s_{i-1}]. \quad (11)
$$

Since the training problem is non-convex, the initialization may have influence. In this paper, we generate initial $\omega$ from Gaussian distributions, which corresponds to using RBF kernels in each layer. RBF kernels’ performance is generally not bad, as verified in many tasks, our initial setting gives a relatively good cascade structure.

**Analysis**

**Trainable:** In RFFNet, we implement a deep structure and its flexibility is expected to be improved. One advantage of using RFF is its wide coverage to a rich class of kernels. If we can efficiently train the parameters, suitable kernels could be found for each layer. Let us consider an example on dataset “monks1”, which contains 124 training samples with 6 features. By support vector machine (SVM) with RBF kernel, of which the parameter is tuned by cross-validation, the accuracy is about 78.5%. By setting $\omega$ from Gaussian distribution, which is actually our initialization strategy, classification by RFF has similar performance. But in our RFFNet, we can train the distribution and the obtained distribution is displayed in Fig. [2] which shows that after training, the kernel has been changed to another shift-invariant and positive definite one.

**Features layer by layer:** With a deep structure and efficient training, RFFNet can significantly improve the performance of shallow kernel methods, if the flexibility is the bottleneck. For example, in the mentioned “monks1” dataset, RFFNet with two layers has 100% accuracy on the test data. The kernel method is to learn the pairwise relationship to formulate the kernel matrix. For “monks1”, let us check the kernel matrix in each layer, by applying kernel principal component analysis (kPCA, Scholkopf, Smola, and Müller [1998]) to reduce features in two dimensional space. Fig. [3] shows that the kernel matrix in the first layer cannot distinguish the two classes, while, in the next layer, they can be perfectly classified. In other words, good pairwise information have been extracted by RFFNet.

Similar performance is observed in “EEG” dataset. There are 14980 samples with 14 features in EEG. Due to the abundant amount of the data, this task requires stronger learning ability. By support vector machine (SVM) with RBF kernel, of which the parameter is tuned by cross-validation, the accuracy is only about 78.5%, while RFFNet with eleven layers achieves about 98.1%. To better visualize the random Fourier features, here we compute the kernel matrix in different layers, and then apply kPCA to reduce the kernel in three-dimensional space. As illustrated in Fig. [4] in the beginning, the features are mixed together. By going up the hierarchy, the features are mapped by multiple kernels and become disperse from the other class, which gradually possess the linear-separable property. This phenomenon demonstrates the effectiveness of the deep architecture when coping with abundant and complicated data.

**Pair-wise measurement:** Always conducted by the kernel trick, pair-wise measurement is a crucial part in kernel methods, which demonstrates the similarity or dissimilarity between two samples to some extent. The potent representation ability enables the RFFNets to extract considerably precise pair-wise information. Fig. [5] indicates that in the pair-wise matrices, samples with the same labels have a larger pair-wise value, while samples that belong to different categories own a lower similarity. In the figures we are able to see obvious boundaries between two types.

**Experiments**

In this section, comparing with other representative methods, we evaluate the proposed deep kernel learning model on a collection of several tasks, including small data problems, large-scale issues and image classification. All experiments were conducted on a Linux machine with 12-core
Figure 2: The histograms of $\omega$ (in the first dimension) in the two-layer model for “monks1”.

Figure 3: The visualization of random Fourier features in two layers for the classification task monks1. The two classes are marked by the red and blue points. From the first picture we could see the two classes in the first layer are still not linear-separable. While mapped again by the kernel in the second layer, the two classes are nearly separated from each other, which results in a 100% test accuracy.

| Datasets | size(d,n) | SVM-CV | MKL | KRF | MLP | DKL(GP) | RFFNet |
|----------|-----------|--------|-----|-----|-----|---------|--------|
| monks1   | (6, 124)  | 81.5±0.0 | 76.9±1.1 | 85.0±1.2 | 99.1±0.8 | 81.6±4.8 | **100.0±0.0** |
| monks2   | (6, 169)  | 85.8±1.0 | 63.7±5.5 | 73.9±1.3 | 84.9±2.2 | 87.8±4.9 | **98.0±0.7** |
| monks3   | (6, 122)  | 93.1±1.1 | 90.1±1.2 | 93.7±0.9 | 90.7±0.9 | 93.6±1.3 | **93.8±0.8** |
| fertility | (9,100)   | 88.6±2.5 | 66.2±7.1 | 83.0±3.7 | 82.9±3.1 | 93.2±1.8 | 91.2±1.8 |
| sonar    | (60,208)  | **83.4±3.0** | 75.0±2.5 | 82.6±2.3 | 82.7±2.4 | 78.4±2.6 | 82.9±2.3 |
| climate  | (20,540)  | 94.2±1.6 | 90.6±1.5 | 93.1±1.7 | 94.4±0.8 | 92.0±0.0 | **94.6±1.5** |
| Qsar     | (41,1055) | 86.7±1.3 | 84.7±1.6 | 86.9±1.6 | 84.2±0.8 | 61.6±0.5 | **86.9±0.9** |

| Datasets     | size(d,n) | SVM-CV | KRF | MLP | DKL(GP) | RFFNet |
|--------------|-----------|--------|-----|-----|---------|--------|
| phishing     | (30,11055)| 95.7±0.3 | 92.8±0.4 | 95.8±0.4 | 89.4±0.5 | **97.1±0.2** |
| EEG          | (14,14980)| 78.5±1.2 | 55.3±0.5 | 92.5±3.4 | 61.6±2.1 | **98.1±0.4** |
| ijcnn1       | (22,49900)| 90.7±0.0 | 93.0±0.1 | 98.1±0.1 | **98.2±0.2** | 97.7±0.2 |
| covtype      | (54,581012)| 80.0±0.1 | 79.0±0.2 | 96.3±0.2 | 86.1±0.1 | **96.6±0.2** |
Figure 4: The visualization of random Fourier features in different layers in EEG with a 11-layer model. The first row represents the results for layer 1, 3, 5 from the left to right. And the second row is corresponding to layer 7, 9 and 11 respectively. Being mapped by the kernels layer by layer, the features gradually spread apart. The final test accuracy is about 98.1%.

Figure 5: pair-wise information on sonar, climate, monks3 from three-layer models. The axises depict the serial numbers of samples.

3.2GHz Intel CPUs using Matlab and PyTorch, a general deep learning platform. For our deep kernel learning model, we utilize Adam algorithms (Kingma and Ba 2015) as the optimizer with the learning rate 0.001, coefficients betas (0.9, 0.999). The number of layers generally follows the rule $\left\lceil \frac{n}{1000} \right\rceil + 1$, where $n$ is the data size. For binary classification tasks, we use the hinge loss or the squared loss. While the cross entropy loss is exploited to tackle with multi-class classification. The input is pre-processed (normalized) with the whitening procedure. In addition, we use the multivariate Gaussian distribution $\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}} \exp\left(-\frac{1}{2}(\omega - \mu)^{\top}\Sigma^{-1}(\omega - \mu)\right)}$ to initialize $p(\omega; \mu, \Sigma)$ with zero mean value $\mu$ and the diagonal matrix $\Sigma$ where the diagonal elements are 0.01. This operation makes that the initial model could be regarded as a cascade connection of multiple Gaussian kernels. And the regularization parameter $\lambda$ is set to $10^{-4}$.

Results on small data

When limited data are available, it is not very easy to catch the inner characteristics. Rather than the common deep neural networks which suffer from the overfitting, kernel methods usually require less samples but have better generalization capability. In our model, we could well approximate a kernel in each layer, hence it is quite suitable for the small data task.

To show the effectiveness, we choose some small datasets from UCI Machine Learning Repository[1] (Blake 1998) and compare the proposed model with other representative method including SVM-CV (Chang and Lin 2011), KRF (Sinha and Duchi 2016), MKL (Jawanpuria, Nath, and Ramakrishnan 2015), MLP, and DKL (Wilson et al. 2016a). All data are normalized to [0,1]. For some problems, both the training and the test sets are provided, otherwise we randomly pick half of the data as the training set and the rest for test. Kernel parameters and penalty parameters in competitive approaches are tuned by 5-fold cross validation. The LIBSVM (Chang and Lin 2011) toolbox is applied for the

[1]http://archive.ics.uci.edu/ml/datasets.php
SVM-CV with a Gaussian kernel. The initial set in MLP follows our model. As illustrated in Table 1, we report the average test accuracy and the standard deviation on 10 trials of different approaches.

As a deep structure, RFFNet has significant improvement from shallow kernel methods, for which SVM-CV is a good representative. For instance, for “monks1”, “monks2”, and “fertility”, the accuracy is improved to more than 90%, which is almost impossible for shallow methods. But of course, if there is no problem in flexibility for shallow models, the improvement is slight. Compared with other representative approaches, the proposed model significantly benefits from both the flexible, learnable kernel and the deep architecture and hence owns a strong representation capability. For the dataset “monks1”, the proposed model achieves 100% test accuracy, which is slightly higher than 99.1% for MLP and much higher than that for other methods. In “monks2”, our model outperforms over 10% than others. Due to the limited data, MLP has a low performance on “fertility”, our result is 91.2% while the DKL(GP) achieves a high performance 93.2%. But in DKL(GP), it actually contains a MLP and a Gaussian process layer which is more complicated than our model. On the one hand, unlike MLP, our model takes the advantage of kernel methods and does better in alleviating the overfitting problems. On the other hand, the incorporation of deep architecture tremendously improves the richness of kernels and thus our model could have a satisfactory performance. In addition, we could see that MKL generally performs not as well as other methods, hence we do not list it in the following experiments.

Results on large scale data
Training on a large amount of data could alleviate the problem of overfitting while requires stronger representative capability. To evaluate the proposed model in large scale situations, we pick several large scale datasets “phishing”, “EEG”, “ijcnn1” and “covtype” from the UCI repository. In this task, we set 11 as the number of layers in our model and the same to MLP. For extremely large datasets like “covtype”, we train the model with a minibatch size of 256 and the maximum epochs are 300.

As shown in Table 2, we achieve promising performance on these large scale datasets, which ranks first in 3 datasets. Due to the deep structure, the learning ability of our model and MLP are remarkably stronger than SVM-CV and other similar kernel-based approaches, which are unable to perform as well as our model and MLP. In “EEG”, the “shallow” kernel methods, including SVM and KRF, do not have a satisfactory accuracy. Especially the KRF, which also has a supervised approach to learn a kernel, only gets 55.3%. As for DKL(GP), it is possible that the feature extraction layer in DKL(GP) could not catch the valuable information in the samples of “EEG” hence it performs not so well.

For the large scale data, broadly speaking, the models equipped with deep architecture possess better learning ability and strong representation capability than shallow ones. Rather than MLP with rectified linear units, the proposed RFFNet owns a powerful kernel in each layer, which leads to a strong non-linearity and the richness to describe the inner distribution of the data.

| Datasets | CKN [Mairal et al. 2014] | ResNet-18 | VGG-16 | DKL(GP) | RFFNet |
|----------|--------------------------|-----------|--------|---------|---------|
| MNIST    | 99.4                     | 99.3      | 99.4   | 99.2    | 99.1    |
| CIFAR-10 | 80.5                     | 86.1      | 86.0   | 77.0    | 84.6    |

Image data MNIST and CIFAR-10
We now move to the image datasets MNIST [LeCun et al. 1998] and CIFAR-10 [Krizhevsky, Sutskever, and Hinton 2012]. The MNIST dataset consists of handwritten digit images and it is divided in 60,000 examples for the training set and 10,000 examples for testing. CIFAR-10 is an established computer-vision dataset used for object recognition, which has 60000 32×32 color images with 10 classes.

Here we evaluate the performance with a 9-layer model, which has the batch normalization and 3 max-pooling layers for (2, 2), (2, 2), (8, 8), respectively. In addition, here we use 3 × 3 as convolutional kernel size, the stride is one with one padding. The transformation of feature maps is mainly 3-64-64-128-128-256-256. The number of epochs is at most 300 with a minibatch size of 256. Beginning from an initial learning rate of 0.01, we drop it by a factor of 10 after 200 epochs and then again after 240 epochs. Other methods are equipped with their recommended settings.

We present the experimental results in Table 3 with the performance of different approaches without data augmentation. Compared with other kernel-based approaches, the results of VGG-16 (Simonyan and Zisserman 2015) and ResNet-18 (He et al. 2016) are outstanding. But our model also could achieve reasonably comparable performance with 99.1% on MNIST and 84.6% on CIFAR-10. Notice that RFFNet has not been specifically designed and polished for image processing. With suitable supplementary blocks and techniques, the performance may have improvement space.

| Datasets | CKN [Mairal et al. 2014] | ResNet-18 | VGG-16 | DKL(GP) | RFFNet |
|----------|--------------------------|-----------|--------|---------|---------|
| MNIST    | 99.4                     | 99.3      | 99.4   | 99.2    | 99.1    |
| CIFAR-10 | 80.5                     | 86.1      | 86.0   | 77.0    | 84.6    |

Conclusion
In this paper, we propose a deep structure for kernel learning by random Fourier features. On the one hand, RFF provides a good approximation for kernel, which leads to good performance in small datasets. On the other hand, RFF makes it possible to construct multi-layer networks and the kernel in each layer could be efficiently trained by back propagation. Without tricky designs, the proposed RFFNet achieves good performance on small data, large scale data and image

---

1http://yann.lecun.com/exdb/mnist/
2http://www.cs.toronto.edu/ kriz/cifar.html
data such as MNIST and CIFAR-10. Generally, RFFNet is a promising direction to make kernel methods deeper and it may also bridge the theoretical analysis for shallow kernel methods and deep learning.

References

[Blake 1998] Blake, C. 1998. UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[Chang and Lin 2011] Chang, C.-C., and Lin, C.-J. 2011. LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) 2(3):27.

[Cho and Saul 2009] Cho, Y., and Saul, L. K. 2009. Kernel methods for deep learning. In NIPS.

[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In CVPR.

[Ioffe and Szegedy 2015] Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[Jawanpuria, Nath, and Ramakrishnan 2015] Jawanpuria, P.; Nath, J. S.; and Ramakrishnan, G. 2015. Generalized hierarchical kernel learning. Journal of Machine Learning Research 16(1):617–652.

[Kingma and Ba 2015] Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In ICLR.

[Krizhevsky, Sutskever, and Hinton 2012] Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet classification with deep convolutional neural networks. In NIPS.

[LeCun et al. 1998] LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P.; et al. 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11):2278–2324.

[Mairal et al. 2014] Mairal, J.; Koniusz, P.; Harchaoui, Z.; and Schmid, C. 2014. Convolutional kernel networks. In NIPS.

[Mairal 2016] Mairal, J. 2016. End-to-end kernel learning with supervised convolutional kernel networks. In NIPS.

[Rahimi and Recht 2008] Rahimi, A., and Recht, B. 2008. Random features for large-scale kernel machines. In NIPS.

[Rudin 1962] Rudin, W. 1962. Fourier analysis on groups, volume 121967. Wiley Online Library.

[Schölkopf and Smola 2001] Schölkopf, B., and Smola, A. J. 2001. Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press.

[Schölkopf, Smola, and Müller 1998] Schölkopf, B.; Smola, A.; and Müller, K.-R. 1998. Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation 10:1299–1319.

[Simonyan and Zisserman 2015] Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In ICLR.