Do Kernel and Neural Embeddings Help in Training and Generalization?

Neural Processing Letters (2023) 55:1681–1695
https://doi.org/10.1007/s11063-022-10958-8

Do Kernel and Neural Embeddings Help in Training and Generalization?

Arman Rahbar[1] · **Emilio Jorge**[1] · **Devdatt Dubhashi**[1] · **Morteza Haghir Chehreghani**[1]

Accepted: 4 July 2022 / Published online: 11 July 2022

Abstract
Recent results on optimization and generalization properties of neural networks showed that in a simple two-layer network, the alignment of the labels to the eigenvectors of the corresponding Gram matrix determines the convergence of the optimization during training. Such analyses also provide upper bounds on the generalization error. We experimentally investigate the implications of these results to deeper networks via embeddings. We regard the layers preceding the final hidden layer as producing different representations of the input data which are then fed to the two-layer model. We show that these representations improve both optimization and generalization. In particular, we investigate three kernel representations when fed to the final hidden layer: the Gaussian kernel and its approximation by random Fourier features, kernels designed to imitate representations produced by neural networks and finally an optimal kernel designed to align the data with target labels. The approximated representations induced by these kernels are fed to the neural network and the optimization and generalization properties of the final model are evaluated and compared.

Keywords Kernel embedding · Gram matrix · Neural Network · Convergence

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation as well as the Chalmers AI Research Centre. We would like to thank Ashkan Panahi and the anonymous reviewers for their constructive comments.

✉ Arman Rahbar
armanr@chalmers.se

Emilio Jorge
emilio.jorge@chalmers.se

Devdatt Dubhashi
dubhashi@chalmers.se

Morteza Haghir Chehreghani
morteza.chehreghani@chalmers.se

[1] Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden

1 Introduction

Deep neural network models [3, 9, 13] provide the state-of-art results in several tasks and applications, although the theory has not been completely understood yet. The well-known work of [19] highlighted intriguing experimental phenomena about deep network training – specifically, optimization and generalization – and called for a rethinking of generalization in statistical learning theory. In particular, two fundamental questions that need to be answered are:

Optimization. Why do true labels give faster convergence rate than random labels for gradient descent?

Generalization. What property of properly labeled data controls generalization?

An approach to addressing these questions was recently made by [1] in the context of a simple two–layer model by conducting a spectral analysis of the associated Gram matrix. They show that *training improves if the label vector aligns with the top eigenvectors* of the associated Gram matrix ($\mathbf{H}^{\infty}$ in Sect. 2). In addition, they provide a data-dependent complexity measure which could be used to upper bound the generalization error of the neural network. This measure is also related to the Gram matrix.

However, their analysis applies only to a simple two layer network. *How could their insights be extended to deeper networks?*

A widely held intuitive view is that deep layers generate expressive *representations* of the raw input data. Adopting this view, one may consider a model where a representation generated by successive neural network layers is viewed as a *kernel embedding* which is then fed into the two–layer model of [1]. The connection between neural networks and kernel machines has long been studied; [4] introduced kernels that mimic deep networks and [15] showed kernels equivalent to certain feed–forward neural networks. Recently, [2] also make the case that progress on understanding deep learning is unlikely to move forward until similar phenomena in classical kernel machines are recognized and understood.

The fundamental question we address is: *Do representations provided by kernel embeddings help in training and generalization*? We address this question in the context of the simple model of [1] and their approach based on a spectral analysis of the associated Gram matrix. Specifically, we take a view of a multi–layer network as first producing an embedding $\phi(x)$ of the input which is then fed as input to the simple two layer network of [1]. While a general transformation $g(x)$ of the input data may have arbitrary effects, one might expect a universal kernel representation to improve performance. Do kernel representations create a better alignment of labels with the top eigenvectors of the Gram matrix? We investigate this first by using kernels which are label-unaware[1] such as Gaussian kernel, using random Fourier features(RFF) to approximate the Gaussian kernel embedding [12].

Next, we address the question: *Do representations provided by neural embeddings help in training and generalization?* We address this in two ways: first, we use kernels that have been designed to specifically mimic neural networks [4]. Then we use data driven embeddings explicitly produced by the hidden layers in neural networks: either using a subset of the same training data to compute such an embedding, or transfer the inferred embedding from a different (but similar) domain.

We indeed find substantial improvements in both training and generalization using any of the above kernel representations.

[1] We say Gaussian kernel is label-unaware because we do not employ any data labels to compute it, i.e., this kernel is not sensitive to the labels.

Finally we ask the question: *What is a good representation for optimization?* The work of [1] suggests seeking a representation which makes the label vector align best to the top eigenvectors of the associated Gram matrix. This connects to the problem of *kernel–target alignment* [5, 6]. We use the criterion suggested in these papers as a proxy for aligning the label vector to the top eigenvectors and compute the representation so produced. Indeed we observe that this representation provides the best results for training.

Thus this work shows that kernel and neural embeddings improve the alignment of target labels to the eigenvectors of the Gram matrix and thus help training. This suggests a way to extend the insights of [1] to deeper networks, and possible theoretical results in this direction.

In addition, the work in [1] yields a data-dependent complexity measure to be used to upper bound the generalization error of a learned neural network. We adapt this measure to be used with kernel and neural embeddings to analyze the generalization performance resulted from these representations.

2 Spectral Theory

Network model. In [1], the authors consider a simple two layer network model:

$$f_{\boldsymbol{W},\boldsymbol{a}}(\boldsymbol{x}) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \max(0, \boldsymbol{w}_r^T \boldsymbol{x}_i), \tag{1}$$

with $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{w}_1, .. \boldsymbol{w}_m \in \mathbb{R}^{d\times m}$ and $(a_1, .. a_m)^T \in \mathbb{R}^m$ (where m specifies the number of neurons in the hidden layer, i.e., its width). These can be written jointly as $\mathbf{a} = (a_1, .., a_m)^T$ and $\mathbf{W} = (\boldsymbol{w}_1, .., \boldsymbol{w}_m)$. This network is trained on dataset of data points $\{x_i\}$ and their targets $\{y_i\}$.

They provide a fine grained analysis of training and generalization error by a spectral analysis of the *Gram matrix*:

$$\begin{aligned} \mathbf{H}^{\infty}_{i,j} :=& E_{\mathbf{W}\sim\mathcal{N}(0,\boldsymbol{\mathcal{I}})}[\mathbf{x}_i^T \mathbf{x}_j 1\{\mathbf{w}^T \mathbf{x}_i \geq 0, \mathbf{w}^T \mathbf{x}_j \geq 0\}] \\ =& \frac{\mathbf{x}_i^T \mathbf{x}_j (\pi - \arccos(\mathbf{x}_i^T \mathbf{x}_j))}{2\pi}. \end{aligned} \tag{2}$$

If

$$\mathbf{H}^{\infty} = \sum_i \lambda_i \mathbf{v}_i \mathbf{v}_i^T \tag{3}$$

is the orthonormal decomposition of $\mathbf{H}^{\infty}$, [1] shows that training improves if the label vector y aligns with the eigenvectors corresponding to the top eigenvalues of $\mathbf{H}^{\infty}$.

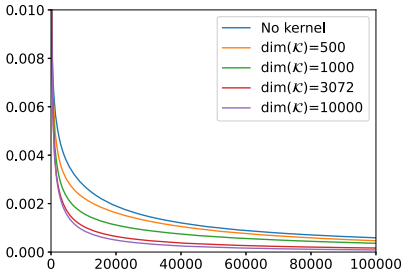
We extend the two-layer ReLU network in [1] via adding different types of embeddings ϕ at the input layer corresponding to a kernel $\mathcal{K}$. The corresponding model is:

$$f_{\mathbf{W},\mathbf{a}}(\mathbf{x}) = \frac{1}{\sqrt{m}} \sum_{r=1}^{m} a_r \max(0, \mathbf{w}_r^T \boldsymbol{\phi}(\mathbf{x}_i)). \tag{4}$$

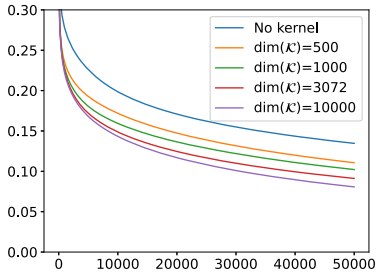
For a representation $(\boldsymbol{\phi}(\boldsymbol{x}_i), i \in [n])$ corresponding to a kernel $\mathcal{K}$, Since $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j), i, j \in [n])$, similar to 2 define the Gram Matrix

$$\begin{aligned} \mathbf{H}(\mathcal{K})^{\infty}_{i,j} :=& E_{\mathbf{W}}[\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) 1\{\mathbf{w}^T \boldsymbol{\phi}(\boldsymbol{x}_i) \geq 0, \mathbf{w}^T \boldsymbol{\phi}(\boldsymbol{x}_j) \geq 0\}] \\ =& \frac{\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)(\pi - \arccos(\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)))}{2\pi}. \end{aligned} \tag{5}$$

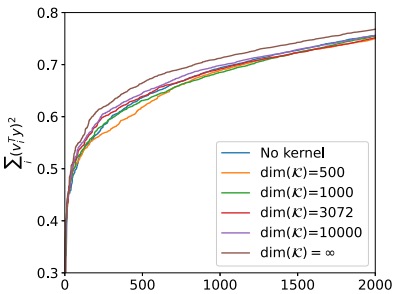
and let its eigenvalues be ordered as



(a) MNIST training loss as a function of epoch number



(b) CIFAR-10 training loss as a function of epoch number



(c) CIFAR-10 projections

Fig. 1 Performance on first two classes of MNIST and CIFAR-10 as training datasets. We observe that the different kernels yield faster convergence of the loss function on training data compared to non-kernel variant. Figure 1(c) demonstrates alignment of top eigenvalues and the projections of true labels on corresponding eigenvectors

$$\lambda_0(\mathcal{K}) \geq \lambda_1(\mathcal{K}) \geq \cdots \geq \lambda_{n-1}(\mathcal{K}) \tag{6}$$

and let $\mathbf{v}_0(\mathcal{K}), \cdots, \mathbf{v}_{n-1}(\mathcal{K})$ be the corresponding eigenvectors.

A kernel $\mathcal{K}$ such that the corresponding eigenvectors align well with the labels would be expected to perform well for training optimization. This is related to kernel target alignment [6].

Optimization. By adapting the convergence of the gradient descent for basic model in [1], the convergence of our kernelized network is controlled by

$$\sqrt{\sum_i (1 - \eta\lambda_i(\mathcal{K}))^{2k} (\mathbf{v}(\mathcal{K})_i^T \mathbf{y})^2} \tag{7}$$

Generalization. We can also adapt the generalization performance of the simple two-layer network in [1] to the kernel embedding setting as

$$\sqrt{\frac{2\mathbf{y}^T (\mathbf{H}(\mathcal{K})^\infty)^{-1}\mathbf{y}}{n}} \tag{8}$$

For both optimization and generalization, we replace the data features with the feature vectors induced by the kernel embeddings.

3 Numerical Analysis

We perform our numerical analysis on various datasets. In addition to MNIST and CIFAIR-10 which are used for the experiments in [1], we used other datasets such as LSUN [18] and Fashion-MNIST in several parts of the paper. As in [1], we only look at two classes when training our models, and set the label $y_i = +1$ if image i belongs to the first class and $y_i = -1$ if it belongs to the second class. To follow the setting in [1], the images are normalized such that $||\boldsymbol{x}_i||_2 = 1$. This is also done for the different kernel embeddings such that $||\boldsymbol{\phi}(\boldsymbol{x}_i)||_2 = 1$.

The weights in Eq. (4) are initialized as follows:

$$\boldsymbol{w}_i \sim \mathcal{N}(0, k^2\boldsymbol{\mathcal{I}}),\ a_r \sim \text{Unif}(\{-1, 1\}), \forall r \in [m]. \tag{9}$$

We then use the following loss function to train the model to predict the image labels.

$$\Phi(\mathbf{W}, \mathbf{a}) = 1/2 \sum_{(i=1)}^{n} (y_i - f_{\mathbf{W},\mathbf{a}}(\mathbf{x}))^2 \tag{10}$$

For optimization, we use (full batch) gradient descent with the learning rate η. We set $k = 10^{-2}$, $\eta = 2 \times 10^{-4}$ similar to [1].

3.1 Gaussian Kernel Method

We first use the Gaussian kernel

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) := \exp\left(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right). \tag{11}$$

The corresponding embedding is infinite dimensional, hence we consider the fast approximations to the kernel given by *random Fourier features* (RFF) [12]. The idea of random Fourier features is to construct an explicit feature map which is of a dimension much lower than the number of observations, but the resulting inner product approximates the desired kernel function. Specifically, instead of using the kernel trick for computing the inner product in the higher dimensional space (with feature map ϕ), a method is proposed to create a randomized feature map z such that $\phi(x)^T\phi(y) \approx z(x)^T z(y)$. The approximated feature map in RFF includes sinusoids that are randomly selected from the Fourier transform of the kernel function [12]. We use $\gamma = 1$ in all our experiments.

We start by investigating the use of Gaussian kernels for a more efficient optimization of the loss function on the training data. Figures 1(a) and 1(b) show the training loss at different steps respectively on first two classes of MNIST and CIFAR-10 datasets. We observe that Gaussian kernel embeddings with different dimensions yield faster convergence in the optimization on both datasets. MNIST is a simple dataset which gives incredibly high score almost immediately, as shown by the training loss (Fig. 1(a)) and by the accuracy on the test data (Table 1). Since we already reach very high accuracy on MNIST without using kernels, different methods yield similar results on this dataset. We observe this behavior in Fashion-MNIST dataset as well. So in this section, we will focus our analysis on the CIFAR-10 dataset. Similar to the setup in [1], in Fig. 1(c), for different methods, we plot the projections of the true class labels on the eigenvectors (i.e., the projections $\{(\mathbf{v}_i^T\mathbf{y})^2\}_{i=0}^{n-1}$). For better visualization, we plot the cumulative forms

$$f_i = \sum_{j=0}^{i} (\mathbf{v}_j^T\mathbf{y})^2 \tag{12}$$

Table 1 Accuracy on MNIST test set (first two classes) after n epochs. We reach a very high accuracy after only a small number of epochs and the test error becomes negligible

$dim(\mathcal{K})$	Steps			
	10	1000	50000	100000
None	0.998582	0.999527	0.999527	0.999527
500	0.996690	0.999054	0.999054	0.998582
1000	0.998109	0.999054	0.999527	0.999054
3072	0.998582	0.999527	0.999527	0.999527
10000	0.998582	0.999527	0.999527	0.999527

Table 2 Quantification of $\sqrt{\frac{2\mathbf{y}^T(\mathbf{H}^\infty)^{-1}\mathbf{y}}{n}}$ (or $\sqrt{\frac{2\mathbf{y}^T(\mathbf{H}(\mathcal{K})^\infty)^{-1}\mathbf{y}}{n}}$) for Gaussian kernels in different experimental settings ($n = 12665$ for MNIST and $n = 10000$ for CIFAR). For both datasets, most of the Gaussian kernels yield smaller upper bounds on generalization error

γ	Dimension	MNIST	CIFAR-10
0.01	∞	0.99	7.37
0.1	∞	0.48	3.77
1	∞	0.26	1.82
10	∞	1.24	0.49
1	500	0.35	3.33
1	1000	0.32	3.20
1	3072	0.29	3.08
1	10000	0.27	2.98
No kernel		0.35	3.86

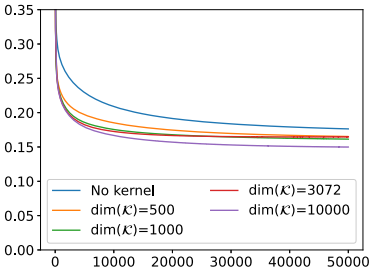
which are normalized such that $\sum_{i=0}^{n-1}(\mathbf{v}_i^T\mathbf{y})^2 = 1$.

The results show that using kernels yields a better alignment of the eigenvectors belonging to the largest eigenvalues and the target labels, leading to faster convergences. In other words, with kernels, we attain larger $(\mathbf{v}_i^T\mathbf{y})^2$'s for top eigenvalues.

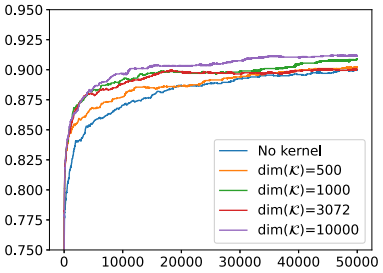
We continue by investigating the generalization performance of the Gaussian kernel method by analyzing the values of Eq. (8). Table 2 shows this quantity for different settings and kernels respectively on first two classes of MNIST and CIFAR-10 datasets. Please note that for the infinite dimensional case we do not need to perform normalization since for Gaussian kernels the images of input data in the feature space are already normalized[2]. We observe that in both datasets with several kernels we obtain a lower theoretical upper bound on the generalization error compared to the no-kernel case. It is clear that the bound improves as the dimension of the representations increases but also that the generalization bound seems quite sensitive to values of γ.

In addition to the theoretical upper bound, we measure the test error for the studied datasets. Figures 2(a) and 2(b) show respectively the test error and the test accuracy at different steps of the optimization by Gradient Descent for first two classes of CIFAR-10. Note that in the mentioned figures (and similar figures that follow) we show the test error (loss) and accuracy at each phase of the training. For instance, to compute accuracy we use our current model to predict the labels in a test dataset. We observe that the kernel methods yield significant improvements of both the test error and the accuracy on the test dataset. We observe that the larger the kernel embedding, the larger the improvement. Additionally, we can see a sharper reduction in test error compared to the no-kernel case. This sharp transition (after a small number of steps) comes with a significant improvement in the test accuracy. This observation suggests that early-stopping can be even more efficient when using kernel methods. It is

[2] This comes from the fact that $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}) = 1$ for Gaussian kernels. For more detail please refer to [14].

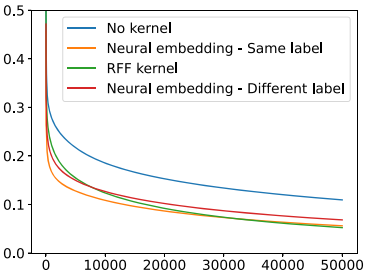


(a) Test error on CIFAR-10 data as a function of epoch number.

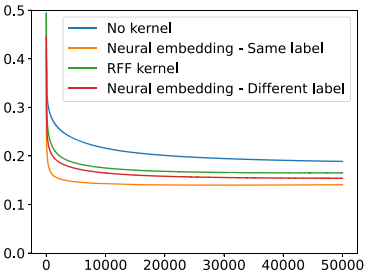


(b) Test accuracy on CIFAR-10 data as a function of epoch number.

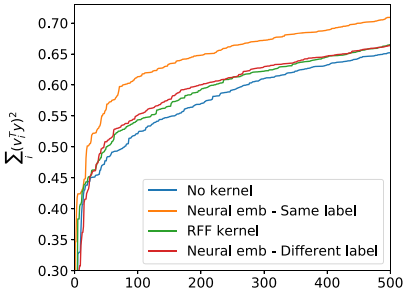
Fig. 2 Experimental test errors and accuracy on the test set at the different steps of the Gradient Descent optimization algorithm for first two classes of CIFAR-10 dataset



(a) Training loss as a function of epoch number.



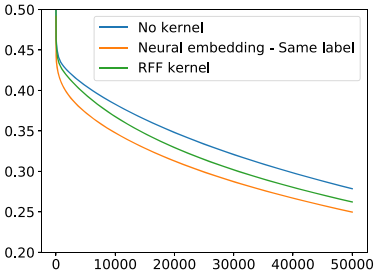
(b) Test loss as a function of epoch number



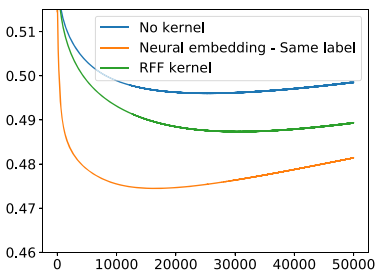
(c) Projections

Fig. 3 Experimental train and test errors at the different steps of Gradient Descent as well as eigenvector projections for the first two classes of CIFAR-10 dataset. For the model pre-trained with the same labels, the training loss and projections are calculated based on the unseen subset of training data. We observe that neural embeddings improve the convergence, generalization and the alignment of eigenvector projections

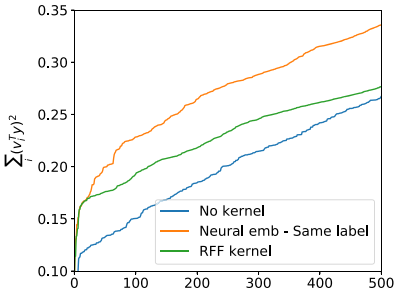
worth noting that these improvements are results of choosing appropriate parameter for the kernels. For instance, a Gaussian kernel with $\gamma = 1000$ would not generalize well. The generalization results shown in Fig. 2(a) are consistent with the measures in Table 2.



(a) Training loss as a function of epoch number.



(b) Test loss as a function of epoch number



(c) Projections

Fig. 4 Experimental train and test errors at the different steps of Gradient Descent as well as eigenvector projections for the "kitchen" and "living room" classes of LSUN dataset. We observe that neural embeddings improve the convergence, generalization and the alignment of eigenvector projections

3.2 Neural Embedding

Choosing an appropriate kernel and its parameters can be challenging [10], as also seen in Table 2. Thus, we investigate a data-dependent neural embedding. For this purpose, we add a second hidden layer to the neural network with $m = 10000$ hidden units and ReLU activation. We pre-train this embedding using two different approaches. The first layer is then kept fix as an embedding where the rest of the network is reinitialized and trained. The first approach is to split the training data in half. We use the first subset to pre-train this three-layer network and the second subset to use for our optimization experiments. In this approach we double η to keep the step length the same. The other approach is to use data from a different domain for pre-training. For instance, we use the last two classes of the CIFAR-10 dataset for pre-training the embedding. We compare our results with not using any kernel and with using an RFF kernel with embedding of size 10000. In Fig. 3 we show results for the two mentioned settings on first two classes of CIFAR-10: i. using a subset of the training data for pre-training (same label), ii. using data from other classes for pre-training (different label). Also in Fig. 4 we show the results of experiments on "kitchen" and "living room" classes of LSUN with the same label setting. Figures 3 and 4 show the average of 5 different random initializations. We observe very insignificant standard deviation in different stages of training. For instance, both Gaussian and neural kernels show standard deviation of order 10^{-4} for training loss when using the LSUN dataset in different steps of training.

3.2.1 Optimization

Figure 3(a) shows the training loss for the first two classes CIFAR-10 dataset. We observe the neural embeddings achieve faster convergence compared to the previous methods. We report the training loss for neural embedding (same label) on the second (unused) subset of the data, whereas in the other cases we report the results on the full training data. If we use only the second subset for the other methods, we observe very consistent results to Fig. 3. Figure 3(c) demonstrates eigenvector projections on the target labels. This shows that both variants of neural embeddings improve alignment of the labels to eigenvectors corresponding to larger eigenvalues (compared to the best RFF kernel). While the effect is unsurprisingly larger when pre-training on the same labels, it is still significantly better when pre-trained on other labels. Also in Figs. 4(a) and 4(c), we see similar results for the LSUN dataset, and the neural embedding outperforms RFF kernel in terms of training loss. The alignment of labels and eigenvectors is improved as well.

3.2.2 Generalization

In Figs. 3(b) and 4(b) we report the test error on the first two classes CIFAR-10 and "kitchen" and "living room" classes of LSUN respectively. This shows that the neural embeddings perform at least comparable with the best studied RFF kernel. If the pre-training is done on the same labels we obtain a clear improvement, even if the actual training is only done on a dataset with half the size.

Finally, it is worth mentioning that similar results are obtained when the pre-training process is stopped earlier (i.e., 5,000 epochs).

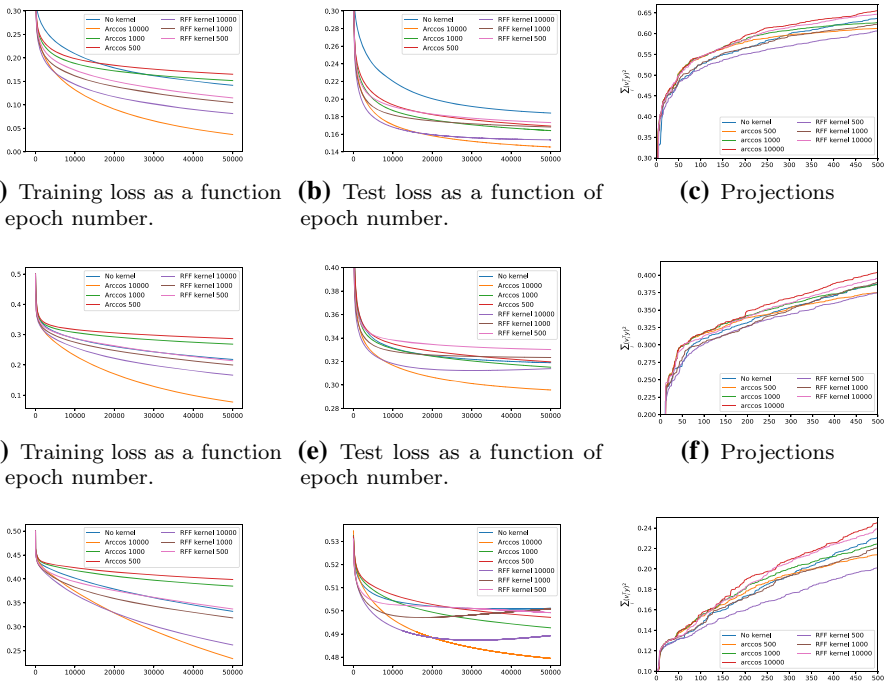
3.3 Embeddings with Other Kernels

So far, we have used two embedding methods: i) a fixed embedding according to Gaussian kernels, and ii) the embeddings induced by deep neural networks. In this section we analyze the use of kernels designed for our task of interest. In particular, we study two types of kernels: i) the arc-cosine kernel proposed in [4] which represents neural network layers. Note that, as we will discuss, computing this kernel is significantly faster than training a neural network layer. Thus, it can be used for the same purpose, i.e., an embedding induced by a neural network, but in a more efficient way. ii) The multiple kernel learning method proposed in [5]. Given the training data and the corresponding labels, this method computes the optimal feasible kernel, independent of any classification model or algorithm. We call the resulting kernel optimal since an optimization function is defined and the final kernel is found based on the solution from the optimization problem (see Sect. 3.3.2).

In both cases, we first approximate the kernel embedding using the Nyström method [17], and then feed the approximated kernel embedding to the two-layer neural network model, instead of the original data features.

3.3.1 Arc-Cosine Kernel

The arc-cosine kernel [4] mimics the computations in a neural network within an infinite dimensional feature space, where the kernel function is equivalent to the inner product of the feature vectors from a neural network. Then, the Nyström method with different number of components can be used to approximate the kernel embeddings.



(a) Training loss as a function of epoch number. **(b)** Test loss as a function of epoch number. **(c)** Projections

(d) Training loss as a function of epoch number. **(e)** Test loss as a function of epoch number. **(f)** Projections

(g) Training loss as a function of epoch number. **(h)** Test loss as a function of epoch number. **(i)** Projections

Fig. 5 The results of the embeddings induced by the arc-cosine kernel on first two classes of CIFAR-10 (a-c), second and third classes of CIFAR-10 (d-f) and "kitchen" and "living room" classes of LSUN (g-i) with different dimensionality of parameterizations. This kernel performs better than the Gaussian kernel for sufficiently large number of components and the case where we use the original features as in [1]. The overlap measure in Eq. 12 is verified for this case as well

The n-th order *arc–cosine* kernel between two inputs x and y is computed by

$$k_n(x, y) = \frac{1}{\pi}\|x\|^n\|y\|^n J_n(\theta). \tag{13}$$

Where the function J_n and θ are given by

$$J_n(\theta) = (-1)^n(\sin\theta)^{2n+1}\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\right)^n\left(\frac{\pi-\theta}{\sin\theta}\right) \tag{14}$$

$$\theta = \cos^{-1}\left(\frac{x.y}{\|x\|\|y\|}\right) \tag{15}$$

In [4] it is shown that the computations in single-layer threshold networks is closely related to this kernel. Specifically, the kernel is equivalent to the inner product of embeddings induced by a single-layer threshold network. Moreover, [4] suggests a family of kernels mimicking multi-layer networks based on the base kernel. In our experiments we use the base kernel with $n = 0$[3]. It is worth mentioning that earlier, but with different activation functions, these

[3] The reason behind this particular choice is that in [4] it is mentioned that best results are achieved by $n = 0$ or 1.

kernels arose as covariance functions of limiting Gaussian processes of wide neural networks [11]. Also in [16] closed form kernels for different activation functions are derived. Moreover, in [15] the kernel for Leaky ReLU activations is calculated.

Figures 5(a)-5(c), 5(d)-5(f) and 5(g)-5(i) show the results of the arc-cosine kernel embeddings on first two classes of CIFAR-10, second and third classes of CIFAR-10, and "kitchen" and "living room" classes of LSUN respectively. We conducted these experiments with 5 different random initializations and these figures illustrate the average results. We observe that arc-cosine kernel (with a sufficiently large number of components) outperforms the Gaussian kernels as well as the no-kernel case in all of our experiments. The standard deviation of our results in different stages of training was very insignificant. For example, for the arc-cosine and Gaussian kernel embeddings of size 10000, the standard deviations of training loss are in the order of 10^{-4}. Similar to the Gaussian kernel increasing the number of components in arc-cosine kernel embeddings improves the estimations, and hence, leads to a faster optimization and a lower test error. Moreover, the overlap parameters are also better for arc-cosine kernel with a large number of compoenents (i.e., arccos 10000). These results show that assuming the arccosine kernel does approximate neural embeddings, then the results of [1] extend to multilayer networks.

Notice that Neural Tangent Kernel (NTK) [8] is another approach to compute a kernel representing a neural network. However, the applicability of this method is confined by its very inefficient computational runtime. By inefficiency we mean it is not a good choice to use NTK together with Nyström method.[4]

3.3.2 Multiple Kernel Learning

Finally, we investigate learning an appropriate kernel, independent of the model or the algorithm to be used for training and classification. For this purpose, we use the method proposed in [5] that suggests an algorithm to learn a new kernel from a group of kernels based on a similarity measure between the kernels, namely centered alignment. Then, the problem of learning a kernel with a maximum alignment between the data and the labels is formulated as a quadratic programming (QP) problem. The respective algorithm is called *alignf* [5]. This multiple kernel learning algorithm does not involve parameter selection and as mentioned it is computed solely based on the true class labels. The kernel learning algorithm works based on *centered kernel alignment*. If K_1 and K_2 are two kernel matrices, the centered alignment between them can be computed by the following [7].

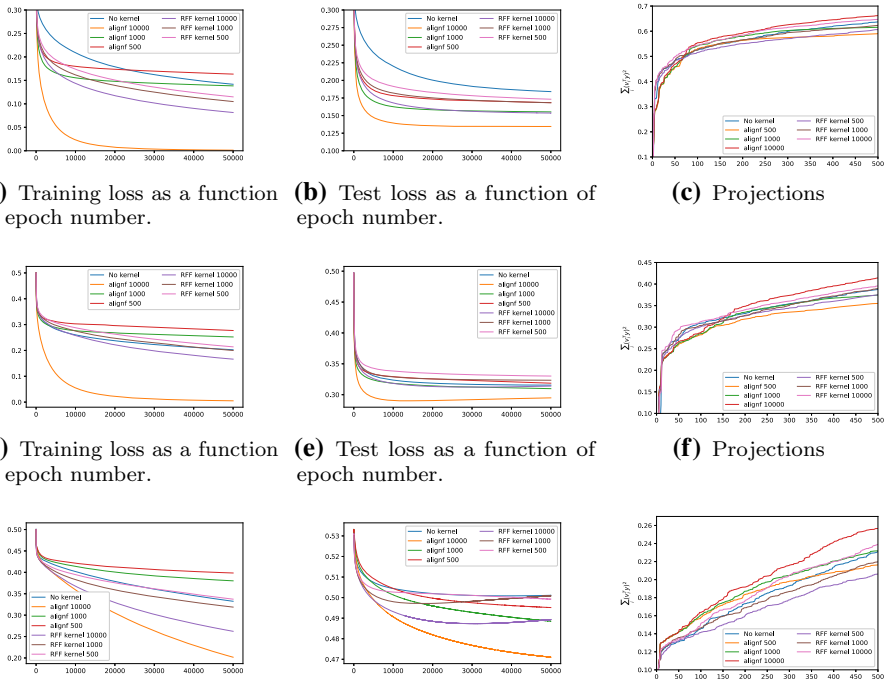
$$CA(K_1, K_2) = \frac{\langle K_1^c, K_2^c \rangle_F}{\sqrt{\langle K_1^c, K_1^c \rangle_F \langle K_2^c, K_2^c \rangle_F}}, \tag{16}$$

where K^c is the centered version of the kernel matrix K. To find the optimal combination of the kernels (i.e. a weighted combination of some base kernels), [5] suggests the objective function to be centered alignment between the combination of the kernels and yy^T, where y is the labels vector. By restricting the weights to be non-negative, a QP can be obtained as

$$minimize \ v^T M v - 2v^T a \ \ w.r.t. \ v \in R_+^P \tag{17}$$

P is the number of the base kernels and $M_{kl} = \langle K_k^c, K_l^c \rangle_F$ for $k, l \in [1, P]$, and finally a is a vector wherein $a_i = \langle K_i^c, yy^T \rangle_F$ for $i \in [1, P]$. If v^* is the solution of the QP, then the vector of kernel weights is given by [5, 7]

[4] In our experiments on even a significantly smaller dataset (i.e., with only 1000 images) it took about 24 hours to run the algorithm on GPU.



(a) Training loss as a function of epoch number. **(b)** Test loss as a function of epoch number. **(c)** Projections

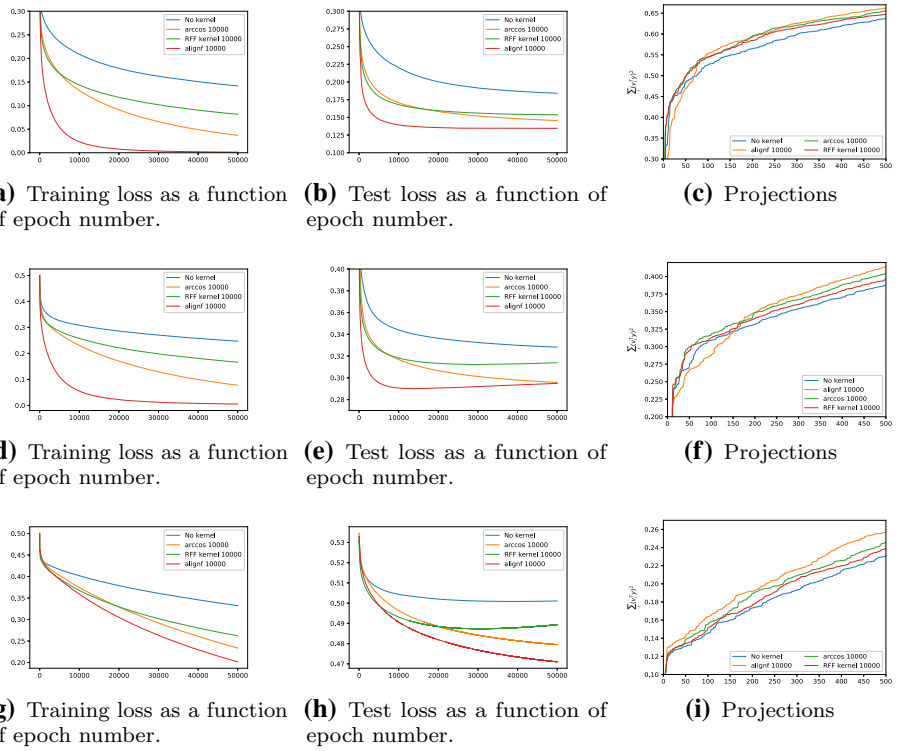
(d) Training loss as a function of epoch number. **(e)** Test loss as a function of epoch number. **(f)** Projections

(g) Training loss as a function of epoch number. **(h)** Test loss as a function of epoch number. **(i)** Projections

Fig. 6 The results of the embeddings induced by the *alignf* kernel on first two classes of CIFAR-10 (a-c), second and third classes of CIFAR-10 (d-f), and "kitchen" and "living room" classes of LSUN (g-i) with different dimensionality of parameterizations. This kernel is optimally learned w.r.t. the data and significantly performs better than the arc-cos kernel, the Gaussian kernel and the no-kernel case. We again observe the consistency of a fast optimization, a low test error and a large overlap measure defined in Eq. 12

$$\mu^* = v^*/\|v^*\| \tag{18}$$

Figures 6(a)-6(c), 6(d)-6(f), and 6(g)-6(i) show the results of the *alignf* embeddings with different dimensions on first two classes of CIFAR-10, second and third classes of CIFAR-10, and "kitchen" and "living room" classes of LSUN respectively. Similar to the experiments in [5], we use a combination of Gaussian kernels (i.e., Eq. (11)) with changing the parameter γ. Specifically, we employ 7 different values for the parameter: $\gamma \in \{2^{-3}, 2^{-2}, \ldots, 2^2, 2^3\}$. The experiments are repeated with 5 different random initializations. Figure 6 illustrates the average of our results. The optimal kernel learned by the *alignf* algorithm achieves the best results, as expected. Similar to Gaussian and arc-cosine kernels, the standard deviation of the results for alignf kernel was also significantly low. For instance, the standard deviation of training loss in different stages of training are in the order of 10^{-5} in the setting of Fig. 6(a) and 10^{-4} in the settings of Figs. 6(d) and 6(g) (when using alignf kernel embeddings with 10000 features). As in previous cases, we observe direct relations between a faster optimization, a better generalization and an improved overlap measure. All the other embeddings such as the arc-cosine kernel designed as a proxy for neural embeddings could be viewed as trying to reach the performance of this *optimal representation*.



(a) Training loss as a function of epoch number. **(b)** Test loss as a function of epoch number. **(c)** Projections

(d) Training loss as a function of epoch number. **(e)** Test loss as a function of epoch number. **(f)** Projections

(g) Training loss as a function of epoch number. **(h)** Test loss as a function of epoch number. **(i)** Projections

Fig. 7 The consistency of fast optimization, low test error and high overlap between the labels vector and the top eigenvectors using the first two classes of CIFAR-10 (a-c), second and third classes of CIFAR-10 (d-f), and "kitchen" and "living room" classes of LSUN (g-i) as the dataset. We see a consistent ranking of kernel methods where *alignf* yields the best results for all the three criteria. The second best method is the arc-cosine kernel, followed by the Gaussian kernel. All the three kernels (alignf, arc-cos and Gaussian) outperform the base setting used in [1]

3.4 Discussion

We have studied several kernel embeddings to obtain more sophisticated features for the basic neural network model. In Figs. 7(a)-7(c), 7(d)-7(f) and 7(g)-7(i) we compare the different kernels with the largest number of dimensions (i.e., when the number of features/components is 10, 000) which provides the most precise approximations. We observe the consistency of fast optimization, low test error and high overlap between the labels vector and the top eigenvectors. In addition, we observe a consistent ranking of different kernel methods. The multiple kernel learning method in [5] (called *alignf*) yields the best results for all the three criteria and in all datasets. Indeed this is expected since it is explicitly optimized for overlap with data labels. All other kernels could be viewed as trying to reach the performance of this kernel. The second best method is the arc-cos kernel which mimics the computations in a neural network, and as expected, it performs better than a generic kernel like Gaussian (the kernel embeddings computed directly from a neural network are comparable to the arc-cos kernels, however they require a significantly larger runtime to be computed). Using the original data features yields the worst results, which indicates the importance of proper feature learning and inference for the task of interest.

Table 3 Quantification of $\sqrt{\frac{2\mathbf{y}^T(\mathbf{H}^\infty)^{-1}\mathbf{y}}{n}}$ (or $\sqrt{\frac{2\mathbf{y}^T(\mathbf{H}(\mathcal{K})^\infty)^{-1}\mathbf{y}}{n}}$) for different kernels on first two classes of CIFAR-10 dataset ($n = 10000$). The alignf method yields the smallest upper bound in the generalization error

Kernel	Dimension	CIFAR-10
Gaussian	10000	2.98
arc-cosine	10000	2.59
alignf	10000	1.45
No kernel		3.86

Finally, in Table 3 we compare the theoretical upper bounds on the generalization errors of the different kernels when used with the first two classes of CIFAR-10. We observe a consistent ranking with the previous results. The *alignf* method yields the best performance, and the arc-cos and the Gaussian kernels are the next choices, both better than the no-kernel case.

4 Conclusions

We empirically explored the implications of recent results of [1] for deeper networks, viewing previous layers as producing better representations of the input data. We studied different kernel and neural embeddings and showed that such representations benefit both optimization and generalization. We demonstrated the applicability of the overlap criteria in [1] for analysing the impacts of different kernels. In particular, we showed that a kernel optimally aligned to data yields the best results in terms of fast optimization, low generalization error, and large overlap between the top eigenvectors and the labels vector, and that other embeddings achieved different approximations to this optimal representation. By combining recent results connecting kernel embeddings to neural networks such as [15], one may be able to extend the fine–grained theoretical results of [1] for two layer networks to deeper networks.

Funding Open access funding provided by Chalmers University of Technology.

References

1. Arora S, Du SS, Hu W, Li Z, Wang R (2019) Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, 322–332
2. Belkin M, Ma S, Mandal S (2018) To understand deep learning we need to understand kernel learning. In: Dy J, Krause A (eds.) Proceedings of the 35th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 80, 541–549. PMLR
3. Bengio Y, Courville AC, Vincent P (2013) Representation learning: A review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. 35(8):1798–1828

4. Cho Y, Saul LK (2009) Kernel methods for deep learning. In: Bengio Y, Schuurmans D, Lafferty JD, Williams CKI, Culotta A (eds.) Advances in Neural Information Processing Systems 22, 342–350. Curran Associates, Inc
5. Cortes C, Mohri M, Rostamizadeh A (2012) Algorithms for learning kernels based on centered alignment. J. Mach. Learn. Res. 13:795–828
6. Cristianini N, Shawe-Taylor J, Elisseeff A, Kandola JS (2001) On kernel-target alignment. In: Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada], 367–373
7. Gönen M, Alpayd E (2011) Multiple kernel learning algorithms. J. Mach. Learn. Res. 12:2211–2268
8. Jacot A, Gabriel F, Hongler C (2018) Neural tangent kernel: Convergence and generalization in neural networks. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R (eds.) Advances in Neural Information Processing Systems 31, 8571–8580. Curran Associates, Inc
9. LeCun Y, Bengio Y, Hinton GE (2015) Deep learning. Nature 521(7553):436–444
10. von Luxburg U (2007) A tutorial on spectral clustering. Statistics and Computing 17(4):395–416
11. Neal RM (1996) Bayesian Learning for Neural Networks. Springer-Verlag, Berlin, Heidelberg
12. Rahimi A, Recht B (2007) Random features for large-scale kernel machines. In: Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007, 1177–1184
13. Schmidhuber J (2015) Deep learning in neural networks: An overview. Neural Networks 61:85–117
14. Schölkopf B, Smola AJ, Bach F, et al (2002) Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press
15. Tsuchida R, Roosta-Khorasani F, Gallagher M (2018) Invariance of weight distributions in rectified mlps. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018, 5002–5011
16. Williams CKI (1998) Computation with infinite neural networks. Neural Computation 10(5):1203–1216
17. Williams CKI, Seeger M (2001) Using the nyström method to speed up kernel machines. In: Leen TK, Dietterich TG, Tresp V (eds.) Advances in Neural Information Processing Systems 13, 682–688. MIT Press
18. Yu F, Zhang Y, Song S, Seff A, Xiao J (2015) Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. arXiv preprint arXiv:1506.03365
19. Zhang C, Bengio S, Hardt M, Recht B, Vinyals O (2017) Understanding deep learning requires rethinking generalization. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings