


Deep Kernelized Autoencoders

Michael Kampffmeyer¹(✉), Sigurd Løkse¹, Filippo M. Bianchi¹ ,
Robert Jenssen¹, and Lorenzo Livi²

¹ Machine Learning Group,
UiT–The Arctic University of Norway, Tromsø, Norway
michael.c.kampffmeyer@uit.no
<http://site.uit.no/ml/>

² Department of Computer Science,
University of Exeter, Exeter, UK

Abstract. In this paper we introduce the deep kernelized autoencoder, a neural network model that allows an explicit approximation of (i) the mapping from an input space to an arbitrary, user-specified kernel space and (ii) the back-projection from such a kernel space to input space. The proposed method is based on traditional autoencoders and is trained through a new unsupervised loss function. During training, we optimize both the reconstruction accuracy of input samples and the alignment between a kernel matrix given as prior and the inner products of the hidden representations computed by the autoencoder. Kernel alignment provides control over the hidden representation learned by the autoencoder. Experiments have been performed to evaluate both reconstruction and kernel alignment performance. Additionally, we applied our method to emulate kPCA on a denoising task obtaining promising results.

Keywords: Autoencoders · Kernel methods · Deep learning · Representation learning

1 Introduction

Autoencoders (AEs) are a class of neural networks that gained increasing interest in recent years [18, 23, 25]. AEs are used for unsupervised learning of *effective* hidden representations of input data [3, 11]. These representations should capture the information contained in the input data, while providing meaningful features for tasks such as clustering and classification [2]. However, what an *effective* representation consists of is highly dependent on the target task.

In standard AEs, representations are derived by training the network to reconstruct inputs through either a bottleneck layer, thereby forcing the network to learn how to compress input information, or through an over-complete representation. In the latter, regularization methods are employed to, e.g., enforce sparse representations, make representations robust to noise, or penalize sensitivity of the representation to small changes in the input [2]. However, regularization provides limited control over the nature of the hidden representation.

In this paper, we hypothesize that an *effective* hidden representation should capture the relations among inputs, which are encoded in form of a kernel matrix. Such a matrix is used as a prior to be reproduced by inner products of the hidden representations learned by the AE. Hence, in addition to minimizing the reconstruction loss, we also minimize the normalized Frobenius distance between the prior kernel matrix and the inner product matrix of the hidden representations. We note that this process resembles the kernel alignment procedure [26].

The proposed model, called *deep kernelized autoencoder*, is related to recent attempts to bridge the performance gap between kernel methods and neural networks [5, 27]. Specifically, it is connected to works on interpreting neural networks from a kernel perspective [21] and the Information Theoretic-Learning Auto-Encoder [23], which imposes a prior distribution over the hidden representation in a variational autoencoder [18].

In addition to providing control over the hidden representation, our method also has several benefits that compensate for important drawbacks of traditional kernel methods. During training, we learn an explicit approximate mapping function from the input to a kernel space, as well as the associated back-mapping to the input space, through an end-to-end learning procedure. Once the mapping is learned, it can be used to relate operations performed in the approximated kernel space, for example linear methods (as is the case of kernel methods), to the input space. In the case of linear methods, this is equivalent to performing non-linear operations on the non-transformed data. Mini-batch training is used in our proposed method in order to lower the computational complexity inherent to traditional kernel methods and, especially, spectral methods [4, 15, 24]. Additionally, our method applies to arbitrary kernel functions, even the ones computed through ensemble methods. To stress this fact, we consider in our experiments the probabilistic cluster kernel, a kernel function that is robust with regards to hyperparameter choices and has been shown to often outperform counterparts such as the RBF kernel [14].

2 Background

2.1 Autoencoders and Stacked Autoencoders

AEs simultaneously learn two functions. The first one, *encoder*, provides a mapping from an input domain, \mathcal{X} , to a code domain, \mathcal{C} , i.e., the hidden representation. The second function, *decoder*, maps from \mathcal{C} back to \mathcal{X} . For a single hidden layer AE, the encoding function $E(\cdot; \mathbf{W}_E)$ and the decoding function $D(\cdot; \mathbf{W}_D)$ are defined as

$$\begin{aligned} \mathbf{h} &= E(\mathbf{x}; \mathbf{W}_E) = \sigma(\mathbf{W}_E \mathbf{x} + \mathbf{b}_E) \\ \tilde{\mathbf{x}} &= D(\mathbf{h}; \mathbf{W}_D) = \sigma(\mathbf{W}_D \mathbf{h} + \mathbf{b}_D), \end{aligned} \tag{1}$$

where $\sigma(\cdot)$ denotes a suitable transfer function (e.g., a sigmoid applied component-wise), \mathbf{x} , \mathbf{h} , and $\tilde{\mathbf{x}}$ denote, respectively, a sample from the input space, its hidden representation, and its reconstruction; finally, \mathbf{W}_E and \mathbf{W}_D are

the weights and \mathbf{b}_E and \mathbf{b}_D the bias of the encoder and decoder, respectively. For the sake of readability, we implicitly incorporate $\mathbf{b}_E, \mathbf{b}_D$ in the notation. Accordingly, we can rewrite

$$\tilde{\mathbf{x}} = D(E(\mathbf{x}; \mathbf{W}_E); \mathbf{W}_D). \quad (2)$$

In order to minimize the discrepancy between the original data and its reconstruction, the parameters in Eq. 1 are typically learned by minimizing, usually through stochastic gradient descent (SGD), a reconstruction loss

$$L_r(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|_2^2. \quad (3)$$

Differently from Eq. 1, a stacked autoencoder (sAE) consists of several hidden layers [11]. Deep architectures are capable of learning complex representations by transforming input data through multiple layers of nonlinear processing [2]. The optimization of the weights is harder in this case and pretraining is beneficial, as it is often easier to learn intermediate representations, instead of training the whole architecture end-to-end [3]. A very important application of pretrained sAE is the initialization of layers in deep neural networks [25]. Pretraining is performed in different phases, each of which consists of training a single AE. After the first AE has been trained, its encoding function $E(\cdot; \mathbf{W}_E^{(1)})$ is applied to the input and the resulting representation is used to train the next AE in the stacked architecture. Each layer, being trained independently, aims at capturing more abstract features by trying to reconstruct the representation in the previous layer. Once all individual AEs are trained, they are unfolded yielding a pretrained sAE. For a two-layer sAE, the encoding function consists of $E(E(\mathbf{x}; \mathbf{W}_E^{(1)}); \mathbf{W}_E^{(2)})$, while the decoder reads $D(D(\mathbf{h}; \mathbf{W}_D^{(2)}); \mathbf{W}_D^{(1)})$. The final sAE architecture can then be fine-tuned end-to-end by back-propagating the gradient of the reconstruction error.

2.2 A Brief Introduction to Relevant Kernel Methods

Kernel methods process data in a kernel space \mathcal{K} associated with an input space \mathcal{X} through an implicit (non-linear) mapping $\phi : \mathcal{X} \rightarrow \mathcal{K}$. There, data are more likely to become separable by linear methods [6], which produces results that are otherwise only obtainable by nonlinear operations in the input space. Explicit computation of the mapping $\phi(\cdot)$ and its inverse $\phi^{-1}(\cdot)$ is, in practice, not required. In fact, operations in the kernel space are expressed through inner products (kernel trick), which are computed as Mercer kernel functions in input space: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$.

As a major drawback, kernel methods scale poorly with the number of data points n : traditionally, memory requirements of these methods scale with $\mathcal{O}(n^2)$ and computation with $\mathcal{O}(n^2 \times d)$, where d is the dimension [8]. For example, kernel principal component analysis (kPCA) [24], a common dimensionality reduction technique that projects data into the subspace that preserves the maximal amount of variance in kernel space, requires to compute the eigendecomposition of a kernel matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$, with $K_{ij} = \kappa(x_i, x_j), x_i, x_j \in \mathcal{X}$, yielding a

computational complexity $\mathcal{O}(n^3)$ and memory requirements that scale as $\mathcal{O}(n^2)$. For this reason, kPCA is not applicable to large-scale problems. The availability of efficient (approximate) mapping functions, however, would reduce the complexity, thereby enabling these methods to be applicable on larger datasets [5]. Furthermore, by providing an approximation for $\phi^{-1}(\cdot)$, it would be possible to directly control and visualize data represented in \mathcal{K} . Finding an explicit inverse mapping from \mathcal{K} is a central problem in several applications, such as image denoising performed with kPCA, also known as the pre-image problem [1, 13].

2.3 Probabilistic Cluster Kernel

The Probabilistic Cluster Kernel (PCK) [14] adapts to inherent structures in the data and it does not depend on any critical user-specified hyperparameters, like the width in Gaussian kernels. The PCK is trained by fitting multiple Gaussian Mixture Models (GMMs) to input data and then combining these models into a single kernel. In particular, GMMs are trained for a variety of mixture components $g = 2, 3, \dots, G$, each with different randomized initial conditions $q = 1, 2, \dots, Q$. Let $\pi_i(q, g)$ denote the *posterior distribution* for data point \mathbf{x}_i under a GMM with g mixture components and initial condition q . The PCK is then defined as

$$\kappa_{\text{PCK}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{Z} \sum_{q=1}^Q \sum_{g=2}^G \pi_i^T(q, g) \pi_j(q, g), \quad (4)$$

where Z is a normalizing constant.

Intuitively, the posterior distribution under a mixture model contains probabilities that a given data point belongs to a certain mixture component in the model. Thus, the inner products in Eq. 4 are large if data pairs often belong to the same mixture component. By averaging these inner products over a range of G values, the kernel function has a large value only if these data points are similar on both global scale (small G) and local scale (large G).

3 Deep Kernelized Autoencoders

In this section, we describe our contribution, which is a method combining AEs with kernel methods: the deep kernelized AE (dkAE). A dkAE is trained by minimizing the following loss function

$$L = (1 - \lambda)L_r(\mathbf{x}, \tilde{\mathbf{x}}) + \lambda L_c(\mathbf{C}, \mathbf{P}), \quad (5)$$

where $L_r(\cdot, \cdot)$ is the reconstruction loss in Eq. 3. λ is a hyperparameter ranging in $[0, 1]$, which weights the importance of the two objectives in Eq. 5. For $\lambda = 0$, the loss function simplifies to the traditional AE loss in Eq. 2. $L_c(\cdot, \cdot)$ is the code loss, a distance measure between two matrices, $\mathbf{P} \in \mathbb{R}^{n \times n}$, the kernel matrix given as prior, and $\mathbf{C} \in \mathbb{R}^{n \times n}$, the inner product matrix of codes associated to

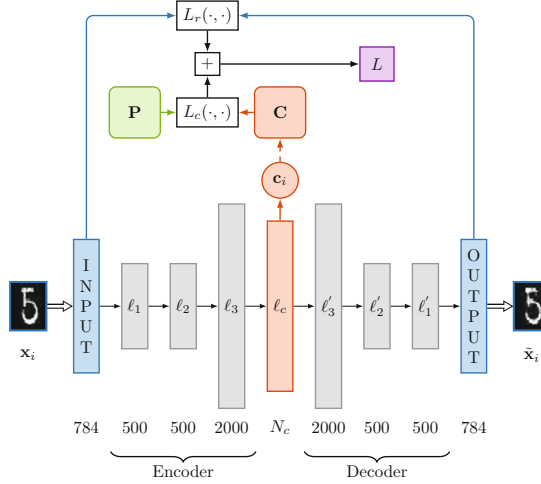


Fig. 1. Schematic illustration of dkAE architecture. Loss function L depends on two terms. First, $L_r(\cdot, \cdot)$, is the reconstruction error between true input \mathbf{x}_i and output of dkAE, $\hat{\mathbf{x}}_i$. Second term, $L_c(\cdot, \cdot)$, is the distance measure between matrices \mathbf{C} (computed as inner products of codes $\{\mathbf{c}_i\}_{i=1}^n$) and the target prior kernel matrix \mathbf{P} . For mini-batch training matrix \mathbf{C} is computed over the codes of the data in the mini-batch and that distance is compared to the submatrix of \mathbf{P} related to the current mini-batch.

input data. The objective of $L_c(\cdot, \cdot)$ is to enforce the similarity between \mathbf{C} and the prior \mathbf{P} . A depiction of the training procedure is reported in Fig. 1.

We implement $L_c(\cdot, \cdot)$ as the normalized Frobenius distance between \mathbf{C} and \mathbf{P} . Each matrix element C_{ij} in \mathbf{C} is given by $C_{ij} = E(\mathbf{x}_i) \cdot E(\mathbf{x}_j)$ and the code loss is computed as

$$L_c(\mathbf{C}, \mathbf{P}) = \left\| \frac{\mathbf{C}}{\|\mathbf{C}\|_F} - \frac{\mathbf{P}}{\|\mathbf{P}\|_F} \right\|_F. \quad (6)$$

Minimizing the normalized Frobenius distance between the kernel matrices is equivalent to maximizing the traditional kernel alignment cost, since

$$\left\| \frac{\mathbf{C}}{\|\mathbf{C}\|_F} - \frac{\mathbf{P}}{\|\mathbf{P}\|_F} \right\|_F = \sqrt{2 - 2A(\mathbf{C}, \mathbf{P})}, \quad (7)$$

where $A(\mathbf{C}, \mathbf{P}) = \frac{\langle \mathbf{C}, \mathbf{P} \rangle_F}{\|\mathbf{C}\|_F \|\mathbf{P}\|_F}$ is exactly the kernel alignment cost function [7, 26]. Note that the distance in Eq. 7 can be implemented also with more advanced differentiable measures of (dis)similarity between PSD matrices, such as divergence and mutual information [9, 19]. However, these options are not explored in this paper and are left for future research.

In this paper, the prior kernel matrix \mathbf{P} is computed by means of the PCK algorithm introduced in Sect. 2.3, such that $\mathbf{P} = \mathbf{K}_{\text{PCK}}$. However, our approach is general and *any* kernel matrix can be used as prior in Eq. 6.

3.1 Mini-Batch Training

We use mini batches of k samples to train the dkAE, thereby avoiding the computational restrictions of kernel and especially spectral methods outlined in Sect. 2.2. Making use of mini-batch training, the memory complexity of the algorithm can be reduced to $O(k^2)$, where $k \ll n$. Finally, we note that the computational complexity scales linearly with regards to the parameters in the network. In particular, given a mini batch of k samples, the dkAE loss function is defined by taking the average of the per-sample reconstruction cost

$$L_{\text{batch}} = \frac{1 - \lambda}{kd} \sum_{i=1}^k L_r(\mathbf{x}_i, \tilde{\mathbf{x}}_i) + \lambda \left\| \frac{\mathbf{C}_k}{\|\mathbf{C}_k\|_F} - \frac{\mathbf{P}_k}{\|\mathbf{P}_k\|_F} \right\|_F, \quad (8)$$

where d is the dimensionality of the input space, \mathbf{P}_k is a subset of \mathbf{P} that contains only the k rows and columns related to the current mini-batch, and \mathbf{C}_k contains the inner products of the codes for the specific mini-batch. Note that \mathbf{C}_k is re-computed in each mini batch.

3.2 Operations in Code Space

Linear operations in code space can be performed as shown in Fig. 2. The encoding scheme of the proposed dkAE explicitly approximates the function $\phi(\cdot)$ that maps an input \mathbf{x}_i onto the kernel space. In particular, in a dkAE the feature vector $\phi(\mathbf{x}_i)$ is approximated by the code \mathbf{c}_i . Following the underlying idea of kernel methods and inspired by Cover’s theorem [6], which states that a high dimensional embedding is more likely to be linearly separable, linear operations can be performed on the code. A linear operation on \mathbf{c}_i produces a result in the code space, \mathbf{z}_i , relative to the input \mathbf{x}_i . Codes are mapped back to the input space by means of a decoder, which in our case approximates the inverse mapping $\phi(\cdot)^{-1}$ from the kernel space back to the input domain. Unlike other kernel methods where this explicit mapping is not defined, this fact permits visualization and interpretation of the results in the original space.

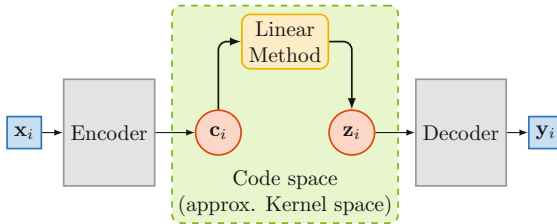


Fig. 2. The encoder maps input \mathbf{x}_i to \mathbf{c}_i , which lies in code space. In dkAEs, the code domain approximates the space associated to the prior kernel \mathbf{P} . A linear method receives input \mathbf{c}_i and produces output \mathbf{z}_i . The decoder maps \mathbf{z}_i back to input space. The result \mathbf{y}_i can be seen as the output of a non-linear operation on \mathbf{x}_i in input space.

4 Experiments and Results

In this section, we evaluate the effectiveness of dkAEs on different benchmarks. In the first experiment we evaluate the effect on the two terms of the objective function (Eq. 8) when varying the hyperparameters λ (in Eq. 5) and the size of the code layer. In a second experiment, we study the reconstruction and the kernel alignment. Further we compare dkAEs approximation accuracy of the prior kernel matrix to kPCA as the number of principle components increases. Finally, we present an application of our method for image denoising, where we apply PCA in the dkAE code space \mathcal{C} to remove noise.

For these experiments, we consider the MNIST dataset, consisting of 60000 images of handwritten digits. However, we use a subset of 20000 samples due to the computational restrictions imposed by the PCK, which we use to illustrate dkAEs ability to learn arbitrary kernels, even if they originate from an ensemble procedure. We train the PCK by fitting the GMMs on a subset of 200 training samples, with the parameters $Q = G = 30$. Once trained, the GMM models are applied on the remaining data to calculate the kernel matrix. We use 70%, 15% and 15% of the data for training, validation, and testing, respectively.

4.1 Implementation

The network architecture used in the experiments is $d - 500 - 500 - 2000 - N_c$ (see Fig. 1), which has been demonstrated to perform well on several datasets, including MNIST, for both supervised and unsupervised tasks [12, 20]. Here, N_c refers to the dimensionality of the code layer. Training was performed using the sAE pretraining approach outlined in Sect. 2.1. To avoid learning the identify mapping on each individual layer, we applied a common [16] regularization technique where the encoder and decoder weights are tied, i.e., $W_E = W_D^T$. This is done during pretraining and fine-tuning. Unlike in traditional sAEs, to account for the kernel alignment objective, the code layer is optimized according to Eq. 5 *also* during pretraining.

Size of mini-batches for training was chosen to be $k = 200$ randomly, independently sampled data points; in our experiments, an epoch consists of processing $(n/k)^2$ batches. Pretraining is performed for 30 epochs per layer and the final architecture is fine-tuned for 100 epochs using gradient descent based on Adam [17]. The dkAE weights are randomly initialized according to Glorot et al. [10].

4.2 Influence of Hyperparameter λ and Size N_c of Code Layer

In this experiment, we evaluate the influence of the two main hyperparameters that determine the behaviour of our architecture. Note that the experiments shown in this section are performed by training the dkAE on the training set and evaluating the performance on the validation set. We evaluate both the out-of-sample reconstruction L_r and L_c . Figure 3(a) illustrates the effect of λ for a fixed value $N_c = 2000$ of neurons in the code layer. It can be observed

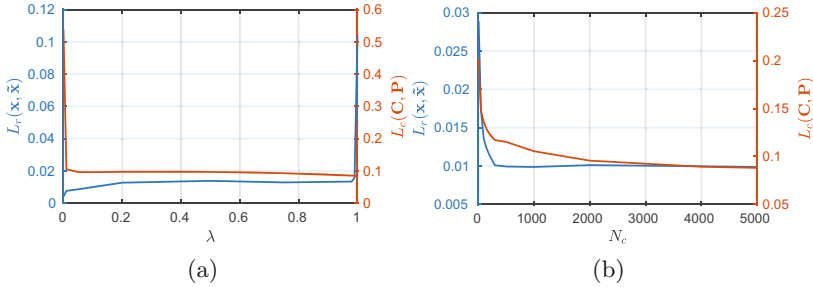


Fig. 3. (a): Tradeoff when choosing λ . High λ values result in low L_c , but high reconstruction cost, and vice-versa. (b): Both L_c and reconstruction costs decrease when code dimensionality N_c increases.

that the reconstruction loss L_r increases as more and more focus is put on minimizing L_c (obtained by increasing λ). This quantifies empirically the tradeoff in optimizing the reconstruction performance and the kernel alignment at the same time. Similarly, it can be observed that L_c decreases when increasing λ . By inspecting the results, specifically the near constant losses for λ in range $[0.1, 0.9]$ the method appears robust to changes in hyperparameter λ .

Analyzing the effect of varying N_c given a fixed $\lambda = 0.1$ (Fig. 3(b)), we observe that both losses decrease as N_c increases. This could suggest that an even larger architecture, characterized by more layers and more neurons w.r.t. the architecture adopted, might work well, as the dkAE does not seem to overfit, due also to the regularization effect provided by the kernel alignment.

4.3 Reconstruction and Kernel Alignment

According to the previous results, in the following experiments we set $\lambda = 0.1$ and $N_c = 2000$. Figure 4 illustrates the results in Sect. 4.2 qualitatively by displaying a set of original images from our test set and their reconstruction for the chosen λ value and a non-optimal one. Similarly, the prior kernel (sorted by class in the figure, to ease the visualization) and the dkAEs approximated kernel matrices, relative to test data, are displayed for two different λ values. Notice that, to illustrate the difference with a traditional sAE, one of the two λ values is set to zero. It can be clearly seen that, for $\lambda = 0.1$, both the reconstruction and the kernel matrix, resemble the original closely, which agrees with the plots in Fig. 3(a).

Inspecting the kernels obtained in Fig. 4, we compare the distance between the kernel matrices, \mathbf{C} and \mathbf{P} , and the ideal kernel matrix, obtained by considering supervised information. We build the ideal kernel matrix \mathbf{K}_I , where $K_I(i, j) = 1$ if elements i and j belong to same class, otherwise $K_I(i, j) = 0$. Table 1 illustrates that the kernel approximation produced by dkAE outperforms a traditional sAE with regards to kernel alignment with the ideal kernel. Additionally it can be seen that the kernel approximation actually improves slightly

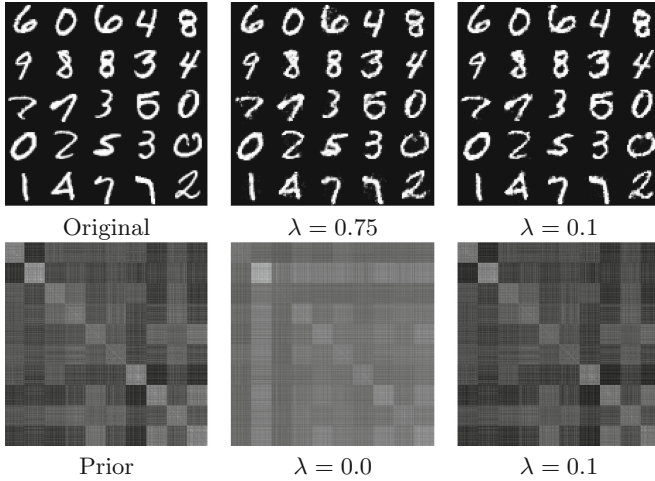


Fig. 4. Illustrating the reconstruction and kernel alignment trade-off for different λ values. We note that the reconstruction for a small λ is generally better (see also Fig. 3(a)), but that small λ yields high L_c .

Table 1. Computing L_c with respect to an ideal kernel matrix \mathbf{K}_I for our test dataset (10 classes) and comparing relative improvement for the three kernels in Fig. 4. Prior kernel \mathbf{P} , a traditional sAE ($\lambda = 0$) \mathbf{K}_{AE} , and dkAEs \mathbf{C} .

Kernel	$L_c(\cdot, \mathbf{K}_I)$	Improvement [%] vs.		
		\mathbf{P}	\mathbf{K}_{AE}	\mathbf{C}
\mathbf{P}	1.0132	0	12.7	-0.2
\mathbf{K}_{AE}	1.1417	-11.3	0	-11.4
\mathbf{C}	1.0115	0.2	12.9	0

on the kernel prior, which we hypothesise is due to the regularization that is imposed by the reconstruction objective.

4.4 Approximation of Kernel Matrix Given as Prior

In order to quantify the kernel alignment performance, we compare dkAE to the approximation provided by kPCA when varying the number of principal components. For this test, we take the kernel matrix \mathbf{P} of the training set and compute its eigendecomposition. We then select an increasing number of components m (with $m \geq 1$ components associated with the largest eigenvalues) to project the input data: $\mathbf{Z}_m = \mathbf{E}_m \mathbf{\Lambda}_m^{1/2}$, $d = 2, \dots, N$. The approximation of the original kernel matrix (prior) is then given as $\mathbf{K}_m = \mathbf{Z}_m \mathbf{Z}_m^T$. We compute the distance between \mathbf{K}_m and \mathbf{P} following Eq. 7 and compare it to dissimilarity between \mathbf{P} and \mathbf{C} . For evaluating the out-of-sample performance, we use the Nyström approximation for kPCA [24] and compare it to the dkAE kernel approximation on the test set.

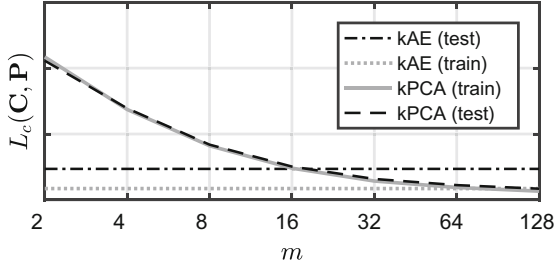


Fig. 5. Comparing dkAEs approximation of the kernel matrix to kPCA for an increasing number of components. The plot shows that dkAE reconstruction is more accurate for low number (i.e., $m < 16$) of components.

Figure 5 shows that the approximation obtained by means of dkAEs outperforms kPCA when using a small number of components, i.e., $m < 16$. Note that it is common in spectral methods to choose a number of components equal to the number of classes in the dataset [22] in which case, for the 10 classes in the MNIST dataset, dkAE would outperform kPCA. As the number of selected components increases, the approximation provided by kPCA will perform better. However, as shown in the previous experiment (Sect. 4.3), this does not mean that the approximation performs better with regards to the ideal kernel. In fact, in that experiment the kernel approximation by dkAE actually performed at least as well as the prior kernel (kPCA with all components taken into account).

4.5 Linear Operations in Code Space

Here we hint at the potential of performing operations in code space as described in Sect. 3.2. We try to emulate kPCA by performing PCA in our learned kernel space and evaluate the performance on the task of denoising. Denoising is a task that requires both a mapping to the kernel space, as well as a back-projection. For traditional kernel methods no explicit back-projection exists, but approximate solutions to this so called pre-image problem have been proposed [1, 13]. We chose the method proposed by Bakir et al. [1], where they use kernel ridge regression, such that a different kernel (in our case an RBF) can be used for the back-mapping. As it was a challenging to find a good σ for the RBF kernel that captures all numbers in the MNIST dataset, we performed this test on the 5 and 6 class only. The regularization parameter and the σ required for the back-projection were found via grid search, where the best regularization parameter (according to MSE reconstruction) was found to be 0.5 and σ as the median of the euclidean distances between the projected feature vectors.

Both models are fitted on the training set and Gaussian noise is added to the test set. For both methods 32 principle components are used. Table 2 illustrates that dkAE+PCA outperforms kPCAs reconstruction with regards to mean squared error. However, as this is not necessarily a good measure for

Table 2. Mean squared error for reconstruction.

Noise std.	kPCA	dkAE+PCA
0.25	0.0427	0.0358

**Fig. 6.** Original images (left), the reconstruction with kPCA (center) and with dkAE+PCA (right).

denoising [1], we also visualize the results in Fig. 6. It can be seen that dkAE yields sharper images in the denoising task.

5 Conclusions

In this paper, we proposed a novel model for autoencoders, based on the definition of a particular unsupervised loss function. The proposed model enables us to learn an approximate embedding from an input space to an arbitrary kernel space as well as the projection from the kernel space back to input space through an end-to-end trained model. It is worth noting that, with our method, we are able to approximate arbitrary kernel functions by inner products in the code layer, which allows us to control the representation learned by the autoencoder. In addition, it enables us to emulate well-known kernel methods such as kPCA and scales well with the number of data points.

A more rigorous analysis of the learned kernel space embedding, as well as applications of the code space representation for clustering and/or classification tasks, are left as future works.

Acknowledgments. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPU used for this research. This work was partially funded by the Norwegian Research Council FRIPRO grant no. 239844 on developing the *Next Generation Learning Machines*.

References

1. Bakir, G.H., Weston, J., Schölkopf, B.: Learning to find pre-images. In: *Advances in Neural Information Processing Systems*, pp. 449–456 (2004)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**(8), 1798–1828 (2013)
3. Bengio, Y.: Learning deep architectures for ai. *Found. Trends Mach. Learn.* **2**(1), 1–127 (2009)
4. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152 (1992)
5. Cho, Y., Saul, L.K.: Kernel methods for deep learning. In: *Advances in Neural Information Processing Systems 22*, pp. 342–350 (2009)

6. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley, New York (1991)
7. Cristianini, N., Elisseeff, A., Shawe-Taylor, J., Kandola, J.: On kernel-target alignment. In: Advances in Neural Information Processing Systems (2001)
8. Dai, B., Xie, B., He, N., Liang, Y., Raj, A., Balcan, M.F.F., Song, L.: Scalable kernel methods via doubly stochastic gradients. In: Advances in Neural Information Processing Systems, pp. 3041–3049 (2014)
9. Giraldo, L.G.S., Rao, M., Principe, J.C.: Measures of entropy from data using infinitely divisible kernels. *IEEE Trans. Inf. Theory* **61**(1), 535–548 (2015)
10. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10) (2010)
11. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. *Science* **313**(5786), 504–507 (2006)
12. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. *Neural Comput.* **18**(7), 1527–1554 (2006)
13. Honeine, P., Richard, C.: A closed-form solution for the pre-image problem in kernel-based machines. *J. Sig. Process. Syst.* **65**(3), 289–299 (2011)
14. Izquierdo-Verdiguier, E., Jenssen, R., Gómez-Chova, L., Camps-Valls, G.: Spectral clustering with the probabilistic cluster kernel. *Neurocomputing* **149**, 1299–1304 (2015)
15. Jenssen, R.: Kernel entropy component analysis. *IEEE Trans. Pattern Anal. Mach. Intell.* **32**(5), 847–860 (2010)
16. Kamyshanska, H., Memisevic, R.: The potential energy of an autoencoder. *IEEE Trans. Pattern Anal. Mach. Intell.* **37**(6), 1261–1273 (2015)
17. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980) (2014)
18. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint [arXiv:1312.6114](https://arxiv.org/abs/1312.6114) (2013)
19. Kulis, B., Sustik, M.A., Dhillon, I.S.: Low-rank kernel learning with Bregman matrix divergences. *J. Mach. Learn. Res.* **10**, 341–376 (2009)
20. Maaten, L.: Learning a parametric embedding by preserving local structure. In: International Conference on Artificial Intelligence and Statistics, pp. 384–391 (2009)
21. Montavon, G., Braun, M.L., Müller, K.R.: Kernel analysis of deep networks. *J. Mach. Learn. Res.* **12**, 2563–2581 (2011)
22. Ng, A.Y., Jordan, M.I., Weiss, Y., et al.: On spectral clustering: analysis and an algorithm. In: Advances in Neural Information Processing Systems, pp. 849–856 (2001)
23. Santana, E., Emigh, M., Principe, J.C.: Information theoretic-learning auto-encoder. arXiv preprint [arXiv:1603.06653](https://arxiv.org/abs/1603.06653) (2016)
24. Schölkopf, B., Smola, A., Müller, K.R.: Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.* **10**(5), 1299–1319 (1998)
25. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.* **11**, 3371–3408 (2010)
26. Wang, T., Zhao, D., Tian, S.: An overview of kernel alignment and its applications. *Artif. Intell. Rev.* **43**(2), 179–192 (2015)
27. Wilson, A.G., Hu, Z., Salakhutdinov, R., Xing, E.P.: Deep kernel learning. In: Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, pp. 370–378 (2016)