Max–Planck–Institut für biologische Kybernetik
Max Planck Institute for Biological Cybernetics

MAX-PLANCK-GESELLSCHAFT

Technical Report No. 109

Kernel Hebbian Algorithm for Iterative Kernel Principal Component Analysis

Kwang In Kim,[1] Matthias O. Franz,[1] Bernhard Schölkopf[1]

June 2003

[1] Department Schölkopf, email: kimki;mof;bs@tuebingen.mpg.de

Kernel Hebbian Algorithm for Iterative Kernel Principal Component Analysis

Kwang In Kim, Matthias O. Franz, Bernhard Schölkopf

Abstract. A new method for performing a kernel principal component analysis is proposed. By kernelizing the generalized Hebbian algorithm, one can iteratively estimate the principal components in a reproducing kernel Hilbert space with only linear order memory complexity. The derivation of the method and preliminary applications in image hyperresolution are presented. In addition, we discuss the extension of the method to the online learning of kernel principal components.

1 Introduction

Kernel Principal Component Analysis (KPCA), a non-linear extension of PCA, is a powerful technique for extracting non-linear structure from data [1]. The basic idea is to map the input data into a *Reproducing Kernel Hilbert Space* (RKHS) and then, to perform PCA in that space. While the direct computation of PCA in a RKHS is in general infeasible due to the high dimensionality of that space, KPCA enables this by using kernel methods [2] and formulating PCA as the equivalent *kernel eigenvalue problem*. A problem of this approach is that it requires to store and manipulate the *kernel matrix* the size of which is square of the number of examples. This becomes computationally expensive when the number of samples is large.

In this work, we adapt the Generalized Hebbian Algorithm (GHA), which was introduced as online algorithm for linear PCA [3, 4], to perform PCA in RKHSs. Expanding the solution of GHA only in inner products of the samples enables us to kernelize the GHA. The resulting *Kernel Hebbian Algorithm* (KHA) estimates the eigenvectors of the kernel matrix with linear order memory complexity.

The capability of the KHA to handle large and high-dimensional datasets will be demonstrated in the context of image hyperresolution where we project the images to be restored (i.e., the low resolution ones) into a space spanned by the kernel principal components of a database of high-resolution images. The hyperresolution image is then constructed from the projections by preimage techniques [5]. The resulting images show a considerably richer structure than those obtained from linear PCA, since higher-order image statistics are taken into account.

This paper is organized as follows. Section 2 briefly introduces PCA, GHA, and KPCA. Section 3 formulates the KHA. Application results for image hyperresolution are presented in Section 4, while conclusions are drawn in Section 5.

2 Background

Principal component analysis. Given a set of centered observations $\mathbf{x}_k = \mathbb{R}^N$, $k = 1, \ldots, l$, and $\sum_{k=1}^{l} \mathbf{x}_k = 0$, PCA diagonalizes the covariance matrix

$$\overline{C} = \frac{1}{l} \sum_{j=1}^{l} \mathbf{x}_j \mathbf{x}_j^\top .^1$$

This is readily performed by solving the eigenvalue equation

$$\lambda \mathbf{v} = \overline{C} \mathbf{v}$$

for eigenvalues $\lambda \geq 0$ and eigenvectors $\mathbf{v}_i \in \mathbb{R}^N \setminus 0$.

[1]More precisely, the covariance matrix is defined as the $E[\mathbf{x}\mathbf{x}^\top]$; $\overline{C}$ is an estimate based on a finite set of examples.

Generalized Hebbian algorithm. From a computational point of view, it can be advantageous to solve the eigenvalue problem by iterative methods which do not need to compute and store $\overline{C}$ directly. This is particulary useful when the size of $\overline{C}$ is large such that the memory complexity becomes prohibitive. Among the existing iterative methods for PCA, the generalized Hebbian algorithm (GHA) is of particular interest, since it does not only provide a memory-efficient implementation but also has the inherent capability to adapt to time-varying distributions.

Let us define a matrix $\mathbf{W}(t) = (\mathbf{w}_1(t)^\top, \dots, \mathbf{w}_r(t)^\top)^\top$, where r is the number of eigenvectors considered and $\mathbf{w}_i(t) \in \mathbb{R}^N$. Given a random initialization of $\mathbf{W}(0)$, the GHA applies the following recursive rule

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \eta(t)(\mathbf{y}(t)\mathbf{x}(t)^\top - \mathrm{LT}[\mathbf{y}(t)\mathbf{y}(t)^\top]\mathbf{W}(t)), \tag{1}$$

where $\mathbf{x}(t)$ is a randomly selected pattern from l input examples, presented at time t, $\mathbf{y}(t) = \mathbf{W}(t)\mathbf{x}(t)$, and $\mathrm{LT}[\cdot]$ sets all elements above the diagonal of its matrix argument to zero, thereby making it lower triangular. It was shown in [3] for $i = 1$ and in [4] for $i > 1$ that $\mathbf{W}(t) \to \mathbf{V}_i$ as $t \to \infty$.[2] For a detailed discussion of the GHA, readers are referred to [4].

Kernel principal component analysis. When the data of interest are highly nonlinear, linear PCA fails to capture the underlying structure. As a nonlinear extension of PCA, KPCA computes the principal components in a possibly high-dimensional *Reproducing Kernel Hilbert Space* (RKHS) F which is related to the input space by a nonlinear map $\Phi : \mathbb{R}^N \to F$ [2]. An important property of a RKHS is that the inner product of two points mapped by Φ can be evaluated using *kernel functions*

$$k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}), \tag{2}$$

which allows us to compute the value of the inner product without having to carry out the map Φ explicitly. Since PCA can be formulated in terms of inner products, we can compute it also implicitly in a RKHS. Assuming that the data are centered in F (i.e., $\sum_{k=1}^{l} \Phi(\mathbf{x}_k) = 0$)[3] the covariance matrix takes the form

$$C = \frac{1}{l}\mathbf{\Phi}^\top\mathbf{\Phi}, \tag{3}$$

where $\mathbf{\Phi} = \left(\Phi(\mathbf{x}_1)^\top, \dots, \Phi(\mathbf{x}_l)^\top\right)^\top$. We now have to find the eigenvalues $\lambda \geq 0$ and eigenvectors $\mathbf{v} \in F \setminus 0$ satisfying

$$\lambda\mathbf{v} = C\mathbf{v}. \tag{4}$$

Since all solutions $\mathbf{v}$ with $\lambda \neq 0$ lie within the span of $\{\Phi(\mathbf{x}_1), \dots, \Phi(\mathbf{x}_l)\}$ [1], we may consider the following equivalent problem

$$\lambda\mathbf{\Phi}\mathbf{v} = \mathbf{\Phi}C\mathbf{v}, \tag{5}$$

and we may represent $\mathbf{v}$ in terms of an l-dimensional vector $\mathbf{q}$ as $\mathbf{v} = \mathbf{\Phi}^\top\mathbf{q}$. Combining this with (3) and (5) and defining an $l \times l$ kernel matrix $\mathbf{K}$ by $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top$ leads to $l\lambda\mathbf{K}\mathbf{q} = \mathbf{K}^2\mathbf{q}$. The solution can be obtained by solving the *kernel eigenvalue problem* [1]

$$l\lambda\mathbf{q} = \mathbf{K}\mathbf{q}. \tag{6}$$

It should be noted that the size of the kernel matrix scales with the square of the number of examples. Thus, it becomes computationally infeasible to solve directly the kernel eigenvalue problem for large number of examples. This motivates the introduction of the Kernel Hebbian Algorithm presented in the next section. For more detailed discussions on PCA and KPCA, including the issue of computational complexity, readers are referred to [2].

3 Kernel Hebbian algorithm

3.1 GHA in RKHSs and its kernelization

The GHA update rule of Eq. (1) is represented in the RKHS F as

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \eta(t)\left(\mathbf{y}(t)\Phi(\mathbf{x}(t))^\top - \mathrm{LT}[\mathbf{y}(t)\mathbf{y}(t)^\top]\mathbf{W}(t)\right), \tag{7}$$

[2]Originally it has been shown that $\mathbf{w}_i$ converges to the i-th eigenvector of $E[\mathbf{x}\mathbf{x}^\top]$, given an infinite sequence of examples. By replacing each $\mathbf{x}(t)$ with a random selection $\mathbf{x}_i$ from a finite training set, we obtain the above statement.

[3]The centering issue will be dealt with later.

where the rows of $\mathbf{W}(t)$ are now vectors in F and $\mathbf{y}(t) = \mathbf{W}(t)\Phi(\mathbf{x}(t))$. $\Phi(\mathbf{x}(t))$ is a pattern presented at time t which is randomly selected from the mapped data points $\{\Phi(\mathbf{x}_1), \ldots, \Phi(\mathbf{x}_l)\}$. For notational convenience we assume that there is a function $J(t)$ which maps t to $i \in \{1, \ldots, l\}$ ensuring $\Phi(\mathbf{x}(t)) = \Phi(\mathbf{x}_i)$. From the direct KPCA solution, it is known that $\mathbf{w}(t)$ can be expanded in the mapped data points $\Phi(\mathbf{x}_i)$. This restricts the search space to linear combinations of the $\Phi(\mathbf{x}_i)$ such that $\mathbf{W}(t)$ can be expressed as

$$\mathbf{W}(t) = \mathbf{A}(t)\mathbf{\Phi} \tag{8}$$

with an $r \times l$ matrix $\mathbf{A}(t) = (\mathbf{a}_1(t)^\top, \ldots, \mathbf{a}_r(t)^\top)^\top$ of expansion coefficients. The ith row $\mathbf{a}_i = (a_{i1}, \ldots, a_{il})$ of $\mathbf{A}(t)$ corresponds to the expansion coefficients of the ith eigenvector of $\mathbf{K}$ in the $\Phi(\mathbf{x}_i)$, i.e., $\mathbf{w}_i(t) = \mathbf{\Phi}^\top \mathbf{a}_i(t)$. Using this representation, the update rule becomes

$$\mathbf{A}(t+1)\mathbf{\Phi} = \mathbf{A}(t)\mathbf{\Phi} + \eta(t)\left(\mathbf{y}(t)\Phi(\mathbf{x}(t))^\top - \mathrm{LT}[\mathbf{y}(t)\mathbf{y}(t)^\top]\mathbf{A}(t)\mathbf{\Phi}\right). \tag{9}$$

The mapped data points $\Phi(\mathbf{x}(t))$ can be represented as $\Phi(\mathbf{x}(t)) = \mathbf{\Phi}^\top \mathbf{b}(t)$ with a canonical unit vector $\mathbf{b}(t) = (0, \ldots, 1, \ldots, 0)^\top$ in $\mathbb{R}^l$ (only the $J(t)$-th element is 1). Using this notation, the update rule can be written solely in terms of the expansion coefficients as

$$\mathbf{A}(t+1) = \mathbf{A}(t) + \eta(t)\left(\mathbf{y}(t)\mathbf{b}(t)^\top - \mathrm{LT}[\mathbf{y}(t)\mathbf{y}(t)^\top]\mathbf{A}(t)\right). \tag{10}$$

Representing (10) in component-wise form gives

$$a_{ij}(t+1) = \begin{cases} a_{ij}(t) + \eta y_i(t) - \eta y_i(t)\sum_{k=1}^{i} a_{kj}(t)y_k(t) & \text{if} \quad J(t) = j \\ a_{ij}(t) - \eta y_i(t)\sum_{k=1}^{i} a_{kj}(t)y_k(t) & \text{otherwise,} \end{cases} \tag{11}$$

where

$$y_i(t) = \sum_{k=1}^{l} a_{ik}(t)\Phi(\mathbf{x}_k) \cdot \Phi(\mathbf{x}(t)) = \sum_{k=1}^{l} a_{ik}(t)k(\mathbf{x}_k, \mathbf{x}(t)),. \tag{12}$$

This does not require $\Phi(\mathbf{x})$ in explicit form and accordingly provides a practical implementation of the GHA in F.

During the derivation of (10), it was assumed that the data are centered in F which is not true in general unless explicit centering is performed. Centering can be done by subtracting the mean of the data from each pattern. Then each pattern $\Phi(\mathbf{x}(t))$ is replaced by $\tilde{\Phi}(\mathbf{x}(t)) \doteq \Phi(\mathbf{x}(t)) - \overline{\Phi}(\mathbf{x})$, where $\overline{\Phi}(\mathbf{x})$ is the sample mean $\overline{\Phi}(\mathbf{x}) = \frac{1}{l}\sum_{k=1}^{l}\Phi(\mathbf{x}_k)$. The centered algorithm remains the same as in (11) except that Eq. (12) has to be replaced by the more complicated expression

$$y_i(t) = \sum_{k=1}^{l} a_{ik}(t)(k(\mathbf{x}(t), \mathbf{x}_k) - \bar{k}(\mathbf{x}_k)) - \overline{a}_i(t)\sum_{k=1}^{l}(k(\mathbf{x}(t), \mathbf{x}_k) + \bar{k}(\mathbf{x}_k)). \tag{13}$$

with $\bar{k}(\mathbf{x}_k) = \frac{1}{l}\sum_{m=1}^{l} k(\mathbf{x}_m, \mathbf{x}_k)$ and $\overline{a}_i(t) = \frac{1}{l}\sum_{m=1}^{l} a_{im}(t)$. It should be noted that not only in training but also in testing, each pattern should be centered, using the training mean.

Now we state the convergence properties of the KHA (Eq. 7) as a theorem:

Theorem 1 *For a finite set of centered data (presented infinitely often) and $\mathbf{A}$ initially in general position,*[4] *(7) (and equivalently (10)) will converge with probability 1,*[5] *and the rows of $\mathbf{W}$ will approach the first r normalized eigenvectors of the correlation matrix C in the RKHS, ordered by decreasing eigenvalue.*

The proof of theorem 1 is straightforward if we note that for a finite set of data $\{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$, we can induce from a given kernel k, an *kernel PCA map* [2])

$$\Phi_l : \mathbf{x} \to \mathbf{K}^{-\frac{1}{2}}(k(\mathbf{x}, \mathbf{x}_1), \ldots, k(\mathbf{x}, \mathbf{x}_l))$$

satisfying

$$\Phi_l(\mathbf{x}_i) \cdot \Phi_l(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j).$$

By applying the GHA in the space spanned by the kernel PCA map, (i.e., replacing each occurrence of $\Phi(\mathbf{x})$ with $\Phi_l(\mathbf{x})$ in Eq. 7, and noting that this time, $\mathbf{W}$ lies in $\mathbb{R}^l$ rather than in F), we obtain an algorithm in $\mathbb{R}^l$ which is exactly equivalent to the KHA in F. The convergence of the KHA then follows from the convergence of the GHA in $\mathbb{R}^l$. It should be noted that, in practice, this approach cannot be taken to construct an iterative algorithm since it involves the computation of $\mathbf{K}^{-\frac{1}{2}}$.

[4] i.e., $\mathbf{A}$ is neither the zero vector nor orthogonal to the eigenvectors.

[5] Assuming that the input data is not always orthogonal to the initialization of $\mathbf{A}$.

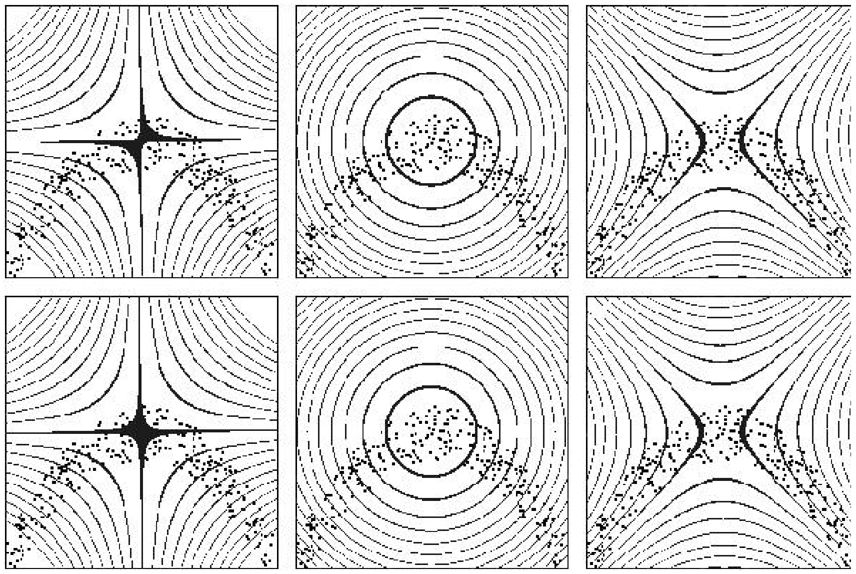

Figure 1: Two dimensional examples, with data generated in the following way: x-values have uniform distribution in $[-1, 1]$, y-values are generated from $y_i = -x_i^2 + \xi$, where ξ is normal noise with standard deviation 0.2. From left to right, contour lines of constant value of the first three PCs obtained from KPCA and KHA with degree-2 polynomial kernel.

4 Experiments

4.1 Toy example

Figure 1 shows the first three PCs of a toy data set, extracted by KPCA and KHA with a polynomial kernel. Visual similarity of PCs from both algorithms show the approximation capability of KHA to KPCA.

4.2 Image hyperresolution

The problem of image hyperresolution is to reconstruct a high resolution image based on one or several low resolution images. The former case, which we are interested in, requires prior knowledge about the image class to be reconstructed. In our case, we encode the prior knowledge in the kernel principal components of a large image database. In contrast to linear PCA, KPCA is capable of capturing part of the higher-order statistics which are particularly important for encoding image structure [8]. Capturing these higher-order statistics, however, can require a large number of training examples, particularly for larger image sizes and complex image classes such as patches taken from natural images. This causes problems for KPCA, since KPCA requires to store and manipulate the kernel matrix the size of which is the square of the number of examples, and necessitates the KHA.

To reconstruct a hyperresolution image from a low-resolution image which was *not* contained in the training set, we first scale up the image to the same size as the training images, then map the image (call it $\mathbf{x}$) into the RKHS F using Φ, and project it into the KPCA subspace corresponding to a limited number of principal components to get $P\Phi(\mathbf{x})$. Via the projection P, the image is mapped to an image which is consistent with the statistics of the high-resolution training images. However, at that point, the projection still lives in F, which can be infinite-dimensional. We thus need to find a corresponding point in $\mathbb{R}^N$ — this is a preimage problem. To solve it, we minimize $\|P\Phi(\mathbf{x}) - \Phi(\mathbf{z})\|^2$ over $\mathbf{z} \in \mathbb{R}^N$. Note that this objective function can be computed in terms of inner products and thus in terms of the kernel (2). For the minimization, we use gradient descent [9] with starting points obtained using the method of [10]. There is a large number of surveys and detailed treatments of image hyperresolution; we exemplarily refer the reader to [11].

Hyperresolution of face images. Here we consider a large database of detailed face images. The direct computation of KPCA for this dataset is not practical on standard hardware. The Yale Face Database B contains 5760 images of 10 persons [12]. 5,000 images were used for training while 10 randomly selected images which are disjoint from the training set were used to test the method (note, however, as there are only 10 persons in the database, the same person, in different views, is likely to occur in training and test set). For training, (60×60)-sized face images were fed into linear PCA and KHA. Then, the test images were subsampled to a 20×20 grid and scaled

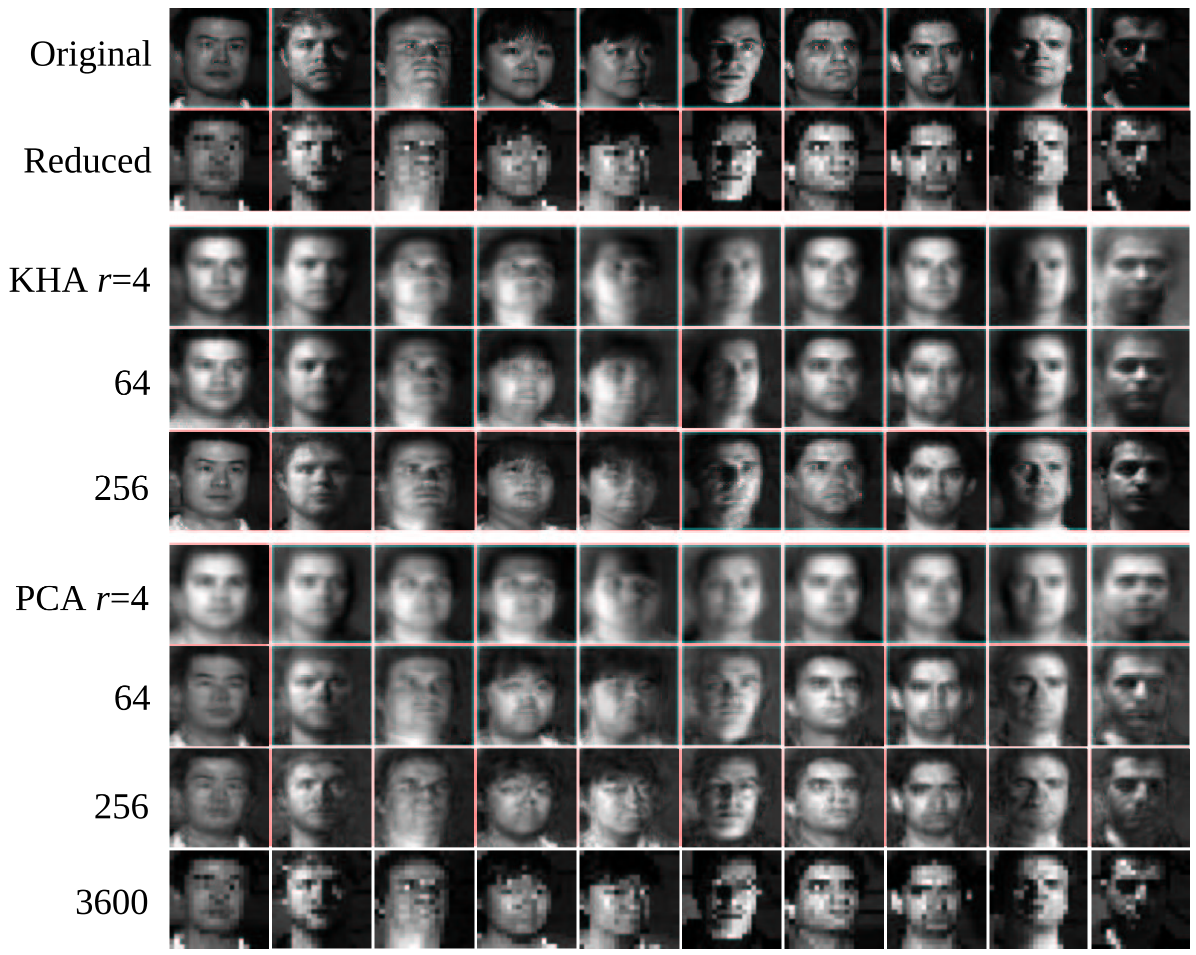

Figure 2: Face reconstruction based on PCA and KHA using a Gaussian kernel $(k(\mathbf{x},\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|^2/(2\sigma^2)))$ with $\sigma = 1$ for varying numbers of principal components. The images can be examined in detail at http://www.kyb.tuebingen.mpg.de/~kimki

up to the original scale (60×60) by turning each pixel into a 3×3 square of identical pixels, before doing the reconstruction. Figure 2 shows reconstruction examples obtained using different numbers of components. While the images obtained from linear PCA look like somewhat uncontrolled superpositions of different face images, the images obtained from its nonlinear counterpart (KHA) are more face-like. In spite of its less realistic results, linear PCA was slightly better than the KHA in terms of the mean squared error (average 9.20 and 8.48 for KHA and PCA, respectively for 100 principal components). This stems from the characteristics of PCA which is constructed to minimize the MSE, while KHA is not concerned with MSE in the input space. Instead, it seems to force the images to be contained in the manifold of face images. Similar observations have been reported by [13].

Interestingly, when the number of examples is small and the sampling of this manifold is sparse, this can have the consequence that the optimal KPCA (or KHA) reconstruction is an image that looks like the face of a wrong person. In a sense, this means that the errors performed by KPCA are errors *along* the manifold of faces. Figure 3 demonstrates this effect by comparing results from KPCA on 1000 example images (corresponding to a sparse sampling of the face manifold) and KHA on 5000 training images (denser sampling). As the examples shows, some of the misreconstructions that are made by KPCA due to the lack of training examples were corrected by the KHA using a large training set.

Hyperresolution of natural images. Figure 4 shows the first 40 principal components of 40,000 natural image patches obtained from the KHA using a Gaussian kernel. The image database was obtained from [14]. Again,

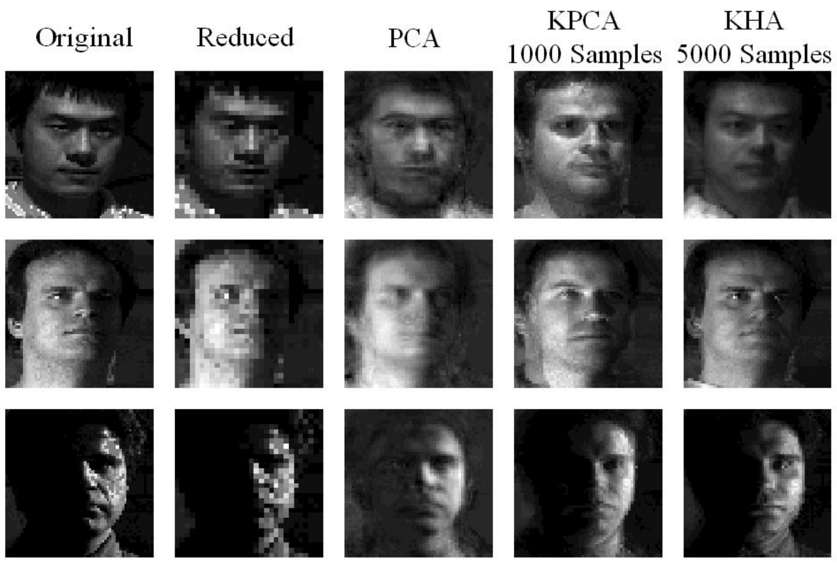

Figure 3: Face reconstruction examples (from 30×30 resolution) obtained from KPCA and KHA trained on 1,000 and 5,000 examples, respectively. Occasional erroneous reconstruction of images indicates that KPCA requires a large amount of data to properly sample the underlying structure.

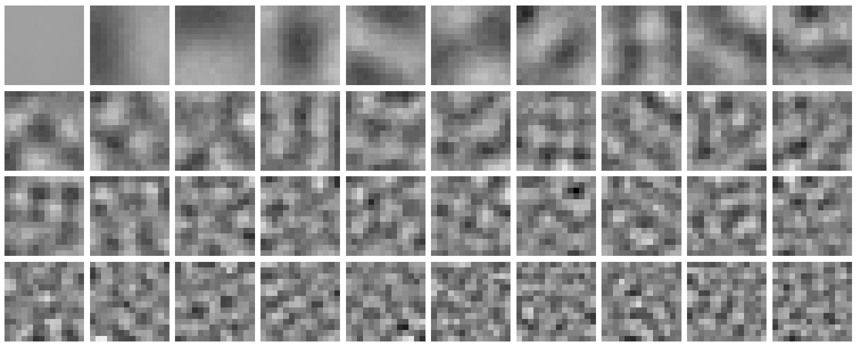

Figure 4: The first 40 kernel principal components of 40,000 (14×14)-sized patches of natural images obtained from the KHA using a Gaussian kernel with $\sigma = 40$.

a direct application of KPCA is not feasible for this large dataset. The plausibility of the obtained principal components can be demonstrated by increasing the size of the Gaussian kernel such that the distance metric of the corresponding RKHS becomes more and more similar to that of input space [2]. As can be seen in Fig. 4, the principal components approach those of linear PCA [4] as expected.

To hyperresolution for larger images, 7,000 (14×14)-sized image patches are used for training the KHA. This time, the σ parameter is set to a rather small value (0.5) to capture the nonlinear structure of the images. The low resolution input image is then divided into a set of (14×14)-sized windows each of which is reconstructed based on 300 principal components. The problem of this approach is that the resulting image as a whole shows a block structure since each window is reconstructed independent of its neighborhood (Fig. 5.f). To reduce this effect, the windows are configured to slightly overlap into their neighboring windows (Fig. 5.e). In the final reconstruction, the overlapping regions are averaged. PCA completely fails to get a hyperresolution image (Fig. 5.c). With a larger number of principal components, it reconstructs the original low resolution image, while a smaller number of components simply resulted in a smoothed image. Bilinear interpolation (Fig. 5.d) produces better results but, of course, fails to recover the complex local structure, especially in the leaves. In this respect, the reconstruction from KHA appears to be much better than the other two methods. This becomes apparent in the simple block-wise reconstruction (Fig. 5.f) where each block, although not consistent with other blocks, shows a natural representation of a patch of a natural scene.

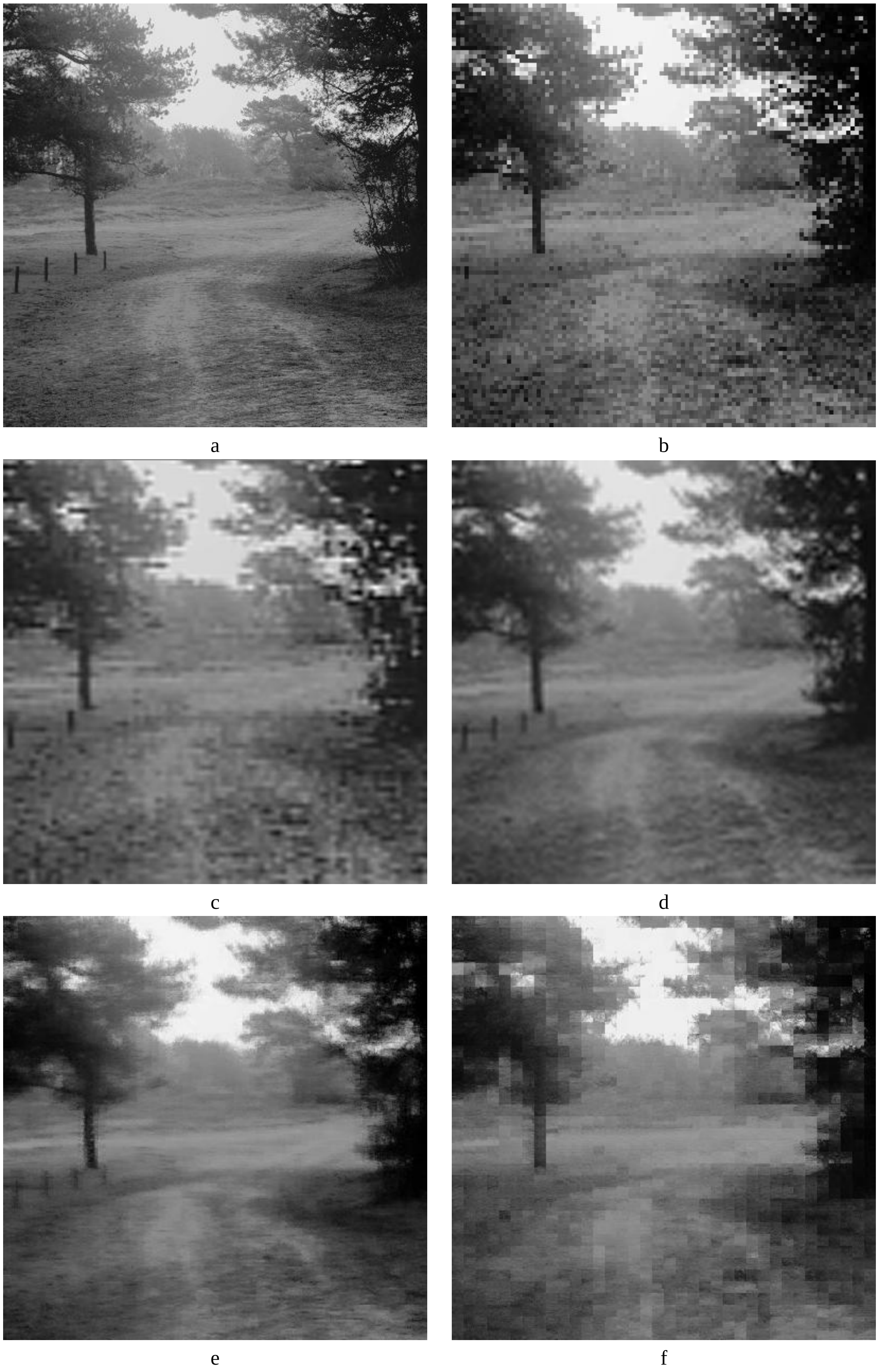

Figure 5: Example of natural image hyperresolution: a. original image of resolution 400×400 , b. low resolution image (100×100) stretched to 400×400, c. PCA reconstruction, d. bilinear interpolation, e. KPCA reconstruction with overlapping windows, and f. block-wise KPCA reconstruction.

5 Discussion

This paper formulates the KHA, a method for the efficient estimation of the principal components in an RKHS. As a kernelization of the GHA, the KHA allows for performing KPCA without storing the kernel matrix, such that large datasets of high dimensionality can be processed. This property makes the KHA particularly suitable for applications in statistical image hyperresolution. The images reconstructed by the KHA appear to be more realistic than those obtained by using linear PCA since the nonlinear principal components capture also part of the higher-order statistics of the input.

The time and memory complexity for each iteration of KHA is $\mathbf{O}(r \times l \times N)$ and $\mathbf{O}(r \times l + l \times N)$, respectively, where r, l, and N are the number of principal components to be computed, the number of examples, and the dimensionality of input space, respectively.[6] This rather high time complexity can be lowered by precomputing and storing the whole or part of the kernel matrix. When we store the entire kernel matrix, as KPCA does, the time complexity reduces to $\mathbf{O}(r \times l)$. The number of iterations for the convergence of KHA depends on the number and characteristic of data. For superresolution experiments, iteration finishes when the squared distance between two solutions from consecutive iterations is larger than a given threshold. It took around 40 and 120 iterations for face and natural image hyperresolution experiments, respectively.

Since KHA assumes a finite number of examples, it is not a true online algorithm. However, many practical problems can be processed in batch mode (i.e., all the patterns are known in advance and the number of them is finite), but, it might be still useful to have an algorithm for applications where the patterns are not known in advance and are time-varying such that KPCA cannot be applied. Some issues regarding online applications are discussed in appendix.

A Appendix: online kernel Hebbian algorithm

The proof in Section 3.2 does not draw any conclusion on the convergence properties of KHA when the number of independent samples in F is infinite. Naturally, this property depends not only on the characteristics of the underlying data but also on the RKHS concerned. A representative example of these spaces is the one induced by a Gaussian kernel where all different patterns are independent. Convergence of an algorithm in this true online problem might be of more theoretical interest, however, it is simply computationally infeasible as in this case the solutions are represented only based on an infinite number of samples. Still, much practical concern lies in applications where the sample set is not known in advance or the environment is not stationary.

This section present a modification of the batch type algorithm for this *semi-online* problem where the number of data points are finite but they are not known in advance or nonstationary. Two issues arise from this online setting: 1. To guarantee the nonorthogonality of $\mathbf{a}(0)$ and $\mathbf{q}$ (Section. 3.3.3); 2. To estimate the center of data in online;

A.1 On the nonorthogonality condition of the initial solution

It should be noted from the original formulation of GHA in RKHS (7) that when the pattern presented at time t is orthogonal to the solution $\mathbf{w}_i(t)$, the output y_i becomes zero, and accordingly $\mathbf{w}_i$ will not change. In general, according to the rule (7), $\mathbf{w}_i(t)$ cannot move into the direction orthogonal to itself. This implies that if the eigenvector $\mathbf{v}_i$ happens to be orthogonal to $\mathbf{w}_i(t)$ at t, $\mathbf{w}_i(t)$ will not be updated in the direction of $\mathbf{v}_i$. If this is true for all time t, (i.e., all patterns contained in the training set are orthogonal either to the $\mathbf{v}_i$ or to the $\mathbf{w}_i$), then clearly $\mathbf{w}_i(t)$ will not converge to $\mathbf{v}_i$. As a consequence, it is a prerequisite for the convergence that $\mathbf{w}_i(t)$ should not be orthogonal to $\mathbf{v}_i$ for all times t. Actually, this condition is equivalent to the nonorthogonality of $\mathbf{q}_i$ and $\mathbf{a}_i(t)$ in the dual space since

$$\begin{aligned} \mathbf{w}_i \cdot \mathbf{v}_i &= \mathbf{a}_i^\top \mathbf{\Phi}\mathbf{\Phi}^\top \mathbf{q}_i \\ &= \mathbf{a}_i^\top \mathbf{K}\mathbf{q}_i \\ &= (\sum_{k=1}^{l} \lambda_k \chi_k \mathbf{q}_i)^\top \mathbf{q}_i \\ &= \lambda_i \chi_i, \end{aligned}$$

where the third equality comes from (23). This is exactly why it is assumed that $\chi_i \neq 0$ during the stability analysis of (20). From this dual representation, it is evident that for the batch problem, this condition can be satisfied with

[6] $\bar{k}(\mathbf{x}_k)$ and $\bar{a}_i(t)$ in (13) for each $k, i = 1, \ldots, l$ are calculated only once at the beginning of each iteration.

probability 1 by simply initializing $\mathbf{a}_i(0)$ randomly. However, this cannot be guaranteed for online learning since the examples are not known in advance and accordingly, $\mathbf{a}_i(0)$ will in general not be contained in the span of the training sample. Furthermore we do not have any method to directly manipulate $\mathbf{w}_i(0)$ in F. Accordingly, we cannot guarantee this condition in general RKHSs.

Instead we will provide a practical way to satisfy the condition for the most commonly used three kernels (Gaussian kernels, polynomial kernels, and tangent hyperbolic kernels). If we randomly choose a vector $\mathbf{x}_i(0)$ in $\mathbb{R}^N$ and initialize $\mathbf{w}_i(0)$ with $\Phi(\mathbf{x}_i(0))$, then

$$\begin{aligned}\mathbf{w}_i(0)\cdot\mathbf{v}_i &= \Phi(\mathbf{x}_i(0))\cdot\mathbf{v}_i\\ &= \Phi(\mathbf{x}_i(0))\mathbf{\Phi}^\top\mathbf{q}_i\\ &= \sum_{k=1} q_{ik}k(\mathbf{x}_i(0),\mathbf{x}_k).\end{aligned}$$

Accordingly, in this case the orthogonality depends on the type of the kernel and is not always satisfied. However, for the Gaussian, polynomial, and tangent hyperbolic kernels, this initialization method is enough to ensure that $\mathbf{w}_i(0)\cdot\mathbf{v}_i \neq 0$ since

1. The output of an even polynomial kernel ($k(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y})^p$) is zero if and only if the inner product of two input $\mathbf{x}$ and $\mathbf{y}$ in the input space is zero. For odd polynomial kernels ($k(\mathbf{x},\mathbf{y}) = (\mathbf{x}\cdot\mathbf{y}+c)^p$), the zero output occurs only if $\mathbf{x}\cdot\mathbf{y} = -c$;

2. The output of a Gaussian kernel ($k(\mathbf{x},\mathbf{y}) = \exp(-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{y}\|^2)$) is not zero for a fixed σ and bounded $\mathbf{x}$ and $\mathbf{y}$ in the input space;

3. Similar to the case of odd polynomial kernel, the output of a tangent hyperbolic kernel ($k(\mathbf{x},\mathbf{y}) = \tanh(\mathbf{x}\cdot\mathbf{y}-b)$) is zero if and only if $\mathbf{x}\cdot\mathbf{y} = b$.

For the case of 1 and 3, random selection of $\mathbf{x}(0)$ in the input space yields $k(\mathbf{x}(0),\mathbf{x}_k) \neq 0$ in F with probability 1. Restricting the input data in a bounded domain guarantees this for the second case. Furthermore for each kernel type, if $\mathbf{x}(0)$ is chosen at random, each $k(\mathbf{x}(0),\mathbf{x}_l)$ is also random. this assures the nonorthogonality with probability 1.

Actually, for Gaussian kernels with very small σ, this initialization method does not guarantee the nonorthogonality in the real world (e.g., digital computers with limited precision). In this case, two arbitrary mapped patterns in F, which are far from each other in the input space would be regarded as orthogonal. This implies that there is a possibility that $k(\mathbf{x}_i(0),\mathbf{x}_k) = 0$, for all k (Figure 6). Furthermore, since all patterns are independent, the eigenvector generally has to be expanded in all the examples. This implies that $\mathbf{w}_i(0)$ should be nonorthogonal to all the data points in order to guarantee convergence. However, if all the patterns in input space are bounded in a ball S which is the case in many practical applications, we can still satisfy this condition by constructing $\mathbf{w}_i(0)$ as a linear combination of the mapped patterns, e,g., sampled in S at a small enough interval (Figure 7).

It should be noted that the above discussion is still valid for non-stationary environments, i.e., $\mathbf{v}_i$ is a time-varying vector. In this case, the orthogonality condition ensures not the convergence but the tracking capability of the algorithm with the additional condition that the presentation of the patterns is fast enough to keep track of the change in the environment. In this case, $\eta(t)$ should not tend to zero but to a small constant.

A.2 Kernelized update rule

The basic algorithm is the same as in (11), except that $\mathbf{w}_i(0) = \Phi(\mathbf{x}_i(0))$ with randomly chosen $\mathbf{x}_i(0)$ (i.e., $a(0)_{i0} = 1$ and $a(0)_{ij} = 0$ for $j > 0$) and $a(t)_{ij} = 0$ for all $j > t$. Then, we get the component-wise update rule:

$$a_{ij}(t+1) = \begin{cases} \eta y_i(t) & \text{if} \quad J(t) = j \\ a_{ij}(t) - \eta y_i(t)\sum_{k=1}^{i} a_{kj}(t)y_k(t) & \text{otherwise,}\end{cases}$$

where

$$y_i(t) = \sum_{k=1}^{t-1} a_i(t)\Phi(\mathbf{x}(k))^\top\Phi(\mathbf{x}(t)) = \sum_{k=1}^{t-1} a_{ik}(t)k(\mathbf{x}(k),\mathbf{x}(t)).$$

It should be noted that for online case $J(t) = t$ and accordingly, the dimensionality of solution vector $\mathbf{a}$ increases proportionally to t.

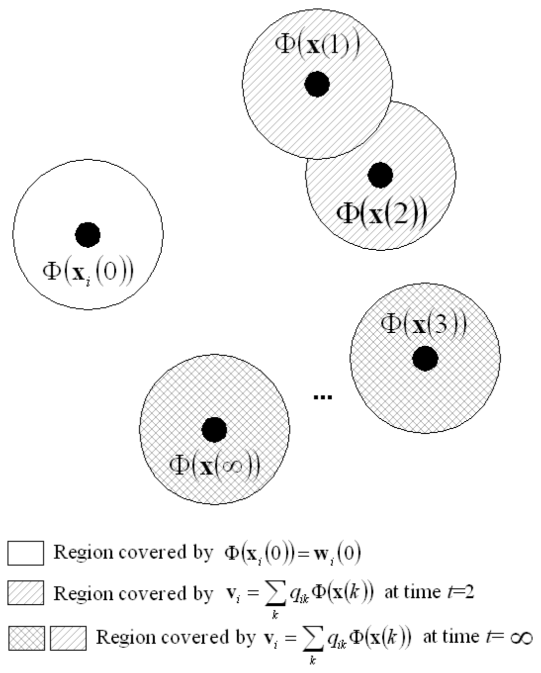
$\Phi(\mathbf{x}(1))$
$\Phi(\mathbf{x}(2))$
$\Phi(\mathbf{x}_i(0))$
$\Phi(\mathbf{x}(3))$
...
$\Phi(\mathbf{x}(\infty))$
Region covered by $\Phi(\mathbf{x}_i(0)) = \mathbf{w}_i(0)$
Region covered by $\mathbf{v}_i = \sum_k q_{ik}\Phi(\mathbf{x}(k))$ at time t=2
Region covered by $\mathbf{v}_i = \sum_k q_{ik}\Phi(\mathbf{x}(k))$ at time $t = \infty$

Figure 6: Example of non-convergence of the online algorithm for a Gaussian kernel: Small black circles represent the locations of patterns in the input space while the large circles around them show the non-orthogonal regions. Gray circles show the region covered by $\mathbf{v}_i$. The white circle is the region covered by $\mathbf{w}_i(0)$. If two regions covered by $\mathbf{w}_i(0)$ and $\mathbf{v}_i$ do not overlap each other, then they are orthogonal.

A.3 Centering data

Centering can be done by subtracting the sample mean from all patterns. Since this sample mean is not available for the online problem, we estimate at each time step the mean based only on the available data. In this case a pattern presented at time t ($\Phi(\mathbf{x}(t))$) is replaced by

$$\widetilde{\Phi}(\mathbf{x}(t)) \doteq \Phi(\mathbf{x}(t)) - \overline{\Phi}(\mathbf{x}(t)), \tag{14}$$

where $\overline{\Phi}(\mathbf{x}(t))$ is the estimated mean at time $t-1$: $\overline{\Phi}(\mathbf{x}(t)) = \frac{1}{t}\sum_{k=1}^{t-1}\Phi(\mathbf{x}(k))$. Now, $\mathbf{w}$ and y are represented based on the new centered expansions as [7]

$$\begin{aligned}\mathbf{w}_i(t) &= \sum_{k=1}^{t-1} a_{ik}(t)\widetilde{\Phi}(\mathbf{x}(k)) \\ &= \sum_{k=1}^{t-1} a_{ik}(t)\left(\Phi(\mathbf{x}(k)) - \frac{1}{t}\sum_{l=0}^{t-1}\Phi(\mathbf{x}(l))\right),\end{aligned}$$

[7] All the patterns $\Phi(\mathbf{x}(k))$ ($k < t$) in $\mathbf{w}$ and y have to be re-centered based on the new estimation of the mean.

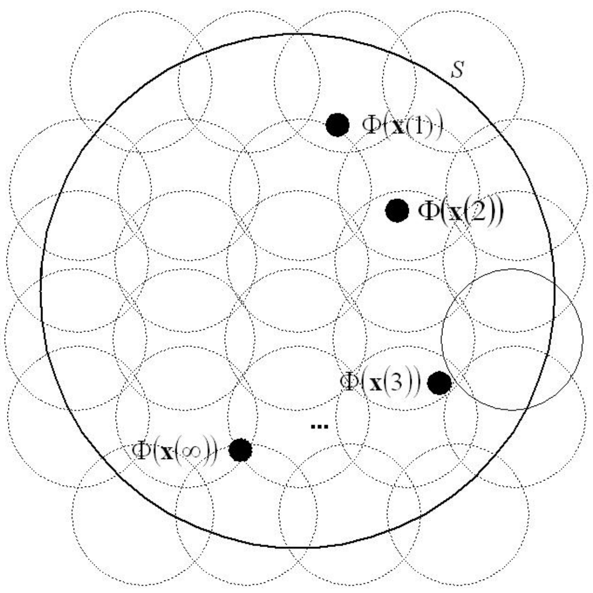

Figure 7: Example of initializing $\mathbf{w}_i(0)$ for a Gaussian kernel: Sampling points are not depicted. Instead, non-orthogonal regions are marked with dotted circles. If the sampling interval is small enough so that these regions cover S, then $\mathbf{w}_i(0)$ cannot be not orthogonal to all the data points within S.

where $\mathbf{w}_i(0) = \Phi(\mathbf{x}_i(0))$ and

$$\begin{aligned} y_i(t) &= \widetilde{\Phi}(\mathbf{x}(t)) \cdot \sum_{k=1}^{t-1} a_{ik}(t)\widetilde{\Phi}(\mathbf{x}(k)) \\ &= \sum_{k=0}^{t-1} a_{ik}(t)k(\mathbf{x}(t),\mathbf{x}(k)) - \frac{1}{t}\left[\sum_{k=0}^{t-1} a_{ik}(t)\left(\sum_{l=0}^{t-1} k(\mathbf{x}(l),\mathbf{x}(k))\right)\right] \\ &\quad -\frac{1}{t}\left(\sum_{k=0}^{t-1} k(\mathbf{x}(t),\mathbf{x}(k))\right)\left(\sum_{l=0}^{t-1} a_{il}(t)\right) + \frac{1}{t^2}\left(\sum_{k,l=0}^{t-1} k(\mathbf{x}(k),\mathbf{x}(l))\right)\left(\sum_{m=0}^{t-1} a_{im}(t)\right). \end{aligned} \tag{15}$$

Then, a new update rule (in component-wise form) based on (27)-(28) is obtained as

$$\begin{aligned} &\sum_{k=0}^{t} a_{ik}(t+1)\left(\Phi(\mathbf{x}(k)) - \frac{1}{t+1}\sum_{l=0}^{t}\Phi(\mathbf{x}(l))\right) \\ &= \sum_{k=0}^{t-1}\left[\left(a_{ik}(t) - \eta y_i(t)\sum_{l=1}^{i} a_{lk}(t)y_l(t)\right)\left(\Phi(\mathbf{x}(k)) - \frac{1}{t}\sum_{l=0}^{t-1}\Phi(\mathbf{x}(l))\right)\right] \\ &\quad +\eta y_i(t)\left(\Phi(\mathbf{x}(t)) - \frac{1}{t}\sum_{l=0}^{t-1}\Phi(\mathbf{x}(l))\right), \end{aligned} \tag{16}$$

the solution of which can be obtained from

$$a_{ik}(t+1) = \begin{cases} \eta y_i(t) + \frac{1}{t}\left[\sum_{k=0}^{t-1}\left(a_{ik}(t) - \eta y_i(t)\sum_{l=1}^{r} a_{lk}(t)y_l(t)\right) + \eta y_r(t)\right] & \text{if} \quad k = t \\ a_{ik}(t) - \eta y_i(t)\sum_{l=1}^{r} a_{lk}(t)y_l(t) & \text{otherwise.} \end{cases} \tag{17}$$

For the computation of $y(t)$, $\sum_{k,l=0}^{t-1} k(\mathbf{x}(k),\mathbf{x}(l))$ and $\sum_{k=0}^{t-1} k(\mathbf{x}(k),\mathbf{x}(i))$ are updated only for $t-1$. Accordingly the time complexity of each step is $\mathbf{O}(r \times t \times N)$.

Acknowledgments. The authors greatly profited from discussions with A. Gretton, M. Hein, and G. Bakır.

References

[1] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[2] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[3] E. Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15:267–273, 1982.

[4] T. D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward neural netowork. *Neural Networks*, 12:459–473, 1989.

[5] S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel PCA and de-noising in feature spaces. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 536–542, Cambridge, MA, 1999. MIT Press.

[6] L. Ljung. Analysis of recursive stochastic algorithms. *IEEE Trans. Automatic Control*, 22(4):551–575, 1977.

[7] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, New Jersey, 2nd edition, 1999.

[8] D. J. Field. What is the goal of sensory coding? *Neural Computation*, 6:559–601, 1994.

[9] C. J. C. Burges. Simplified support vector decision rules. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 71–77, San Mateo, CA, 1996. Morgan Kaufmann.

[10] J. T. Kwok and I. W. Tsang. Finding the pre-images in kernel principal component analysis. *6th Annual Workshop On Kernel Machines*, Whistler, Canada, 2002, poster available at http://www.cs.ust.hk/~jamesk/kernels.html.

[11] F. M. Candocia. *A unified superresolution approach for optical and synthetic aperture radar images*. PhD thesis, Univ. of Florida, Grainesville, 1998.

[12] A. S. Georghiades, P. N. Belhumeur, and D. J. Kriegman. From few to many: illumination cone models for face recognition under variable lighting and pose. *IEEE Trans. Pattern Analalysis and Machine Intelligence*, 23(6):643–660, 2001.

[13] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pages 41–48. IEEE, 1999.

[14] J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex. *Proc. R. Soc. Lond. B*, 265:359–366, 1997.