Journal of Machine Learning Research 8 (2007) 1893–1918 Submitted 11/06; Revised 4/07; Published 8/07

Fast Iterative Kernel Principal Component Analysis

Simon Günter SIMON.GUENTER@NICTA.COM.AU
Nicol N. Schraudolph NIC.SCHRAUDOLPH@NICTA.COM.AU
S.V. N. Vishwanathan SVN.VISHWANATHAN@NICTA.COM.AU

Research School of Information Sciences and Engineering
Australian National University –and–
Statistical Machine Learning Program
National ICT Australia, Locked Bag 8001
Canberra ACT 2601, Australia

Editor: Aapo Hyvarinen

Abstract

We develop gain adaptation methods that improve convergence of the kernel Hebbian algorithm (KHA) for iterative kernel PCA (Kim et al., 2005). KHA has a scalar gain parameter which is either held constant or decreased according to a predetermined annealing schedule, leading to slow convergence. We accelerate it by incorporating the reciprocal of the current estimated eigenvalues as part of a gain vector. An additional normalization term then allows us to eliminate a tuning parameter in the annealing schedule. Finally we derive and apply stochastic meta-descent (SMD) gain vector adaptation (Schraudolph, 1999, 2002) in reproducing kernel Hilbert space to further speed up convergence. Experimental results on kernel PCA and spectral clustering of USPS digits, motion capture and image denoising, and image super-resolution tasks confirm that our methods converge substantially faster than conventional KHA. To demonstrate scalability, we perform kernel PCA on the entire MNIST dataset.

Keywords: Step size adaptation, gain vector adaptation, stochastic meta-descent, kernel Hebbian algorithm, online learning.

1. Introduction

Principal components analysis (PCA) is a standard linear technique for dimensionality reduction. Given a matrix $\boldsymbol{X} \in \mathbb{R}^{n\times l}$ of l centered, n-dimensional observations, PCA performs an eigendecomposition of the covariance matrix $\boldsymbol{Q} := \boldsymbol{X}\boldsymbol{X}^\top$. The $r \times n$ matrix $\boldsymbol{W}$ whose rows are the eigenvectors of $\boldsymbol{Q}$ associated with the $r \leq n$ largest eigenvalues minimizes the least-squares reconstruction error

$$\|\boldsymbol{X} - \boldsymbol{W}^\top\boldsymbol{W}\boldsymbol{X}\|_F, \tag{1}$$

where $\|\cdot\|_F$ is the Frobenius norm.

As it takes $O(n^2 l)$ time to compute $\boldsymbol{Q}$ and $O(n^3)$ time to eigendecompose it, PCA can be prohibitively expensive for large amounts of high-dimensional data. Iterative methods exist that do not compute $\boldsymbol{Q}$ explicitly, and thereby reduce the computational cost to $O(rn)$ per iteration. They assume that each individual observation $\boldsymbol{x}$ is drawn from a statistical

©2007 Simon Günter, Nicol N. Schraudolph, and S.V. N. Vishwanathan.

distribution[1], and the aim is to maximize the variance of $\boldsymbol{y} := \boldsymbol{W}\boldsymbol{x}$, subject to some orthonormality constraints on the weight matrix $\boldsymbol{W}$. In particular, we obtain the so-called hierarchical PCA network if we assume that the i^{th} row of $\boldsymbol{W}$ must have unit norm and must be orthogonal to the j^{th} row, where $j = 1, \ldots, i-1$ (Karhunen, 1994). By using Lagrange multipliers to incorporate the constraints into the objective, we can rewrite the merit function $J(\boldsymbol{W})$ succinctly as (Karhunen and Joutsensalo, 1994):

$$J(\boldsymbol{W}) = \mathbb{E}[\boldsymbol{x}^\top \boldsymbol{W}^\top \boldsymbol{W}\boldsymbol{x}] + \tfrac{1}{2}\mathrm{tr}[\boldsymbol{\Lambda}(\boldsymbol{W}\boldsymbol{W}^\top - \boldsymbol{I})], \tag{2}$$

where the Lagrange multiplier matrix $\boldsymbol{\Lambda}$ is constrained to be lower triangular. Taking gradients with respect to $\boldsymbol{W}$ and setting to zero yields

$$\partial_{\boldsymbol{W}} J(\boldsymbol{W}) = \mathbb{E}[\boldsymbol{W}\boldsymbol{x}]\boldsymbol{x}^\top + \boldsymbol{\Lambda}\boldsymbol{W} = \boldsymbol{0}. \tag{3}$$

As a consequence of the KKT conditions (Boyd and Vandenberghe, 2004), at optimality

$$\boldsymbol{\Lambda}(\boldsymbol{W}\boldsymbol{W}^\top - \boldsymbol{I}) = \boldsymbol{0}. \tag{4}$$

Right multiplying (3) by $\boldsymbol{W}^\top$, using (4), and noting that $\boldsymbol{\Lambda}$ must be lower triangular yields

$$\boldsymbol{\Lambda} = -\mathrm{lt}(\mathbb{E}\,[\boldsymbol{W}\boldsymbol{x}]\,\boldsymbol{x}^\top \boldsymbol{W}^\top) = -\mathrm{lt}(\mathbb{E}[\boldsymbol{y}]\boldsymbol{y}^\top), \tag{5}$$

where $\mathrm{lt}(\cdot)$ makes its argument lower triangular by zeroing all elements above the diagonal. Plugging (5) into (3) and stochastically approximating the expectation $\mathbb{E}[\boldsymbol{y}]$ with its instantaneous estimate $\boldsymbol{y}_t := \boldsymbol{W}_t\boldsymbol{x}_t$, where $\boldsymbol{x}_t \in \mathbb{R}^n$ is the observation at time t, yields

$$\partial_{\boldsymbol{W}_t} J(\boldsymbol{W}) = \boldsymbol{y}_t\boldsymbol{x}_t^\top - \mathrm{lt}(\boldsymbol{y}_t\boldsymbol{y}_t^\top)\boldsymbol{W}_t. \tag{6}$$

Gradient ascent in (6) gives the *generalized Hebbian algorithm* (GHA) of Sanger (1989):

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \eta_t[\boldsymbol{y}_t\boldsymbol{x}_t^\top - \mathrm{lt}(\boldsymbol{y}_t\boldsymbol{y}_t^\top)\boldsymbol{W}_t]. \tag{7}$$

For an appropriate scalar gain, η_t, (7) will tend to converge to the principal component solution as $t \to \infty$; though its global convergence is not proven (Kim et al., 2005).

A closely related algorithm by Oja and Karhunen (1985, section 5) omits the lt operator:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t + \eta_t[\boldsymbol{y}_t\boldsymbol{x}_t^\top - \boldsymbol{y}_t\boldsymbol{y}_t^\top\boldsymbol{W}_t]. \tag{8}$$

This update is also motivated by maximizing the variance of $\boldsymbol{W}\boldsymbol{x}$ subject to orthonormality constraints on $\boldsymbol{W}$. In contrast to GHA it requires the i^{th} row of $\boldsymbol{W}$ to be orthogonal to *all* other rows of $\boldsymbol{W}$, *i.e.,* that $\boldsymbol{W}$ be orthonormal. The resulting algorithm converges to an arbitrary orthonormal basis — not necessarily the eigen-basis — for the subspace spanned by the first r eigenvectors.

One can do better than PCA in minimizing the reconstruction error (1) by allowing *nonlinear* projections of the data into r dimensions. Unfortunately such approaches often pose difficult nonlinear optimization problems. Kernel methods (Schölkopf and Smola, 2002) provide a way to incorporate non-linearity without unduly complicating the optimization

1. It is customary to assume that the distribution is centered, *i.e.,* $\mathbb{E}[\boldsymbol{x}] = \boldsymbol{0}$.

problem. Kernel PCA (Schölkopf et al., 1998) performs an eigendecomposition on the kernel expansion of the data, an $l \times l$ matrix. To reduce the attendant $O(l^2)$ space and $O(l^3)$ time complexity, Kim et al. (2005) introduced the *kernel Hebbian algorithm* (KHA) by kernelizing GHA.

Both GHA and KHA are examples of *stochastic approximation* algorithms, whose iterative updates employ individual observations in place of — but, in the limit, approximating — statistical properties of the entire data. By interleaving their updates with the passage through the data, stochastic approximation algorithms can greatly outperform conventional methods on large, redundant data sets, even though their convergence is comparatively slow.

Both GHA and KHA updates incorporate a scalar gain parameter η_t, which is either held fixed or annealed according to some predefined schedule. Robbins and Monro (1951) were first to establish conditions on the sequence of η_t that guarantee the convergence of many stochastic approximation algorithms on stationary input. A widely used annealing schedule (Darken and Moody, 1992) that obeys these conditions is

$$\eta_t = \frac{\tau}{t+\tau}\,\eta_0, \tag{9}$$

where t denotes the iteration number, and η_0, τ are positive tuning parameters. τ determines the length of an initial *search phase* with near-constant gain ($\eta_t \approx \eta_0$ for $t \ll \tau$), before the gain decays asymptotically as τ/t (for $t \gg \tau$) in the *annealing phase* (Darken and Moody, 1992). For non-stationary inputs (*e.g.*, in a online setting) Kim et al. (2005) suggest a small constant gain.

Here we propose the inclusion of a gain vector in the KHA, which provides each estimated eigenvector with its individual gain parameter. In Section 3.1 we describe our KHA/et* algorithm, which sets the gain for each eigenvector inversely proportional to its estimated eigenvalue, in addition to using (9) for annealing. Our KHA/et algorithm in Section 3.3 additionally multiplies the gain vector by the length of the vector of estimated eigenvalues; this allows us to eliminate the τ tuning parameter.

We then derive and apply the *stochastic meta-descent* (SMD) gain vector adaptation technique (Schraudolph, 1999, 2002) to KHA/et* and KHA/et to further speed up their convergence. Our resulting KHA-SMD* and KHA-SMD methods (Section 4.2) adapt gains in a reproducing kernel Hilbert space (RKHS), as pioneered in the recent *Online SVMD* algorithm (Vishwanathan et al., 2006). The application of SMD to the KHA is not trivial; a naive implementation would require $O(rl^2)$ time per update. By incrementally maintaining and updating two auxiliary matrices we reduce this cost to $O(rl)$. Our experiments in Section 5 show that the combination of preconditioning by the estimated eigenvalues and SMD can yield much faster convergence than either technique applied in isolation.

The following section summarizes the KHA, before we provide our eigenvalue-based gain modifications in Section 3. Section 4 describes SMD and its application to the KHA. We report the results of our experiments with these algorithms in Section 5, then conclude with a discussion of our findings.

2. Kernel Hebbian Algorithm

Kim et al. (2005) adapt Sanger's (1989) GHA algorithm to work with data mapped into a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ via a feature map $\Phi : \mathcal{X} \to \mathcal{H}$ (Schölkopf and

Smola, 2002). Here $\mathcal{X}$ is the input space, and $\mathcal{H}$ and Φ are implicitly defined by the kernel $k : \mathcal{X} \times \mathcal{X} \to \mathcal{H}$ with the property $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X} : \; k(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle_{\mathcal{H}}$, where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the inner product in $\mathcal{H}$. Let $\boldsymbol{\Phi}$ denote the transposed data vector in feature space:

$$\boldsymbol{\Phi} := [\Phi(\boldsymbol{x}_1), \Phi(\boldsymbol{x}_2), \; \ldots \; \Phi(\boldsymbol{x}_l)]^\top. \tag{10}$$

This assumes a fixed set of l observations whereas GHA relies on an infinite sequence of observations for convergence. Following Kim et al. (2005), we use an indexing function $p : \mathbb{N} \to \mathbb{Z}_l$ which concatenates random permutations of $\mathbb{Z}_l$ to reconcile this discrepancy. Our implementations loop through a fixed data set, permuting it anew before each pass.

PCA, GHA, and hence KHA all assume that the data is centered. Since the kernel which maps the data into feature space does not necessarily preserve such centering, we must re-center the data in feature space:

$$\boldsymbol{\Phi}' := \boldsymbol{\Phi} - \boldsymbol{M}\boldsymbol{\Phi}, \tag{11}$$

where $\boldsymbol{M}$ denotes the $l \times l$ matrix with entries all equal to $1/l$. This is achieved by replacing the *kernel matrix* $\boldsymbol{K} := \boldsymbol{\Phi}\boldsymbol{\Phi}^\top$ (that is, $[\boldsymbol{K}]_{ij} := k(\boldsymbol{x}_i, \boldsymbol{x}_j)$) by its centered version

$$\begin{aligned} \boldsymbol{K}' := \boldsymbol{\Phi}'\boldsymbol{\Phi}'^\top &= (\boldsymbol{\Phi} - \boldsymbol{M}\boldsymbol{\Phi})(\boldsymbol{\Phi} - \boldsymbol{M}\boldsymbol{\Phi})^\top \\ &= \boldsymbol{\Phi}\boldsymbol{\Phi}^\top - \boldsymbol{M}\boldsymbol{\Phi}\boldsymbol{\Phi}^\top - \boldsymbol{\Phi}\boldsymbol{\Phi}^\top\boldsymbol{M}^\top + \boldsymbol{M}\boldsymbol{\Phi}\boldsymbol{\Phi}^\top\boldsymbol{M}^\top \\ &= \boldsymbol{K} - \boldsymbol{M}\boldsymbol{K} - (\boldsymbol{M}\boldsymbol{K})^\top + \boldsymbol{M}\boldsymbol{K}\boldsymbol{M}. \end{aligned} \tag{12}$$

Since all rows of $\boldsymbol{M}\boldsymbol{K}$ are identical (as are all elements of $\boldsymbol{M}\boldsymbol{K}\boldsymbol{M}$) we can pre-calculate each row in $O(l^2)$ time and store it in $O(l)$ space to efficiently implement operations with the centered kernel. The kernel centered on the training data is also used when testing the trained system on new data.

From kernel PCA (Schölkopf et al., 1998) it is known that the principal components must lie in the span of the centered data in feature space; we can therefore express the GHA weight matrix as $\boldsymbol{W}_t = \boldsymbol{A}_t\boldsymbol{\Phi}'$, where $\boldsymbol{A}$ is an $r \times l$ matrix of expansion coefficients, and r the desired number of principal components. The GHA weight update (7) thus becomes

$$\boldsymbol{A}_{t+1}\boldsymbol{\Phi}' = \boldsymbol{A}_t\boldsymbol{\Phi}' \, + \, \eta_t[\boldsymbol{y}_t\Phi'(\boldsymbol{x}_{p(t)})^\top - \, \mathrm{lt}(\boldsymbol{y}_t\boldsymbol{y}_t^\top)\boldsymbol{A}_t\boldsymbol{\Phi}'], \tag{13}$$

where $\mathrm{lt}(\cdot)$ extracts the lower triangular part of its matrix argument (by setting all matrix elements above the diagonal to zero), and

$$\boldsymbol{y}_t := \boldsymbol{W}_t\Phi'(\boldsymbol{x}_{p(t)}) = \boldsymbol{A}_t\boldsymbol{\Phi}'\Phi'(\boldsymbol{x}_{p(t)}) = \boldsymbol{A}_t\boldsymbol{k}'_{p(t)}, \tag{14}$$

using $\boldsymbol{k}'_i$ to denote the i^{th} column of the centered kernel matrix $\boldsymbol{K}'$. Since we have $\Phi'(\boldsymbol{x}_i)^\top = \boldsymbol{e}_i^\top\boldsymbol{\Phi}'$, where $\boldsymbol{e}_i$ is the unit vector in direction i, (13) can be rewritten solely in terms of expansion coefficients as

$$\boldsymbol{A}_{t+1} = \boldsymbol{A}_t \, + \, \eta_t[\boldsymbol{y}_t\boldsymbol{e}_{p(t)}^\top - \, \mathrm{lt}(\boldsymbol{y}_t\boldsymbol{y}_t^\top)\boldsymbol{A}_t]. \tag{15}$$

Introducing the update coefficient matrix

$$\boldsymbol{\Gamma}_t := \boldsymbol{y}_t\boldsymbol{e}_{p(t)}^\top - \, \mathrm{lt}(\boldsymbol{y}_t\boldsymbol{y}_t^\top)\boldsymbol{A}_t \tag{16}$$

we obtain the compact update rule

$$\boldsymbol{A}_{t+1} = \boldsymbol{A}_t + \eta_t \boldsymbol{\Gamma}_t. \tag{17}$$

In their experiments, Kim et al. (2005) employed the KHA update (17) with a constant scalar gain $\eta_t = \eta_0$; they also proposed letting the gain decay as $\eta_t = \eta_0/t$. Our implementation (which we denote KHA/t) employs the more general (9) instead, from which an $\eta_0/(t+1)$ decay is obtained by setting $\tau = 1$, and a constant gain in the limit as $\tau \to \infty$.

3. Gain Decay with Reciprocal Eigenvalues

Consider the term $\boldsymbol{y}_t \boldsymbol{x}_t^\top = \boldsymbol{W}_t \boldsymbol{x}_t \boldsymbol{x}_t^\top$ appearing on the right-hand side of the GHA update (7). At the desired solution, the rows of $\boldsymbol{W}_t$ contain the principal components, *i.e.,* the leading eigenvectors of $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{X}^\top$. The elements of $\boldsymbol{y}_t$ thus scale with the associated eigenvalues of $\boldsymbol{Q}$. Large differences in eigenvalues can therefore lead to ill-conditioning (hence slow convergence) of the GHA; the same holds for the KHA.

We counteract this problem by furnishing the KHA with a gain vector $\boldsymbol{\eta}_t \in \mathbb{R}_+^r$ that provides each eigenvector estimate with its individual gain parameter; we will discuss how to set $\boldsymbol{\eta}_t$ below. The update rule (17) thus becomes

$$\boldsymbol{A}_{t+1} = \boldsymbol{A}_t + \operatorname{diag}(\boldsymbol{\eta}_t)\, \boldsymbol{\Gamma}_t\,, \tag{18}$$

where $\operatorname{diag}(\cdot)$ maps a vector into a diagonal matrix.

3.1 The KHA/et* Algorithm

To improve the KHA's convergence, we set $\boldsymbol{\eta}_t$ proportional to the reciprocal of the estimated eigenvalues. Let $\boldsymbol{\lambda}_t \in \mathbb{R}_+^r$ denote the vector of eigenvalues associated with the current estimate of the first r eigenvectors. Our KHA/et* algorithm sets the i^{th} component of $\boldsymbol{\eta}_t$ to

$$[\boldsymbol{\eta}_t]_i = \frac{1}{[\boldsymbol{\lambda}_t]_i}\, \frac{\tau}{t+\tau}\, \eta_0, \tag{19}$$

with η_0 and τ positive tuning parameters as in (9) before. Since we do not want the annealing phase to start before we have seen all observations at least once, we tune τ in small integer multiples of the data set size l.

KHA/et* thus conditions the KHA update by proportionately decreasing (increasing) the gain (19) for rows of $\boldsymbol{A}_t$ associated with large (small) eigenvalues. A similar approach (with a simple $1/t$ gain decay) was applied by Chen and Chang (1995) to GHA for neural network feature selection.

3.2 Calculating the Eigenvalues

The above update (19) requires the first r eigenvalues of $\boldsymbol{K}'$ — but the KHA is an algorithm for estimating these eigenvalues and their associated eigenvectors in the first place. The true eigenvalues are therefore not available at run-time. Instead we use the eigenvalues associated with the KHA's current eigenvector estimate in $\boldsymbol{A}_t$, computed as

$$[\boldsymbol{\lambda}_t]_i = \frac{\|\boldsymbol{K}'[\boldsymbol{A}_t]_{i*}^\top\|_2}{\|[\boldsymbol{A}_t]_{i*}^\top\|_2}, \tag{20}$$

where $[\boldsymbol{A}_t]_{i*}$ denotes the i^{th} row of $\boldsymbol{A}_t$, and $\|\cdot\|_2$ the 2-norm of a vector. This can be stated compactly as

$$\boldsymbol{\lambda}_t = \sqrt{\frac{\operatorname{diag}(\boldsymbol{A}_t\boldsymbol{K}'(\boldsymbol{A}_t\boldsymbol{K}')^\top)}{\operatorname{diag}(\boldsymbol{A}_t\boldsymbol{A}_t^\top)}}, \tag{21}$$

where the division and square root operation are performed element-wise, and diag$(\cdot)$ applied to a matrix extracts the vector of elements along the matrix diagonal.

The main computational effort for calculating $\boldsymbol{\lambda}_t$ lies in computing $\boldsymbol{A}_t\boldsymbol{K}'$, which — if done naively — is quite expensive: $O(rl^2)$. Fortunately it is not necessary to do this at every iteration, since the eigenvalues evolve but gradually. We empirically found it sufficient to update $\boldsymbol{\lambda}_t$ and $\boldsymbol{\eta}_t$ only once after each pass through the data, *i.e.,* every l iterations — see Figure 4. Finally, Section 4.2 below introduces incremental updates (33) and (34) that reduce the cost of calculating $\boldsymbol{A}_t\boldsymbol{K}'$ to $O(rl)$.

3.3 The KHA/et Algorithm

The τ parameter of the KHA/et* update (19) above determines at what point in the iterative kernel PCA we gradually shift from the initial search phase (with near-constant $\boldsymbol{\eta}_t$) into the asymptotic annealing phase (with $\boldsymbol{\eta}_t$ near-proportional to $1/t$). It would be advantageous if this parameter could be determined adaptively (Darken and Moody, 1992), obviating the manual tuning required in KHA/et*.

One way to achieve this is to have some measure of progress counteract the gain decay: As long as we are making rapid progress, we are in the search phase, and do not want to decrease the gains; when progress stalls it is time to start annealing them. A suitable measure of progress is $\|\boldsymbol{\lambda}_t\|$, the length of the vector of eigenvalues associated with our current estimate of the eigenvectors, as calculated via (20) above. This quantity is maximized by the true eigenvectors; in the KHA it tends to increase rapidly early on, then approach the maximum asymptotically.

Our KHA/et algorithm fixes the gain decay schedule of KHA/et* at $\tau = l$, but multiplies the gains by $\|\boldsymbol{\lambda}_t\|$:

$$[\boldsymbol{\eta}_t]_i = \frac{\|\boldsymbol{\lambda}_t\|}{[\boldsymbol{\lambda}_t]_i}\,\frac{l}{t+l}\,\eta_0. \tag{22}$$

The rapid early growth of $\|\boldsymbol{\lambda}_t\|$ thus serves to counteract the gain decay until the leading eigenspace has been identified. Asymptotically $\|\boldsymbol{\lambda}_t\|$ approaches its (constant) maximum, and so the gain decay will ultimately dominate (22). This achieves an effect comparable to an "adaptive search then converge" (ASTC) gain schedule (Darken and Moody, 1992) while eliminating the τ tuning parameter. Since (19) and (22) can both be expressed as

$$[\boldsymbol{\eta}_t]_i = \frac{\hat{\eta}_t}{[\boldsymbol{\lambda}_t]_i}, \tag{23}$$

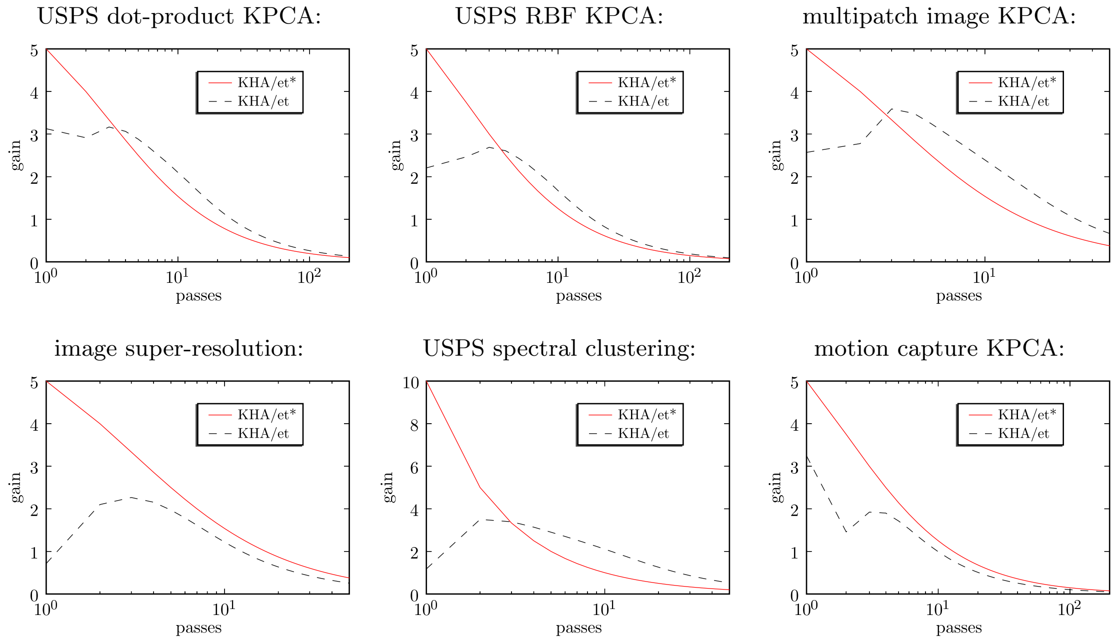


Figure 1: Comparison of gain $\hat{\eta}_t$ (23) between KHA/et* and KHA/et in all applications reported in Section 5, at individually optimal values of η_0 and (for KHA/et*) τ.

for particular choices of $\hat{\eta}_t$, we can compare the gain vectors used by KHA/et* and KHA/et by monitoring how they evolve the scalar $\hat{\eta}_t$; this is shown in Figure 1 for all experiments reported in Section 5. We see that although both algorithms ultimately anneal $\hat{\eta}_t$ in a similar fashion, their behavior early on is quite different: KHA/et keeps a lower initial gain roughly constant for a prolonged search phase, whereas KHA/et* (for the optimal choice of τ) starts decaying $\hat{\eta}_t$ far earlier, albeit from a higher starting value. In Section 5 we shall see how this affects the performance of the two algorithms.

4. KHA with Stochastic Meta-Descent

While KHA/et* and KHA/et make reasonable assumptions about how the gains of a KHA update should be scaled, further improvements are possible by adapting gains in response to the observed history of parameter updates so as to optimize convergence. We briefly review gradient-based gain adaptation methods, then derive and implement Schraudolph's (1999; 2002) *stochastic meta-descent* (SMD) algorithm for both KHA/et* and KHA/et, focusing on the scalar form of SMD that can be used in an RKHS.

4.1 Scalar Stochastic Meta-Descent

Let V be a vector space, $\boldsymbol{\theta} \in V$ a parameter vector, and $J : V \to \mathbb{R}$ the objective function which we would like to optimize. We assume that J is twice differentiable almost everywhere. Denote by $J_t : V \to \mathbb{R}$ the stochastic approximation of the objective function at time t. Our goal is to find $\boldsymbol{\theta}$ such that $\mathbb{E}_t[J_t(\boldsymbol{\theta})]$ is minimized. We adapt $\boldsymbol{\theta}$ via the stochastic gradient descent

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - e^{\rho_t}\boldsymbol{g}_t, \quad \text{where} \quad \boldsymbol{g}_t = \partial_{\boldsymbol{\theta}_t} J_t(\boldsymbol{\theta}_t), \tag{24}$$

using $\partial_{\boldsymbol{\theta}_t}$ as a shorthand for $\frac{\partial}{\partial \boldsymbol{\theta}}\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$. Stochastic gradient descent is sensitive to the value of the *log-gain* $\rho_t \in \mathbb{R}$: If it is too small, (24) will take many iterations to converge; if it is too large, (24) may diverge.

One solution is to adapt ρ_t by a simultaneous meta-level gradient descent. Thus we could seek to minimize the value of the objective at the next iteration by adjusting ρ_t in proportion to the gradient $\partial_{\rho_t} J_{t+1}(\boldsymbol{\theta}_{t+1})$. Using the chain rule and (24) we find

$$\begin{aligned} \rho_{t+1} &= \rho_t - \mu\, \partial_{\rho_t} J_{t+1}(\boldsymbol{\theta}_{t+1}) \\ &= \rho_t - \mu\, [\partial_{\boldsymbol{\theta}_{t+1}} J_{t+1}(\boldsymbol{\theta}_{t+1})]^\top \partial_{\boldsymbol{\rho}_t} \boldsymbol{\theta}_{t+1} \\ &= \rho_t + \mu\, e^{\rho_t} \boldsymbol{g}_{t+1}^\top \boldsymbol{g}_t, \end{aligned} \tag{25}$$

where the *meta-gain* $\mu \geq 0$ is a scalar tuning parameter. Intuitively, the gain adaptation (25) is driven by the angle between successive gradient measurements: If it is less than 90°, then $\boldsymbol{g}_{t+1}^\top \boldsymbol{g}_t > 0$, and ρ_t will be increased. Conversely, if the angle is more than 90° (oscillating gradient), then ρ_t will be decreased because $\boldsymbol{g}_{t+1}^\top \boldsymbol{g}_t < 0$. Thus (25) serves to *decorrelate* successive gradients, which leads to improved convergence of (24).

One shortcoming of (25) is that the decorrelation occurs only across a single time step, making the gain adaptation overly sensitive to spurious short-term correlations in the data. *Stochastic meta-descent* (SMD; Schraudolph, 1999, 2002) addresses this issue by employing an exponentially decaying trace of gradients across time:

$$\begin{aligned} \rho_{t+1} &= \rho_t - \mu \sum_{i=0}^{t} \xi^i \partial_{\rho_{t-i}} J_{t+1}(\boldsymbol{\theta}_{t+1}) \\ &= \rho_t - \mu\, [\partial_{\boldsymbol{\theta}_{t+1}} J_{t+1}(\boldsymbol{\theta}_{t+1})]^\top \sum_{i=0}^{t} \xi^i \partial_{\boldsymbol{\rho}_{t-i}} \boldsymbol{\theta}_{t+1} \\ &=: \rho_t - \mu\, \boldsymbol{g}_{t+1}^\top \boldsymbol{v}_{t+1}, \end{aligned} \tag{26}$$

where the vector $\boldsymbol{v}_{t+1} \in V$ characterizes the dependence of $\boldsymbol{\theta}_{t+1}$ on its gain history over a time scale governed by the decay factor $0 \leq \xi \leq 1$, a scalar tuning parameter.

To compute $\boldsymbol{v}_{t+1}$ efficiently, we expand $\boldsymbol{\theta}_{t+1}$ in terms of its recursive definition (24):

$$\begin{aligned} \boldsymbol{v}_{t+1} &:= \sum_{i=0}^{t} \xi^i \partial_{\rho_{t-i}} \boldsymbol{\theta}_{t+1} \\ &= \sum_{i=0}^{t} \xi^i \partial_{\rho_{t-i}} \boldsymbol{\theta}_t - \sum_{i=0}^{t} \xi^i \partial_{\rho_{t-i}} [e^{\rho_t} \boldsymbol{g}_t] \\ &\approx \xi \boldsymbol{v}_t - e^{\rho_t} (\boldsymbol{g}_t + \partial_{\boldsymbol{\theta}_t} \boldsymbol{g}_t \sum_{i=0}^{t} \xi^i \partial_{\rho_{t-i}} \boldsymbol{\theta}_t). \end{aligned} \tag{27}$$

Here we have used $\partial_{\rho_t}\boldsymbol{\theta}_t = \mathbf{0}$, and approximated

$$\sum_{i=1}^{t} \xi^i \partial_{\rho_{t-i}} \rho_t \approx 0, \tag{28}$$

which amounts to stating that the log-gain adaptation must be in equilibrium on the time scale determined by ξ. Noting that $\partial_{\boldsymbol{\theta}_t}\boldsymbol{g}_t$ is the Hessian $\boldsymbol{H}_t$ of $J_t(\boldsymbol{\theta}_t)$, we arrive at the simple iterative update

$$\boldsymbol{v}_{t+1} = \xi \boldsymbol{v}_t - e^{\rho_t}(\boldsymbol{g}_t + \xi \boldsymbol{H}_t \boldsymbol{v}_t). \tag{29}$$

Since the initial parameters $\boldsymbol{\theta}_0$ do not depend on any gains, $\boldsymbol{v}_0 = \mathbf{0}$. Note that for $\xi = 0$ (29) and (26) reduce to the single-step gain adaptation (25).

Computation of the Hessian-vector product $\boldsymbol{H}_t\boldsymbol{v}_t$ would be expensive if done naively. Fortunately, efficient methods exist to calculate this quantity directly without computing the Hessian (Pearlmutter, 1994; Griewank, 2000; Schraudolph, 2002). In essence, these methods work by propagating $\boldsymbol{v}$ as a differential (*i.e.,* directional derivative) through the gradient computation:

$$\mathrm{d}\boldsymbol{\theta}_t := \boldsymbol{v}_t \quad \Rightarrow \quad \boldsymbol{H}_t\boldsymbol{v}_t := \mathrm{d}\boldsymbol{g}_t. \tag{30}$$

In other words, if we set the differential $\mathrm{d}\boldsymbol{\theta}_t$ of the parameter vector to $\boldsymbol{v}_t$, then the resulting differential of the gradient $\boldsymbol{g}_t$ (a function of $\boldsymbol{\theta}_t$) is the Hessian-vector product $\boldsymbol{H}_t\boldsymbol{v}_t$. We will see this at work for the case of the KHA in (36) below.

4.2 SMD for KHA

The KHA update (18) can be viewed as r coupled updates in RKHS, one for each row of $\boldsymbol{A}_t$, each associated with a scalar gain. To apply SMD here we introduce an additional log-gain vector $\boldsymbol{\rho}_t \in \mathbb{R}^r$:

$$\boldsymbol{A}_{t+1} = \boldsymbol{A}_t + e^{\mathrm{diag}(\boldsymbol{\rho}_t)}\,\mathrm{diag}(\boldsymbol{\eta}_t)\,\boldsymbol{\Gamma}_t. \tag{31}$$

(The exponential of a diagonal matrix is obtained simply by exponentiating the individual diagonal entries.) We are thus applying SMD to KHA/et, *i.e.,* to a gradient descent *preconditioned* by the reciprocal estimated eigenvalues. SMD will happily work with such a preconditioner, and benefit from it.

In an RKHS, SMD adapts a *scalar* log-gain whose update is driven by the inner product between the gradient and a differential of the system parameters, all in the RKHS (Vishwanathan et al., 2006). In the case of KHA, $\boldsymbol{\Gamma}_t\boldsymbol{\Phi}'$ can be interpreted as the gradient in the RKHS of the merit function (2) maximized by KHA. Therefore SMD's adaptation of $\boldsymbol{\rho}_t$ in (31) is driven by the diagonal entries of $\langle \boldsymbol{\Gamma}_t\boldsymbol{\Phi}', \boldsymbol{B}_t\boldsymbol{\Phi}'\rangle_{\mathcal{H}}$, where $\boldsymbol{B}_t := \mathrm{d}\boldsymbol{A}_t$ denotes the $r \times l$ matrix of expansion coefficients for SMD's differential parameters, analogous to the $\boldsymbol{v}$ vector in Section 4.1:

$$\begin{aligned}
\boldsymbol{\rho}_t &= \boldsymbol{\rho}_{t-1} + \mu\,\mathrm{diag}(\langle \boldsymbol{\Gamma}_t\boldsymbol{\Phi}', \boldsymbol{B}_t\boldsymbol{\Phi}'\rangle_{\mathcal{H}}) \\
&= \boldsymbol{\rho}_{t-1} + \mu\,\mathrm{diag}(\boldsymbol{\Gamma}_t\boldsymbol{\Phi}'\boldsymbol{\Phi}'^{\top}\boldsymbol{B}_t^{\top}) \\
&= \boldsymbol{\rho}_{t-1} + \mu\,\mathrm{diag}(\boldsymbol{\Gamma}_t\boldsymbol{K}'\boldsymbol{B}_t^{\top}).
\end{aligned} \tag{32}$$

Naive computation of $\mathbf{\Gamma}_t \boldsymbol{K}'$ in (32) would cost $O(rl^2)$ time, which is prohibitively expensive for large l. We can, however, reduce this cost to $O(rl)$ by noting that (16) implies that

$$\begin{aligned}\mathbf{\Gamma}_t \boldsymbol{K}' &= \boldsymbol{y}_t \boldsymbol{e}_{p(t)}^\top \boldsymbol{K}' - \operatorname{lt}(\boldsymbol{y}_t \boldsymbol{y}_t^\top) \boldsymbol{A}_t \boldsymbol{K}' \\ &= \boldsymbol{y}_t \boldsymbol{k}'^\top_{p(t)} - \operatorname{lt}(\boldsymbol{y}_t \boldsymbol{y}_t^\top) \boldsymbol{A}_t \boldsymbol{K}',\end{aligned} \tag{33}$$

where the $r \times l$ matrix $\boldsymbol{A}_t \boldsymbol{K}'$ can be stored and updated incrementally via (31):

$$\boldsymbol{A}_{t+1} \boldsymbol{K}' = \boldsymbol{A}_t \boldsymbol{K}' + e^{\operatorname{diag}(\boldsymbol{\rho}_t)} \operatorname{diag}(\boldsymbol{\eta}_t)\, \mathbf{\Gamma}_t \boldsymbol{K}'. \tag{34}$$

The initial computation of $\boldsymbol{A}_1 \boldsymbol{K}'$ still costs $O(rl^2)$ in general but is affordable as it is performed only once. Alternatively, the time complexity of this step can easily be reduced to $O(rl)$ by making $\boldsymbol{A}_1$ suitably sparse.

Finally, we apply SMD's standard update (29) of the differential parameters:

$$\boldsymbol{B}_{t+1} = \xi \boldsymbol{B}_t + e^{\operatorname{diag}(\boldsymbol{\rho}_t)} \operatorname{diag}(\boldsymbol{\eta}_t)\, (\mathbf{\Gamma}_t + \xi \mathrm{d}\mathbf{\Gamma}_t). \tag{35}$$

The differential $\mathrm{d}\mathbf{\Gamma}_t$ of the gradient, analogous to $\mathrm{d}\boldsymbol{g}_t$ in (30), can be computed by applying the rules of calculus:

$$\begin{aligned}\mathrm{d}\mathbf{\Gamma}_t &= \mathrm{d}[\boldsymbol{y}_t \boldsymbol{e}_{p(t)}^\top - \operatorname{lt}(\boldsymbol{y}_t \boldsymbol{y}_t^\top) \boldsymbol{A}_t] \\ &= (\mathrm{d}\boldsymbol{A}_t) \boldsymbol{k}'_{p(t)} \boldsymbol{e}_{p(t)}^\top - \operatorname{lt}(\boldsymbol{y}_t \boldsymbol{y}_t^\top)(\mathrm{d}\boldsymbol{A}_t) - [\mathrm{d}\operatorname{lt}(\boldsymbol{y}_t \boldsymbol{y}_t^\top)] \boldsymbol{A}_t \\ &= \boldsymbol{B}_t \boldsymbol{k}'_{p(t)} \boldsymbol{e}_{p(t)}^\top - \operatorname{lt}(\boldsymbol{y}_t \boldsymbol{y}_t^\top) \boldsymbol{B}_t - \operatorname{lt}(\boldsymbol{B}_t \boldsymbol{k}'_{p(t)} \boldsymbol{y}_t^\top + \boldsymbol{y}_t \boldsymbol{k}'^\top_{p(t)} \boldsymbol{B}_t^\top) \boldsymbol{A}_t,\end{aligned} \tag{36}$$

using the fact that since $\boldsymbol{k}'$ and $\boldsymbol{e}$ are both independent of $\boldsymbol{A}$ we have $\mathrm{d}(\boldsymbol{k}'_{p(t)} \boldsymbol{e}_{p(t)}^\top) = 0$. Inserting (16) and (36) into (35) finally yields the update rule

$$\begin{aligned}\boldsymbol{B}_{t+1} = \xi \boldsymbol{B}_t + e^{\operatorname{diag}(\boldsymbol{\rho}_t)} \operatorname{diag}(\boldsymbol{\eta}_t)[&(\boldsymbol{A}_t + \xi \boldsymbol{B}_t)\, \boldsymbol{k}'_{p(t)} \boldsymbol{e}_{p(t)}^\top \\ &- \operatorname{lt}(\boldsymbol{y}_t \boldsymbol{y}_t^\top)(\boldsymbol{A}_t + \xi \boldsymbol{B}_t) - \xi \operatorname{lt}(\boldsymbol{B}_t \boldsymbol{k}'_{p(t)} \boldsymbol{y}_t^\top + \boldsymbol{y}_t \boldsymbol{k}'^\top_{p(t)} \boldsymbol{B}_t^\top) \boldsymbol{A}_t].\end{aligned} \tag{37}$$

In summary, our application of SMD to the KHA comprises Equations (32), (37), and (31), in that order. Our approach allows us to incorporate *a priori* knowledge about suitable gains in $\boldsymbol{\eta}_t$, which SMD will then improve upon by using empirical information gathered along the update trajectory to adaptively tune $\boldsymbol{\rho}_t$.

Algorithm 1 shows KHA-SMD, the algorithm obtained by applying SMD to KHA/et in this fashion. To obtain KHA-SMD*, the analogous algorithm applying SMD to KHA/et*, simply change step 2(b) to use (19) instead of (22). To recover KHA/et *resp.* KHA/et* from Algorithm 1, omit the steps marked with a single vertical bar. The double-barred steps do not have to be performed on every iteration; omitting them entirely, along with the single-barred steps, recovers the original KHA algorithm.

We list the worst-case time complexity of every step in terms of the number l and dimensionality n of observations, and the number r of kernel principal components to extract. For $r \ll n$ (as is typical), the most expensive step in the iteration loop will be the computation of a row of the kernel matrix in 2(c), required by all algorithms.

We initialize $\boldsymbol{\rho}_0$ to all ones, $\boldsymbol{B}_1$ to all zeroes, and $\boldsymbol{A}_1$ to an isotropic normal density with suitably small variance. The resulting time complexity of $O(rl^2)$ of step 1(c) can easily be reduced to $O(rl)$ by initializing $\boldsymbol{A}_1$ sparsely in step 1(b). This leaves the centering of the kernel in step 1(a), required by all algorithms, as the most expensive initialization step.

Algorithm 1 KHA-SMD	Eq.no.	time complexity
1. Initialize:		
(a) calculate $\boldsymbol{MK}$, $\boldsymbol{MKM}$		$O(l^2)$
(b) $\boldsymbol{A}_1 \sim N(\boldsymbol{0}, (rl)^{-1}\boldsymbol{I})$		$O(rl)$
(c) calculate $\boldsymbol{A}_1\boldsymbol{K}'$		$O(rl^2)$
(d) $\boldsymbol{\rho}_0 := [1 \ldots 1]^\top$, $\boldsymbol{B}_1 := \boldsymbol{0}$		$O(rl)$
2. **Repeat** for $t = 1, 2, \ldots$		
(a) calculate $\boldsymbol{\lambda}_t$	(20)	$O(rl)$
(b) calculate $\boldsymbol{\eta}_t$	(22)	$O(r)$
(c) calculate $\boldsymbol{k}'_{p(t)}$		$O(nl)$
(d) calculate $\boldsymbol{y}_t$	(14)	$O(rl)$
(e) calculate $\boldsymbol{\Gamma}_t$	(16)	$O(rl)$
(f) calculate $\boldsymbol{\Gamma}_t\boldsymbol{K}'$	(33)	$O(rl)$
(g) update $\boldsymbol{\rho}_{t-1} \to \boldsymbol{\rho}_t$	(32)	$O(rl)$
(h) update $\boldsymbol{B}_t \to \boldsymbol{B}_{t+1}$	(37)	$O(rl)$
(i) update $\boldsymbol{A}_t\boldsymbol{K}' \to \boldsymbol{A}_{t+1}\boldsymbol{K}'$	(34)	$O(rl)$
(j) update $\boldsymbol{A}_t \to \boldsymbol{A}_{t+1}$	(31)	$O(rl)$

5. Experiments

We present two sets of experiments. In the first, we benchmark against the KHA with a conventional gain decay schedule (9), which we denote KHA/t, in a number of different settings: Performing kernel PCA and spectral clustering on the well-known USPS dataset (LeCun et al., 1989), replicating image denoising and face image super-resolution experiments of Kim et al. (2005), and denoising human motion capture data. For Kim et al.'s (2005) experiments we also compare to their original KHA with the constant gain $\eta_t = \eta_0$ they employed. A common feature of all these datasets is that the kernel matrix can be stored in main memory, and the optimal reconstruction can thus be computed with a conventional eigensolver. In our second set of experiments we demonstrate scalability by performing kernel PCA on 60000 digits from the MNIST dataset (LeCun, 1998). Here the kernel matrix cannot be stored in main memory of a standard PC, and hence one is forced to resort to iterative methods.

5.1 Experiments on Small Datasets

In these experiments the KHA and our enhanced variants are used to find the first r eigenvectors of the centered kernel matrix $\boldsymbol{K}'$. To assess the quality of the solution, we reconstruct the kernel matrix using the eigenvectors found by the iterative algorithms, and measure the

Table 1: Parameter settings for our experiments. Footnotes indicate parameters which were individually tuned for each experiment and the given algorithm(s).

Experiment	Section	σ	τ^1	τ^2	${\eta_0}^1$	${\eta_0}^2$	${\eta_0}^3$	μ^4	μ^5	ξ
USPS (dot-prod. kernel)	5.1.1	–	$2l$	$4l$	.002	5	10^{-3}	10^{-5}	10^{-4}	0.99
USPS (RBF kernel)	5.1.1	8	l	$3l$	1	5	0.2	0.05	0.1	0.99
Lena image denoising	5.1.2	1	l	$4l$	2	5	0.1	1	2	0.99
face super-resolution	5.1.3	1	l	$4l$	0.2	5	0.02	0.2	5	0.99
USPS spectral clustering	5.1.4	8	l	l	200	10	50	20	10^3	0.99
motion capture KPCA	5.1.5	$\sqrt{1.5}$	l	$3l$	2	5	0.1	0.1	1	0.99

[1]for KHA/t [2]for KHA/et*, KHA/SMD* [3]for KHA/et, KHA/SMD [4]for KHA/SMD* [5]for KHA/SMD

reconstruction error

$$\mathcal{E}(\boldsymbol{A}) := \|\boldsymbol{K}' - (\boldsymbol{A}\boldsymbol{K}')^\top \boldsymbol{A}\boldsymbol{K}'\|_F. \tag{38}$$

Since the kernel matrix can be stored in memory, the optimal reconstruction error from r eigenvectors, $\mathcal{E}_{\min} := \min_{\boldsymbol{A}} \mathcal{E}(\boldsymbol{A})$, is computed with a conventional eigensolver. This allows us to report reconstruction errors as excess errors relative to the optimal reconstruction, *i.e.*, $\mathcal{E}(\boldsymbol{A})/\mathcal{E}_{\min} - 1$.

To compare algorithms we plot the excess reconstruction error on a logarithmic scale after each pass through the entire data set. This is a fair comparison since the overhead for KHA/et*, KHA/et, and their SMD versions is negligible compared to the time required by the KHA base algorithm: The most expensive operations — the initial centering of the kernel matrix, and the repeated calculation of a row of it — are shared by all these algorithms.

Each non-SMD algorithm had η_0 and (where applicable) τ manually tuned, by iterated hill-climbing over $\eta_0 \in \{a \cdot 10^b : a \in \{1, 2, 5\}, b \in \{-3, -2, -1, 0, 1, 2\}\}$ and $\tau \in \{l, 2l, 3l, 4l, 5l, 7l, 10l, 15l, 20l, 30l, 40l, 50l\}$, for the lowest final reconstruction error in each experiment. The SMD versions used the same values of η_0 and τ as their corresponding non-SMD variant; for them we hand-tuned μ (over the same set of values as η_0), and set $\xi = 0.99$ *a priori* throughout. Thus KHA/t and KHA/et* each had two parameters tuned specifically for them, the other algorithms one. Table 1 lists the parameter settings for each experiment, with the individually tuned parameters indicated.

5.1.1 USPS Digit KPCA

Our first benchmark is to perform iterative kernel PCA on a subset of the well-known USPS dataset (LeCun et al., 1989) — namely, the first 100 samples of each digit — with two different kernel functions: the dot-product kernel[2]

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}' \tag{39}$$

and the RBF kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(\frac{(\boldsymbol{x} - \boldsymbol{x}')^\top (\boldsymbol{x} - \boldsymbol{x}')}{2\sigma^2}\right) \tag{40}$$

2. Kernel PCA with a dot-product kernel is equivalent to ordinary PCA in the input space.

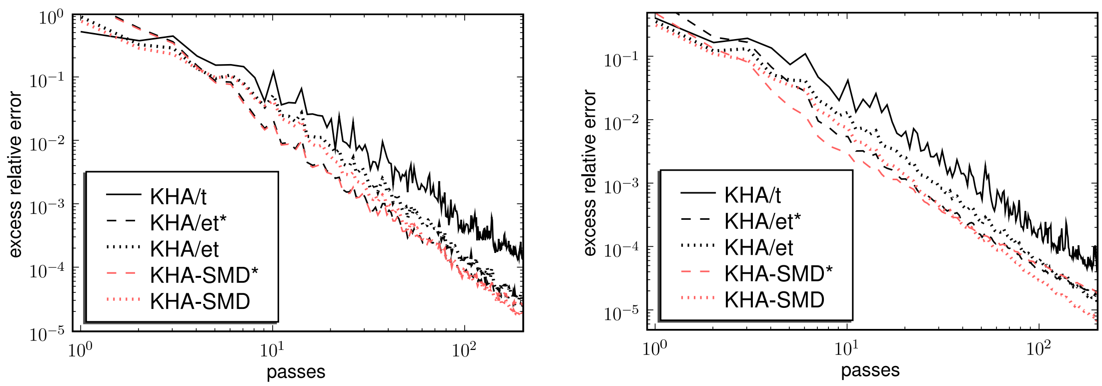


Figure 2: Excess relative reconstruction error of KHA variants for kernel PCA (16 eigenvectors) on the USPS data, using a dot-product (left) *resp.* RBF (right) kernel. (On the left, the curves for KHA/et* and KHA-SMD* virtually coincide.)

with $\sigma = 8$, the value used by Mika et al. (1999). We extract the first 16 eigenvectors of the kernel matrix and plot the excess relative error in Figure 2. Although KHA/et and KHA/et* differ in their transient behavior — the former performing better for the first 6 passes through the data, the latter thereafter — their error after 200 passes is quite similar; both clearly outperform KHA/t. SMD is able to significantly improve the performance of KHA/et but not KHA/et*, and so KHA-SMD achieves the best results on this task. These results hold for either choice of kernel. We show the first 10 eigenvectors obtained by KHA/et* for each kernel in Figure 3.

In Figure 4 we compare the performance of our algorithms, which estimate the eigenvalues and update the gains only once after every pass through the data ('p'), against variants ('i') which do this after every iteration. Tuning parameters were re-optimized for the new variants, though most optimal settings remained the same.[3] Updating the estimated eigenvalues after every iteration, though computationally expensive, is beneficial

3. The exceptions were minor: $\tau = 4$ (instead of $\tau = 3$) for KHA/et* and KHA-SMD*, $\mu = 0.1$ (instead of $\mu = 0.05$) for KHA-SMD*, and $\mu = 0.05$ (instead of $\mu = 0.1$) for KHA-SMD.

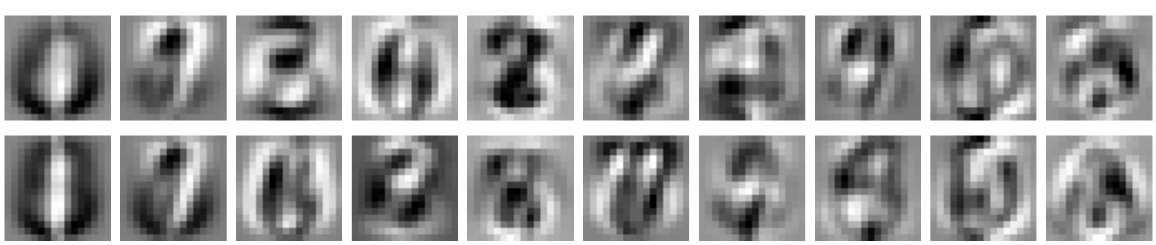


Figure 3: First ten eigenvectors (from left to right) found by KHA/et* for the dot-product (top row) *resp.* RBF kernel (bottom row).

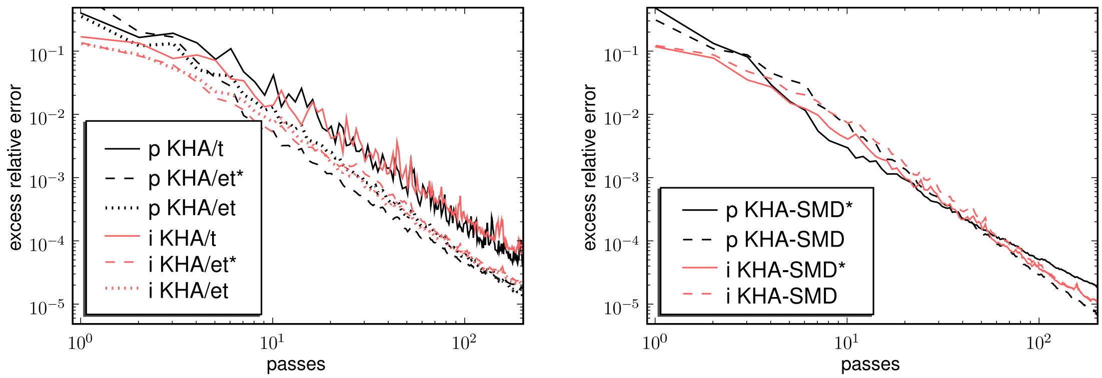


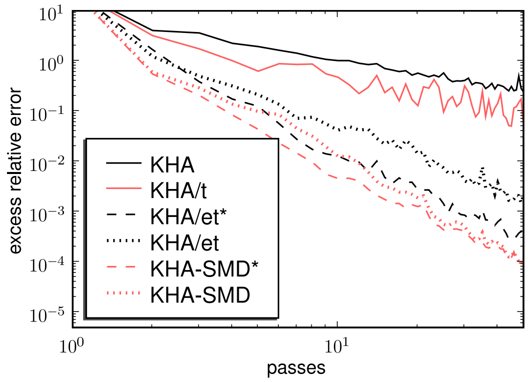
Figure 4: Comparison of excess relative reconstruction error of KHA variants estimating eigenvalues and updating gains every iteration ('i') *vs.* once every pass ('p') through the USPS data, for RBF kernel PCA extracting 16 eigenvectors.

initially but does not seem to affect the quality of the solution much in the long run; the minor differences that can be observed are attributable to differences in parameter settings.

5.1.2 Multipatch Image Denoising

For our second benchmark we replicate the image denoising problem of Kim et al. (2005), the idea being that noise can be removed from images by reconstructing image patches from their r leading eigenvectors. We divide the well-known Lena image (Munson, 1996) into four sub-images, from which 11×11 pixel windows are sampled on a grid with two-pixel spacing to produce 3844 vectors of 121 pixel intensity values each. Following Kim et al. (2005) we use an RBF kernel with $\sigma = 1$ to find the 20 best eigenvectors for each sub-image. Results averaged over the four sub-images are plotted in Figure 6 (left), including the KHA with constant gain of $\eta_t = 0.05$ employed by Kim et al. (2005) for comparison. The original, noisy, and denoised Lena images are shown in Figure 5.

Figure 5: Lena image — original (left), noisy (center), and denoised by KHA-SMD (right).



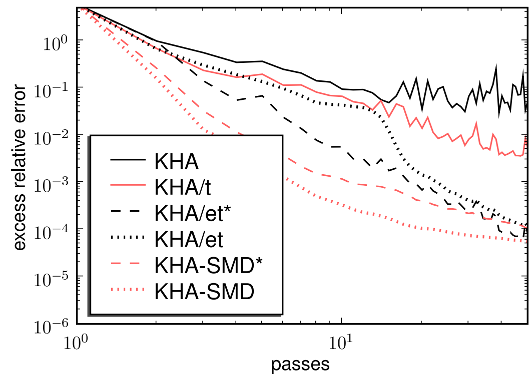


Figure 6: Excess relative reconstruction error of KHA variants in our replication of experiments due to Kim et al. (2005). Left: multipatch image kernel PCA on a noisy Lena image; Right: super-resolution of face images.

KHA/t, while better than the conventional KHA with constant gain, is clearly not as effective as our methods. Of these, KHA/et is outperformed by KHA/et* but benefits more from the addition of SMD, so that the performance of KHA-SMD is almost comparable to KHA-SMD*. KHA-SMD and KHA-SMD* achieved an excess reconstruction error that is over three orders of magnitude better than the conventional KHA after 50 passes through the data.

Replicating Kim et al.'s (2005) 800 passes through the data with the constant-gain KHA we obtain an excess relative reconstruction error of 5.64%, 500 times that of KHA-SMD after 50 passes. The signal-to-noise ratio (SNR) of the reconstruction after 800 passes with constant gain is 13.46,[4] comparable to the SNR of 13.49 achieved by KHA/et* in 50 passes.

To illustrate the large difference in early performance between conventional KHA and KHA-SMD, we show the images reconstructed from either method after 1, 2, and 3 passes through the data set in Figure 7. KHA-SMD delivers good-quality reconstructions very quickly, while those of the conventional KHA are rather blurred.

We now investigate how the different components of KHA-SMD* affect its performance. The overall gain used by KHA-SMD* comprises three factors: the scheduled gain decay over time (9), the reciprocal of the current estimated eigenvalues, and the gain adapted by SMD. Let us denote these three factors as t, e, and s, respectively, and explore which of their combinations make sense. We clearly need either t or s to give us some form of gain decay, which e does not provide. This means that in addition to the KHA/t (using only t), KHA/et* (t and e), and KHA-SMD* (t, e, and s) algorithms, there are three more feasible variants: a) s alone, b) t and s, and c) e and s.

We compare the performance of these "anonymous" variants to that of KHA/t, KHA/et*, and KHA-SMD* on the Lena image denoising problem. Parameters were tuned for each variant individually, yielding $\eta_0 = 0.5$ and $\mu = 2$ for variant s, $\eta_0 = 1$ and $\mu = 2$ for variant es, and $\tau = l$, $\eta_0 = 2$, and $\mu = 1$ for variant ts. Figure 8 (left) shows the excess

4. Kim et al. (2005) reported an SNR of 14.09; the discrepancy is due to different reconstruction methods.

Figure 7: Reconstructed Lena image after (left to right) 1, 2, and 3 passes through the data set, for KHA with constant gain $\eta_t = 0.05$ (top row) *vs.* KHA-SMD (bottom row).

relative error as a function of the number of passes through the data. On its own, SMD (s) outperforms the scheduled gain decay (t), but combining the two (ts) is better still. Introducing the reciprocal eigenvalues (e) further improves performance in every context. In short, all three factors convey a significant benefit, both individually and in combination. The "anonymous" variants represent intermediate forms between the (poorly performing) KHA/t and KHA-SMD*, which combines all three factors to attain the best results.

Next we examine the sensitivity of the KHA with SMD to the value of the meta-gain μ by increasing $\mu \in \{a \cdot 10^b : a \in \{1, 2, 5\}, b \in \{-1, 0, 1\}\}$ until the algorithm diverges. Figure 8 (right) plots the excess relative error of the s variant (SMD alone, black) and KHA-SMD* (red/gray) on the Lena image denoising problem for the last three values of μ prior to divergence. In both cases the largest non-divergent meta-gain ($\mu = 2$ for s, $\mu = 1$ for KHA-SMD*) yields the fastest convergence. The differences are comparatively small though, illustrating that SMD is not overly sensitive to the value of μ. This holds in particular for KHA-SMD*, where SMD is assisted by the other two factors, t and e.

5.1.3 Face Image Super-Resolution

We also replicate a face image super-resolution experiment of Kim et al. (2005). Here the eigenvectors learned from a training set of high-resolution images are used to predict high-resolution detail from low-resolution test images. The training set consists of 5000 face images of 10 different people from the Yale face database B (Georghiades et al., 2001),

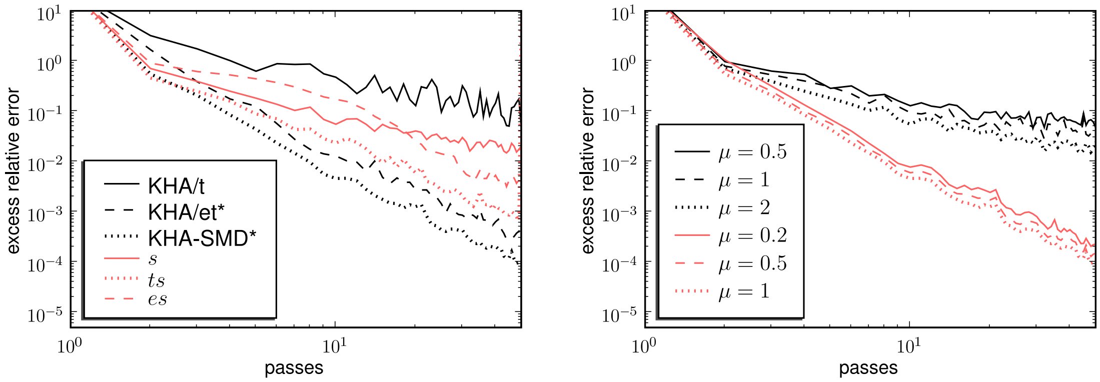


Figure 8: Excess relative reconstruction error for multipatch image PCA on a noisy Lena image. Left: comparison of original KHA variants (black) with those using other combinations (red/gray) of gain decay (t), reciprocal eigenvalues (e), and SMD (s). Right: effect of varying μ on the convergence of variant s (black) and KHA-SMD* (red/gray).

down-sampled to 60×60 pixels. Testing is done on 10 different images from the same database; the test images are first down-sampled to 20×20 pixels, then scaled back up to 60×60 by mapping each pixel to a 3×3 block of identical pixel values. These are then projected into a 16-dimensional eigenspace learned from the training set to predict the test images at the 60×60 pixel resolution.

Figure 6 (right) plots the excess relative reconstruction error of the different algorithms on this task. KHA/t again produces better results than the KHA with constant gain but is ineffective compared to our methods. KHA/et* again does better than KHA/et but benefits less from the addition of SMD making SMD-KHA once more the best-performing method. After 50 passes through the data, all our methods achieve an excess reconstruction error about three orders of magnitude better than the conventional KHA, though KHA-SMD is substantially faster than the others at reaching this level of performance. Figure 9 illustrates that the reconstructed face images after one pass through the training data generally show better high-resolution detail for KHA-SMD than for the conventional KHA with constant gain.

5.1.4 Spectral Clustering of USPS Digits

Our next experiment uses the spectral clustering algorithm of Ng et al. (2002):

1. Define the normalized transition matrix $\boldsymbol{P} := \boldsymbol{D}^{-\frac{1}{2}}\boldsymbol{K}\boldsymbol{D}^{-\frac{1}{2}}$, where $\boldsymbol{K} \in \mathbb{R}^{l\times l}$ is the kernel matrix of the data, and $\boldsymbol{D}$ is a diagonal matrix with $[\boldsymbol{D}]_{ii} = \sum_j [\boldsymbol{K}]_{ij}$.

2. Let $\boldsymbol{A} \in \mathbb{R}^{r\times l}$ be the matrix whose rows correspond to the first r eigenvectors of $\boldsymbol{P}$.

3. Normalize the columns of $\boldsymbol{A}$ to unit length, and map each input pattern to its corresponding column in $\boldsymbol{A}$.

Figure 9: Pairs of rows from top to bottom: Original face images (60×60 pixels) and subsampled images (20×20 pixels); super-resolution images produced by KHA after one *resp.* four passes through the data set; likewise for KHA-SMD.

4. Cluster the columns of $\boldsymbol{A}$ into r clusters (using, for instance, k-means clustering), and assign each pattern to the cluster its corresponding column vector belongs to.

We can obviously employ the KHA in Step 2 above. We evaluate our results in terms of the variation of information (VI) metric (Meila, 2005): For a clustering algorithm c, let $|c|$ denote the number of clusters, and $c(\cdot)$ its cluster assignment function, *i.e.*, $c(\boldsymbol{x}_i) = j$ iff c assigns pattern $\boldsymbol{x}_i$ to cluster j. Let $P_c \in \mathbb{R}^{|c|}$ denote the probability vector whose j^{th} component denotes the fraction of points assigned to cluster j, and H_c the entropy associated with P_c:

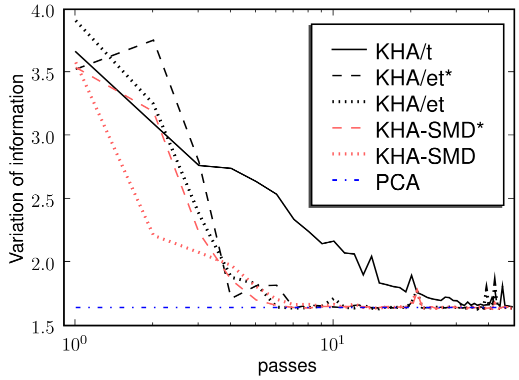
$$H_c = -\sum_{i=1}^{|c|} [P_c]_i \ln[P_c]_i. \tag{41}$$

Given two clustering algorithms c and c' we define the confusion matrix $P_c^{c'} \in \mathbb{R}^{|c|\times|c'|}$ by

$$[P_c^{c'}]_{km} = \frac{1}{l}|\{i|(c(x_i) = k) \wedge (c'(x_i) = m)\}|, \tag{42}$$

where l is the number of patterns. The mutual information $I_c^{c'}$ associated with $P_c^{c'}$ is

$$I_c^{c'} = \sum_{i=1}^{|c|}\sum_{j=1}^{|c'|} [P_c^{c'}]_{ij} \ln \frac{[P_c^{c'}]_{ij}}{[P_c]_i[P_{c'}]_j}. \tag{43}$$



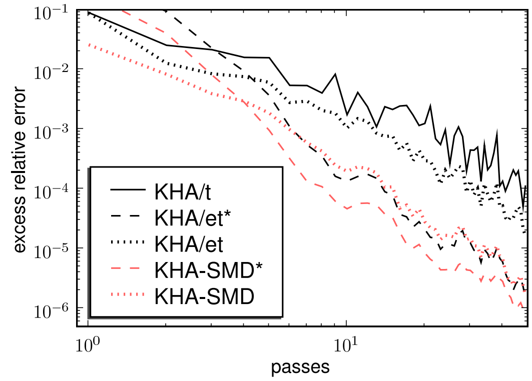


Figure 10: Quality of spectral clustering of the USPS data using an RBF kernel, as measured by variation of information (left) and excess relative reconstruction error (right). Horizontal 'PCA' line on the left marks the variation of information achieved by an exact eigensolver.

The VI metric is now defined as

$$\mathrm{VI} = H_c + H_{c'} - 2I_c^{c'}. \tag{44}$$

Our experimental task consists of applying spectral clustering to all 7291 patterns of the USPS data (LeCun et al., 1989), using 10 kernel principal components. We used a Gaussian kernel with $\sigma = 8$ and k-means with $k = 10$ (the number of labels) for clustering the columns of $\boldsymbol{A}$. The clusterings obtained by our algorithms are compared to the clustering induced by the class labels. On the USPS data, a VI of 4.54 corresponds to random grouping, while clustering in perfect accordance with the class labels would give a VI of zero.

In Figure 10 (left) we plot the VI metric as a function of the number of passes through the data. All our accelerated KHA variants converge towards an optimal clustering in less than 10 passes — in fact, after around 7 passes their results are statistically indistinguishable from that obtained by using an exact eigensolver (labeled 'PCA' in Figure 10, left). KHA/t, by contrast, needs about 30 passes through the data to reach a similar level of performance.

The excess relative reconstruction errors — for spectral clustering, of the matrix $\boldsymbol{P}$ — plotted in Figure 10 (right) confirm that our methods outperform KHA/t. They also show KHA/et* significantly outperforming KHA/et, by about an order of magnitude. Again SMD is able to substantially accelerate both KHA/et and KHA/et*. As usual the improvement is larger for the former, though in this case not by quite enough to close the performance gap to the latter.

5.1.5 Human Motion Denoising

For our next experiment we employ the KHA to denoise a human walking motion trajectory from the CMU motion capture database (http://mocap.cs.cmu.edu), converted to Cartesian coordinates via Neil Lawrence's matlab motion capture toolbox (http://www.dcs.shef.ac.uk/~neil/mocap/). The experimental setup is similar to that of Tangkuampien

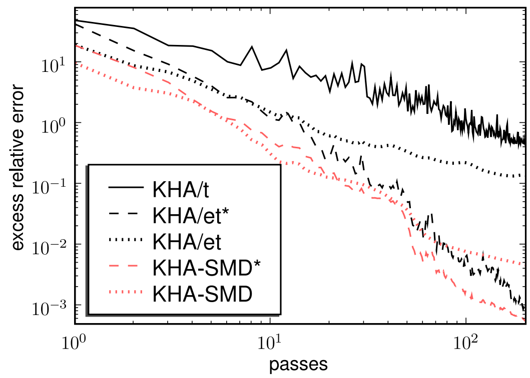


Figure 11: Excess relative reconstruction error on human motion capture data.

and Suter (2006): First zero-mean Gaussian noise is added to the frames of the original motion, then KHA using 25 principal components is used to denoise them. The noise is applied in "delta pose space," where each body part is represented by the normalized vector from its start to its end point, with a variance of 2 degrees for each of the two vector angles. The walking motion we consider has 343 frames, each represented by a 90-dimensional vector specifying the spatial orientation of 30 body parts. The motion is reconstructed in $\mathbb{R}^3$ via the KHA with an RBF kernel ($\sigma = \sqrt{1.5}$); the resulting excess relative error is shown for various KHA variants in Figure 11.

As in the previous experiment, KHA/et* clearly outperforms KHA/et which in turn is better than KHA/t. Again SMD is able to improve KHA/et to a much larger extent than KHA/et*, though not enough to surpass the latter. KHA/et* reduces the noise variance by 87.5%; it is hard to visually detect any difference between the denoised frames and the original ones — see Figure 12 for an example.

5.2 Experiments on MNIST Dataset

The MNIST dataset (LeCun, 1998) consists of 60000 handwritten digits, each 28×28 pixels in size. While kernel PCA has previously been applied to subsets of this data, to the best of our knowledge nobody has attempted it on the entire dataset — for obvious reasons: the full kernel matrix has $3.6 \cdot 10^9$ entries, requiring over 7 GB of storage in single-precision floating-point format. Storing this matrix in main memory is already a challenge, let alone computing its eigenvalues; it thus makes sense to resort to iterative schemes.

We will perform a single pass through the MNIST data, attempting to find the first 50 eigenvalues of the centered kernel matrix. Since we run through the data just once, we will update the estimated eigenvalues after each iteration rather than after every pass. Hitherto we have used the excess reconstruction error relative to the optimal reconstruction error to measure the performance of the KHA. For MNIST this is no longer possible since existing eigensolvers cannot handle such a large matrix. Instead we simply report the reconstruction error (38), which we can still compute — albeit with a rather high time complexity, as it requires calculating all entries of the kernel matrix.

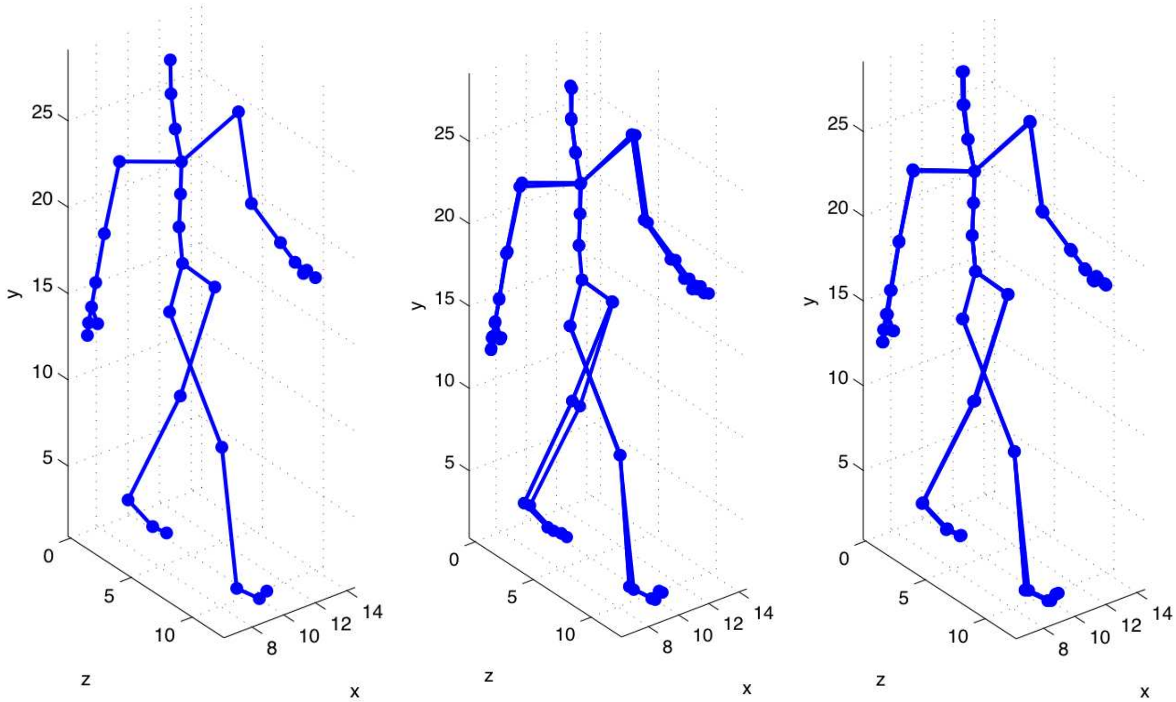


Figure 12: Reconstruction of human motion capture data: One frame of the original data (left), a superposition of this original and the noisy data (center), and a superposition of the original and reconstructed (*i.e.,* denoised) data (right).

Since our algorithms are fairly robust with respect to the value of τ, we simply set $\tau = 0.05l$ *a priori*, which corresponds to decreasing the gain by a factor of 20 during the first (and only) pass through the data. In our previous experiments we observed that the best values of η_0 and μ were usually the largest ones for which the run did not diverge. We also found that when divergence occurs, it tends to do so early and dramatically, making this event simple and inexpensive to detect. Algorithm 2 exploits this to automatically tune a gain parameter (η_0 *resp.* μ):

Algorithm 2 Auto-tune gain parameter x for KHA (any variant)

1. Compute (Algorithm 1, Step 1) and save initial KHA state;
2. $x := 500$;
3. **While** $\forall i, j : \text{is\_finite}([\boldsymbol{A}_t]_{ij})$:

   Run KHA (Algorithm 1, Step 2) for 100 iterations;
4. $x := \max_{a,b}\ a \cdot 10^b : a \in \{1, 2, 5\}, b \in \mathbb{Z}, a \cdot 10^b < x$;
5. restore initial KHA state and **Goto** Step 3.

Algorithm 2 starts with a parameter value so large (here: 500) as to surely cause divergence (Step 2). It then runs the KHA (any variant) while testing the coefficient matrix $\boldsymbol{A}_t$ every 100 iterations for signs of divergence (Step 3). If any element of $\boldsymbol{A}_t$ becomes infinite or NaN ("not a number"), the KHA has diverged; in this case the parameter value is lowered (Step 4) and the KHA restarted (Step 5). In order to make these restarts efficient, we have precomputed and saved in Step 1 the initial state of the KHA — namely a row of $\boldsymbol{MK}$, an element of $\boldsymbol{MKM}$, the initial coefficient matrix $\boldsymbol{A}_1$, and $\boldsymbol{A}_1\boldsymbol{K}'$. Once the parameter value is low enough to avoid divergence, Algorithm 2 runs the KHA to completion in Step 3.

We use Algorithm 2 to tune η_0 for KHA/et and KHA/et*, and μ for KHA-SMD and KHA-SMD*. For η_0 the SMD variants use the same value as their respective non-SMD analogues. In our experiments, divergence always occurred within the first 600 iterations (1% of the data), or not at all. It is therefore possible to tune both η_0 and μ for the SMD variants as follows: first run Algorithm 2 to tune η_0 (with $\mu = 0$) on a small fraction of the data, then run it a second time to tune μ (with the previously obtained value for η_0) on the entire dataset.

Our experiments were performed on an AMD Athlon 2.4 GHz CPU with 2 GB main memory and 512 kB cache, using a Python interface to PETSc (http://www-unix.mcs.anl.gov/petsc/petsc-as/). For a fair comparison, all our algorithms use the same initial random matrix $\boldsymbol{A}_1$, whose absolute reconstruction error is 33417. The reconstruction error after one pass through the data is shown in Table 2; it is evident that all our algorithms significantly improve upon the performance of KHA/t, with the SMD variants slightly ahead of their non-SMD analogues.

Table 2 also reports the time spent in parameter tuning, the resulting tuned parameter values, the time needed by each KHA variant for one pass through the data, and the total runtime (comprising kernel centering, parameter tuning, KHA proper, and computing the reconstruction error). Our KHA variants incur an overhead of 10–60% over the total runtime of KHA/t; the SMD variants are the more expensive. In all cases less than 5% of the total runtime was spent on parameter tuning.

6. Discussion and Conclusion

We modified the kernel Hebbian algorithm (KHA) of Kim et al. (2005) by providing a separate gain for each eigenvector estimate, and presented two methods, KHA/et* and

Table 2: Tuned parameter values (col. 2), reconstruction errors (col. 3), and runtimes for various KHA variants on the MNIST dataset. The total runtime (col. 6) is the sum of the times required to: center the kernel (11h 13'), tune the parameter (col. 4), run the KHA (col. 5), and calculate the reconstruction error (12h 16').

algorithm	parameter	rec. error	tuning	KHA time	total time
KHA/t	$\eta_0 = 5$	508.42	20'	33h 29'	57h 17'
KHA/et*	$\eta_0 = 50$	363.09	13'	41h 41'	65h 22'
KHA-SMD*	$\mu = 1$	362.44	1h 9'	53h 19'	77h 57'
KHA/et	$\eta_0 = 0.5$	415.48	47'	39h 26'	63h 42'
KHA-SMD	$\mu = 0.05$	404.06	3h 59'	64h 39'	92h 07'

KHA/et, which set those gains inversely proportional to the current estimate of the eigenvalues. KHA/et has a normalization term which allowed us to eliminate one of the free parameters of the gain decay scheme. Both methods were then enhanced by applying stochastic meta-descent (SMD) to perform gain adaptation in RKHS.

We compared our algorithms to the conventional approach of using KHA with constant gain, *resp.* with a scheduled gain decay (KHA/t), in seven different experimental settings. All our methods clearly outperformed the conventional approach in all our experiments. KHA/et* was superior to KHA/et, at the cost of having an additional free parameter τ. Its parameters, however, proved particularly easy to tune, with $\eta_0 = 5$ and $\tau = 3l$ or $4l$ optimal in all but the spectral clustering and MNIST experiments. This suggests that KHA/et* has good normalization properties and may well be preferable to KHA/et.

SMD improved the performance of both KHA/et and KHA/et*, where the improvements for the former were often larger than for the latter. This is not surprising *per se*, as it is naturally easier to improve upon a good algorithm than an excellent one. However, the fact that KHA-SMD frequently outperformed KHA-SMD* indicates that the interaction between KHA/et and SMD appears to be more effective.

Principal component analysis (PCA) is an important tool for analysis, preprocessing, and modeling of empirical data in a Euclidean space. Like other kernel methods, kernel PCA (Schölkopf et al., 1998) generalizes this to arbitrary RKHS, including those defined on structured data. Traditionally, kernel methods require computation and storage of the entire kernel matrix. As the data sets available for learning grow larger and larger, this is rapidly becoming infeasible. Recent advances eliminate this requirement by repeatedly cycling through the data set, computing kernels on demand (e.g. Platt, 1999; Joachims, 1999; Zanni et al., 2006). This is done for kernel PCA by the KHA (Kim et al., 2005), which as originally introduced suffers from slow convergence. The acceleration techniques we have introduced here rectify this situation, and hence open the way for kernel PCA to be applied to large data sets.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. A short version of this paper was presented at the 2006 NIPS conference (Schraudolph et al., 2007). National ICT Australia is funded by the Australian Government's Department of Communications, Information Technology and the Arts and the Australian Research Council through Backing Australia's Ability and the ICT Center of Excellence program. This work is supported by the IST Program of the European Community, under the Pascal Network of Excellence, IST-2002-506778.

References

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, Cambridge, England, 2004.

Liang-Hwe Chen and Shyang Chang. An adaptive learning algorithm for principal component analysis. *IEEE Transaction on Neural Networks*, 6(5):1255–1263, 1995.

Christian Darken and John E. Moody. Towards faster stochastic gradient search. In John E. Moody, Stephen J. Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 1009–1016. Morgan Kaufmann Publishers, 1992.

Athinodoros S. Georghiades, Peter N. Belhumeur, and David J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):643–660, 2001. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.927464.

Andreas Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Frontiers in Applied Mathematics. SIAM, Philadelphia, 2000.

Thorsten Joachims. Making large-scale SVM learning practical. In Bernhard Schölkopf, Chris J. C. Burges, and Alex J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

Juha Karhunen. Optimization criteria and nonlinear PCA neural networks. In *IEEE World Congress on Computational Intelligence*, volume 2, pages 1241–1246, 1994.

Juha Karhunen and Jyrki Joutsensalo. Representation and separation of signals using nonlinear PCA type learning. *Neural Networks*, 7(1):113–127, 1994.

Kwang In Kim, Matthias O. Franz, and Bernhard Schölkopf. Iterative kernel principal component analysis for image modeling. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 27(9):1351–1366, 2005.

Yann LeCun. MNIST handwritten digit database, 1998. URL `http://www.research.att.com/~yann/ocr/mnist/`.

Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

Marina Meila. Comparing clusterings: An axiomatic view. In *Proc. 22nd Intl. Conf. Machine Learning (ICML)*, pages 577–584, New York, NY, USA, 2005. ACM Press.

Sebastian Mika, Bernhard Schölkopf, Alex J. Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel PCA and de-noising in feature spaces. In Michael S. Kearns, Sara A. Solla, and David A. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11, pages 536–542. MIT Press, 1999.

David C. Munson, Jr. A note on Lena. *IEEE Trans. Image Processing*, 5(1), 1996.

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14, 2002.

Erkki Oja and Juha Karhunen. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of Mathematical Analysis and Applications*, 106(1):69–84, February 1985.

Barak A. Pearlmutter. Fast exact multiplication by the Hessian. *Neural Computation*, 6 (1):147–160, 1994.

John Platt. Fast training of support vector machines using sequential minimal optimization. In Bernhard Schölkopf, Chris J. C. Burges, and Alex J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

Terrence D. Sanger. Optimal unsupervised learning in a single-layer linear feedforward network. *Neural Networks*, 2:459–473, 1989.

Bernhard Schölkopf and Alex J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

Bernhard Schölkopf, Alex J. Smola, and Klaus-Robert Müller. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, 10:1299–1319, 1998.

Nicol N. Schraudolph. Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent. *Neural Computation*, 14(7):1723–1738, 2002.

Nicol N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *Proc. Intl. Conf. Artificial Neural Networks*, pages 569–574, Edinburgh, Scotland, 1999. IEE, London.

Nicol N. Schraudolph, Simon Günter, and S. V. N. Vishwanathan. Fast iterative kernel PCA. In Bernhard Schölkopf, John Platt, and Thomas Hofmann, editors, *Advances in Neural Information Processing Systems*, volume 19, Cambridge MA, June 2007. MIT Press.

Therdsak Tangkuampien and David Suter. Human motion de-noising via greedy kernel principal component analysis filtering. In *Proc. Intl. Conf. Pattern Recognition*, 2006.

S. V. N. Vishwanathan, Nicol N. Schraudolph, and Alex J. Smola. Step size adaptation in reproducing kernel Hilbert space. *Journal of Machine Learning Research*, 7:1107–1133, June 2006.

Luca Zanni, Thomas Serafini, and Gaetano Zanghirati. Parallel software for training large scale support vector machines on multiprocessor systems. *Journal of Machine Learning Research*, 7:1467–1492, July 2006.