January 19, 2001
RT0401
Computer Science; Mathematics 8 pages

Research Report

On-line Kernel Principal Component Analysis

Hisashi Kashima

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

On-line Kernel Principal Component Analysis

Hisashi Kashima

Tokyo Research Laboratory, IBM Research

Abstract

We introduce a new method of kernel principal component analysis that was first introduced by Shölkopf et.al.[7]. Our method extracts principal components on-line whereas that of Shölkopf et.al. is a batch algorithm. We apply kernel tricks to the on-line principal component analysis algorithms of Oja[5, 6] and Kung and Diamantaras[4] as Freund and Schapire[1] apply it to perceptron algorithm.

Keywords : Kernel Trick, Support Vector Machine, Kernel Principal Component Analysis, Perceptron

1 Introduction

Kernel trick is a method of using kernel functions to avoid explicit computation in high dimension, which was first introduced to the machine learning community by Vapnik's Support Vector Machine[8] and is applied to many learning algorithms. Shölkopf et.al.[7] applied the kernel trick to unsupervised learning and gave a nonlinear principal component analysis method using kernels. In their paper, it was shown that the kernel principal component analysis (kernel PCA) is reduced to an Eigenvalue problem like the conventional principal component analysis. While the latter uses a covariance matrix, the former uses a kernel matrix of the given data. Therefore the size of the problem depends on the number of the data. Shölkopf et.al. also demonstrated that a linear support vector machine given the features that were extracted by the kernel PCA had a good result in supervised learning.

On the other hand, Freund and Schapire[1] applied the kernel trick to on-line learning of the perceptron which is a neural network with no hidden layers. This can give an ability of nonlinear descrimination to a linear perceptron without using hidden layers. However since all learning machines using the kernel trick do not have the parameters explicitly but by a linear combination of kernels of the data processed so far, the further the learning process proceeds, the more computation the parameter updating needs.

By the way, the method of extracting principal components on-line by unsupervised learning of perceptrons was given by Oja[5]. This method can extract the first principal component by simple updating of the weights of the perceptron without solving Eigenvalue problem of a matrix. Kung and Diamantaras[4] extended Oja's method to extract pth principal component when given up to $p-1$ principal components. Oja[6] gave a method to extract P principal components simultaneously.

In this paper, by applying the kernel trick to the methods of Oja and the method of Kung and Diamantaras, we give methods of the on-line kernel PCA without solving Eigenvalue problems. Our methods are based on the on-line supervised learning of Freund and Schapire[1] since the updating rule of the on-line PCA is similar to that of the supervised perceptron. The computation times of our algorithms at each step depend on the number of the data that has already

been processed because, like Freund and Schapire's kernel perceptron, our algorithms maintain their weights as linear combinations of the kernel functions centered at the training data processed so far.
The next section reviews the conventional methods of the on-line principal component analysis by unsupervised learning of perceptrons. In section 3, we describe our method of the on-line kernel principal component analysis by applying the kernel trick. Finally we present conclusion and discussion in section 4.

2 On-line principal component analysis

2.1 Principal component analysis

Principal component analysis[3] is a method of summarizing high dimensional data into lower dimension without losing information of the original data. Let $\mathbf{x}(1), \ldots, \mathbf{x}(n)$ be the given data where each $\mathbf{x}(i)$ is a $m-$dimensional real vector $\mathbf{x}(i) = (x_1(i), \ldots, x_m(i))$. Let V be the covariance matrix of the given data. The principal component analysis is reduced to an Eigenvalue problem of V

$$\mathbf{V}\mathbf{v} = \lambda\mathbf{v}. \tag{1}$$

The Eigenvector with the largest Eigenvalue is called the first principal component, and the vector with the second largest Eigenvalue is the second principal component, and so on.

2.2 Extracting the first principal component

While the conventional PCA is reduced to an Eigenvalue problem, on-line PCA gets Eigenvectors by processing data one by one. Oja[5] showed that perceptron whose square norm of weights is normalized to be one can extract the first principal component by updating its weights to make the square of the output larger. For the tth data, the output of the normalized perceptron is

$$y(t) = \sum_i w_i(t)x_i(t) \tag{2}$$

$$\sum_i |w_i(t)|^2 = 1. \tag{3}$$

When the tth data is given, the ith element of the weight is updated by the following rule to maximize $y(t)^2$ while restricting the square norm of the weight being one.

$$w_i(t+1) = w_i(t) + \eta y(t)[x_i(t) - y(t)w_i(t)] \tag{4}$$

where $\eta > 0$ is the learning rate and the initial weight is normalized as

$$\sum_i |w_i(1)|^2 = 1. \tag{5}$$

By rewriting this rule in a vector form,

$$\mathbf{w}(t) = \mathbf{w}(t) + \eta y(t)[\mathbf{x}(t) - y(t)\mathbf{w}(t)] \tag{6}$$

We use this form in the remainder of this paper.

2.3 Extracting the pth principal component

Kung and Diamantaras[4] extended the previous algorithm to give the adaptive principal component extraction (APEX) algorithm that extracts the pth principal component when the $p-1$ largest principal components are given. We prepare one perceptron for each of the principal components.

$$y^{(p)}(t) = \sum_i w_i^{(p)}(t) x_i(t) \tag{7}$$

$$\sum_i |w_i^{(p)}(t)|^2 = 1 \tag{8}$$

Suppose that training of the perceptrons are completed up to the $p-1$th principal components and we are to extract the pth principal component. Let their outputs be $\mathbf{y}^{(p-1)}(t) = (y^{(1)}(t), \ldots, y^{(p-1)}(t))^T$ and coefficients that determine relationship between them be $\mathbf{a}^{(p-1)} = (a^{(1)}, \ldots, a^{(p-1)})^T$. The updating rule of the weights for the p th principal component is

$$\tilde{y}^{(p)}(t) = \mathbf{w}^{(p)}(t)^T \mathbf{x}(t) + \mathbf{a}^{(p-1)}(t)^T \mathbf{y}^{(p-1)}(t) \tag{9}$$

$$\mathbf{w}^{(p)}(t+1) = \mathbf{w}^{(p)}(t) + \eta y^{(p)}(t)[\mathbf{x}(t) - \tilde{y}^{(p)}(t)\mathbf{w}^{(p)}(t)] \tag{10}$$

$$\mathbf{a}^{(p)}(t+1) = \mathbf{a}^{(p)}(t) - \eta y^{(p)}(t)[\mathbf{y}^{(p-1)}(t) - y^{(p)}(t)\mathbf{a}^{(p)}(t)]. \tag{11}$$

On the other hand Oja[6] gave a method to P principal components simultaneously. The updating rule for the pth principal component is

$$\tilde{\mathbf{x}}^{(p)}(t) = \mathbf{x}(t) - \beta \sum_{j=1}^{p-1} y^{(j)}(t)\mathbf{w}^{(j)}(t) \tag{12}$$

$$\mathbf{w}^{(p)}(t+1) = \mathbf{w}^{(p)}(t) + \eta y^{(p)}(t)[\tilde{\mathbf{x}}^{(p)}(t) - y^{(p)}(t)\mathbf{w}^{(p)}(t)] \tag{13}$$

3 On-line kernel principal component analysis

3.1 kernel principal component analysis

The kernel PCA maps data in $\Re^m$ into feature space by a nonlinear map ϕ and performs conventional (i.e. linear) PCA in the feature space. Note that we suppose that inner product is defined in the feature space. The principal components in the feature space are nonlinear principal components in the original space. Let the inner product in the feature space $< \phi(\mathbf{x}(i)), \phi(\mathbf{x}(j)) >$ be defined as a kernel function $K(\mathbf{x}(i), \mathbf{x}(j)) =< \phi(\mathbf{x}(i)), \phi(\mathbf{x}(j)) >$. As examples of the kernel functions, polynimial kernels

$$K(\mathbf{x}(i), \mathbf{x}(j)) = (< \mathbf{x}(i), \mathbf{x}(j) > +1)^d, \tag{14}$$

or Gaussian kernels

$$K(\mathbf{x}(i), \mathbf{x}(j)) = \exp(-\alpha \parallel \mathbf{x}(i) - \mathbf{x}(j) \parallel^2) \tag{15}$$

are often used. The feature space of the polynomial kernels has every combination of up to d attributes in the original space and the feature space of the Gaussian kernels has infinite dimensions. The major merit of using such kernel functions is to allow computation whose time complexity depends on the dimension of the original space though it actually performs computation in a very high, sometimes infinite, dimension.
PCA in the feature space is reduced to an Eigenvalue problem of the kernel matrix

$$\mathbf{K}\mathbf{v} = \lambda \mathbf{v}. \tag{16}$$

This fact means that even when the dimension of the feature space is very high or not given explicitly, given a kernel function, we can perform PCA in the feature space. We apply this kernel trick to the on-line PCA algorithms introduced in the previous section to give nonlinear PCA algorithms not by solving Eigenvalue problems but by sequential processing of the training data.

3.2 Extracting the first principal component

The updating rule of Oja[5] is described as the following in the feature space.

$$\mathbf{w}(t) = \mathbf{w}(t) + \eta y(t)[\phi(\mathbf{x}(t)) - y(t)\mathbf{w}(t)] \tag{17}$$

Notice that $\mathbf{w}(t)$ is the weight vector in the feature space and different from the previous $\mathbf{w}(t)$ in the original space. By rewriting the weights after tth update,

$$\begin{aligned}
\mathbf{w}(t+1) &= \eta y(t)\phi(\mathbf{x}(t)) + (1 - y(t)^2)\mathbf{w}(t) && (18)\\
&= \eta y(t)\phi(\mathbf{x}(t)) + (1 - y(t)^2)\{\eta y(t-1)\phi(\mathbf{x}(t-1)) + (1 - y(t-1)^2)\mathbf{w}(t-1)\} && (19)\\
&= \eta y(t)\phi(\mathbf{x}(t)) + \eta(1 - y(t)^2)y(t-1)\phi(\mathbf{x}(t-1)) \\
&\quad +(1 - y(t)^2)(1 - y(t-1)^2)\mathbf{w}(t-1) && (20)\\
&\cdots && (21)\\
&= \eta y(t)\phi(\mathbf{x}(t)) + \eta \sum_{\tau=1}^{t-1} y(\tau)\phi(\mathbf{x}(\tau)) \prod_{\upsilon=\tau+1}^{t} (1 - y(\upsilon)^2) + \prod_{\tau=1}^{t}(1 - y(\tau)^2)\mathbf{w}(1). && (22)
\end{aligned}$$

Let us define that

$$\begin{aligned}
\psi(t,t) &= 1 && (23)\\
\psi(\tau,t) &= \prod_{\upsilon=\tau+1}^{t} (1 - y(\upsilon)^2) \text{ for } \tau < t. && (24)
\end{aligned}$$

If we know them up to a certain step t, we can recursively calculate at the next step $t+1$ as

$$\psi(\tau, t+1) = \psi(\tau,t)(1 - y(t+1)^2) \text{ for } \tau < t+1. \tag{25}$$

By using these notations, we can rewrite (22) as

$$\mathbf{w}(t+1) = \eta \sum_{\tau=1}^{t} y(\tau)\phi(\mathbf{x}(\tau))\psi(\tau,t) + \psi(0,t)\mathbf{w}(1). \tag{26}$$

Though the initial weight vector $\mathbf{w}(1)$ can be given as an arbitrary vector that satisfies $\| \mathbf{w}(1) \|_2 = 1$, there are cases where the dimension of the feature space can be very high or only kernel function is given so that the nonlinear map $\phi(\cdot)$ is not given explicitly. Therefore we firstly define any vector $\mathbf{x}(0)$ in the original space and then map it into the feature space to get $\phi(\mathbf{x}(0))$. Multiplying a constant c to $\phi(\mathbf{x}(0))$, we scale it to satisfy $\| \mathbf{w}(1) \|_2 = 1$. Note that we should not make $\mathbf{x}(0)$ be 0 in the feature space, in other words, we should take $\mathbf{x}(0)$ to satisfy $K(\mathbf{x}(0), \mathbf{x}(0)) \neq 0$. c is determined by solving the following equation.

$$\begin{aligned}
\| \mathbf{w}(1) \|_2^2 &= c^2\phi^T(\mathbf{x}(0))\phi(\mathbf{x}(0)) && (27)\\
&= c^2 K(\mathbf{x}(0), \mathbf{x}(0)) && (28)\\
&= 1. && (29)
\end{aligned}$$

Solving this gives

$$c = \sqrt{\frac{1}{K(\mathbf{x}(0), \mathbf{x}(0))}}. \tag{30}$$

Therefore the initial weight vector becomes

$$\mathbf{w}(1) = \sqrt{\frac{1}{K(\mathbf{x}(0), \mathbf{x}(0))}}\phi(\mathbf{x}(0)). \tag{31}$$

Finally the output of the perceptron at $t = 1$ is

$$\begin{aligned} y(1) &= \mathbf{w}^T(1)\phi(\mathbf{x}(1)) && (32)\\ &= \sqrt{\frac{1}{K(\mathbf{x}(0), \mathbf{x}(0))}}\phi^T(\mathbf{x}(0))\phi(\mathbf{x}(1)) && (33)\\ &= \sqrt{\frac{1}{K(\mathbf{x}(0), \mathbf{x}(0))}}K(\mathbf{x}(0), \mathbf{x}(1)) && (34)\\ \psi(1,1) &:= 1. && (35) \end{aligned}$$

For $t \geq 2$,

$$\begin{aligned} y(t) &= \mathbf{w}^T(t)\phi(\mathbf{x}(t)) && (36)\\ &= \eta \sum_{\tau=1}^{t-1} y(\tau)\psi(\tau, t-1)K(\mathbf{x}(\tau), \mathbf{x}(t)) && (37)\\ &\quad + \sqrt{\frac{1}{K(\mathbf{x}(0), \mathbf{x}(0))}}\psi(0, t-1)K(\mathbf{x}(0), \mathbf{x}(t)) && (38)\\ \psi(t,t) &:= 1 && (39)\\ \psi(\tau, t) &:= \psi(\tau, t-1)(1 - y^2(t-1)) \text{ for } \tau = 1, \ldots, t-1 && (40) \end{aligned}$$

Since the tth update needs computation time of $O(t)$, $O(T^2)$ for T updates.

3.3 Extracting the pth principal component

3.3.1 Extracting principal components on by one

By defining the perceptron for the pth principal component as

$$y^{(p)}(t) = {\mathbf{w}^{(p)}}^T(t) \cdot \phi(\mathbf{x}(t)), \tag{41}$$

the update rule of Kung and Diamantaras[4] is described as

$$\begin{aligned} \tilde{y}^{(p)}(t) &= \mathbf{w}^{(p)}(t)^T\phi(\mathbf{x}(t)) + \mathbf{a}^{(p-1)}(t)^T\mathbf{y}^{(p-1)}(t) && (42)\\ \mathbf{w}^{(p)}(t+1) &= \mathbf{w}^{(p)}(t) + \eta y^{(p)}(t)[\phi(\mathbf{x}(t)) - \tilde{y}^{(p)}(t)\mathbf{w}^{(p)}(t)] && (43)\\ \mathbf{a}^{(p)}(t+1) &= \mathbf{a}^{(p)}(t) - \eta y^{(p)}(t)[\mathbf{y}^{(p-1)}(t) - y^{(p)}(t)\mathbf{a}^{(p)}(t)]. && (44) \end{aligned}$$

Calculations of (42) and (44) can be done in $O(p)$ time. By rewriting the weight updating rule (45),

$$\begin{aligned} \mathbf{w}^{(p)}(t+1) &= \mathbf{w}^{(p)}(t) + \eta y^{(p)}(t)[\phi(\mathbf{x}(t)) - \tilde{y}^{(p)}(t)\mathbf{w}^{(p)}(t)] && (45)\\ &= \eta y^{(p)}(t)\phi(\mathbf{x}(t)) + (1 - y^{(p)}(t)\tilde{y}^{(p)}(t))\mathbf{w}^{(p)}(t) && (46) \end{aligned}$$

$$\cdots \tag{47}$$

$$= \eta y^{(p)}(t)\phi(\mathbf{x}(t)) + \eta \sum_{\tau=1}^{t-1} y^{(p)}(\tau)\phi(\mathbf{x}(\tau)) \prod_{\upsilon=\tau+1}^{t} (1 - \eta y^{(p)}(\upsilon)\tilde{y}^{(p)}(\upsilon)) + \prod_{\tau=1}^{t}(1 - \eta y^{(p)}(\tau)\tilde{y}^{(p)}(\tau))\mathbf{w}^{(p)}(1) \tag{48}$$

Here we define the initial weight vector $\mathbf{x}^{(p)}(0)$ as in the previous subsection.

$$\mathbf{w}^{(p)}(1) = \sqrt{\frac{1}{K(\mathbf{x}^{(p)}(0), \mathbf{x}^{(p)}(0))}}\phi(\mathbf{x}^{(p)}(0)) \tag{49}$$

By defining

$$\psi^{(p)}(t,t) = 1 \tag{50}$$

$$\psi^{(p)}(\tau,t) = \prod_{\upsilon=\tau+1}^{t} (1 - \eta y^{(p)}(\upsilon)\tilde{y}^{(p)}(\upsilon)) = \psi^{(p)}(\tau, t-1)(1 - \eta y^{(p)}(t)\tilde{y}^{(p)}(t)) \text{ for } \tau < t, \tag{51}$$

the weights become

$$\mathbf{w}^{(p)}(t+1) = \eta \sum_{\tau=1}^{t} y^{(p)}(\tau)\phi(\mathbf{x}(\tau))\psi^{(p)}(\tau,t) + \psi^{(p)}(0,t)\sqrt{\frac{1}{K(\mathbf{x}^{(p)}(0), \mathbf{x}^{(p)}(0))}}\phi(\mathbf{x}^{(p)}(0)). \tag{52}$$

Finally, the output of the perceptron is written as the following.

$$y^{(p)}(t) = \eta \sum_{\tau=1}^{t-1} y^{(p)}(\tau)\psi^{(p)}(\tau, t-1)K(\mathbf{x}(\tau), \mathbf{x}(t)) + \psi^{(p)}(0, t-1)\sqrt{\frac{1}{K(\mathbf{x}^{(p)}(0), \mathbf{x}^{(p)}(0))}}K(\mathbf{x}^{(p)}(0), \mathbf{x}^{(p)}(t)) \tag{53}$$

Since the tth update for the pth principal component needs computation time of $O(t+p)$, $O(T^2)$ for T updates, $O(T^2P + TP^2)$ for T updates.

3.3.2 Extracting P principal components simultaneously

By defining the perceptron for the pth principal component as

$$y^{(p)}(t', t) = \mathbf{w}^{(p)^T}(t') \cdot \phi(\mathbf{x}(t)), \tag{54}$$

the method of Oja[6] for extracting P principal components simultaneously in the feature space is described as the following.

$$\phi(\tilde{\mathbf{x}}^{(p)}(t)) = \phi(\mathbf{x}(t)) - \beta \sum_{j=1}^{p-1} y^{(j)}(t)\mathbf{w}^{(j)}(t) \tag{55}$$

$$\mathbf{w}^{(p)}(t+1) = \mathbf{w}^{(p)}(t) + \eta y^{(p)}(t,t)[\phi(\tilde{\mathbf{x}}^{(p)}(t)) - y^{(p)}(t,t)\mathbf{w}^{(p)}(t)] \tag{56}$$

The reason why to define outputs for old weights is for the following discussion.
By rewriting the weights as in the previous subsection,

$$\begin{aligned}
\mathbf{w}^{(p)}(t+1) &= \eta \sum_{\tau=1}^{t} y(\tau,\tau)\phi(\tilde{\mathbf{x}}(\tau))\psi^{(p)}(\tau,t) + \psi^{(p)}(0,t)\mathbf{w}^{(p)}(1) \qquad (57)\\
&= \eta \sum_{\tau=1}^{t} y(\tau,\tau)\phi(\mathbf{x}(\tau))\psi^{(p)}(\tau,t) + \psi^{(p)}(0,t)\mathbf{w}^{(p)}(1)\\
&\quad -\beta\eta \sum_{\tau=1}^{t} y(\tau,\tau)\psi^{(p)}(\tau,t) \sum_{j=1}^{p-1} y^{(j)}(\tau,\tau)\mathbf{w}^{(j)}(\tau) \qquad (58)
\end{aligned}$$

where

$$\begin{aligned}
\psi^{(p)}(t,t) &= 1 \qquad (59)\\
\psi^{(p)}(\tau,t) &= \prod_{\upsilon=\tau+1}^{t} (1 - y^{(p)}(\upsilon,\upsilon)^2) \text{ for } \tau < t. \qquad (60)
\end{aligned}$$

Finally, the output of the perceptron of the pth principal component is described as

$$\begin{aligned}
y^{(p)}(t,t) &= \eta \sum_{\tau=1}^{t-1} y(\tau,\tau)\psi^{(p)}(\tau,t-1)K(\mathbf{x}(\tau),\mathbf{x}(t))\\
&\quad +\psi^{(p)}(0,t-1)\sqrt{\frac{1}{K(\mathbf{x}^{(p)}(0),\mathbf{x}^{(p)}(0))}}K(\mathbf{x}^{(p)}(0),\mathbf{x}^{(p)}(t))\\
&\quad -\beta\eta \sum_{\tau=1}^{t-1} y(\tau,\tau)\psi^{(p)}(\tau,t-1) \sum_{j=1}^{p-1} y^{(j)}(\tau,\tau)y^{(j)}(\tau,t). \qquad (61)
\end{aligned}$$

The third term is because of extracting the principal components simultaneously. Thanks to this term, we have to calculate the outputs for the present input using the old weights. As a result, computation time of $O(t^2p)$ is needed for tth update of pth principal component. Finally, computation time of $O(T^3P^2)$ is needed for T updates.

4 Concluding remarks

We presented on-line principal component analysis algorithms by combining the on-line PCA algorithms firstly introduced by Oja[5] and the kernel PCA introduced by Shölkopf et.al.[7]. In fact, it is not clear whether it is advantageous or not to perform the kernel PCA in an on-line manner. All learning algorithms using the kernel trick has computation time depending on the number of the training data. In the case of the on-line kernel PCA, the computation time needed for updating the weights at a certain time depends on the number of the training data processed so far. Therefore, especially when extracting principal components simultaneously, the on-line method needs cubic order of computation time and does not seem to have an advantage over off-line methods that employ matrix calculation. One of the merit of on-line algorithm is to allow to use principal components estimated so far and to continue its training when they do not have enough accuracy. It is contrastive to the off-line cases where we need to fix the number of the training data beforehand. Furthermore, on-line learning has an ability of tracking nonstationary targets. By giving features extracted in such a manner to on-line supervised algorithms, it may be possible to construct an on-line supervised algorithm of good performance. By fixing the number of memorized data, we can perform the weight updating in a constant time.

References

[1] Freund, Y. and Schapire, R. E. Large Margin Classification Using the Perceptron Algorithm. *Machine Learning*, 1998.

[2] Haykin, S. *Neural Networks.* Prentice Hall, NJ, 1999.

[3] Jollife, I. T. *Principal Component Analysis.* Springer-Verlag, New York, 1986.

[4] Kung, S.I. and Diamantaras, K.I. A neural network learning algorithm for adaptive principal component extraction (APEX). *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2:861–864, 1990.

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

[6] Oja, E. Principal components, minor components, and linear neural networks. *Neural Networks*, 5:927–935, 1992.

[7] Shölkopf, B., Burges, C. and Smola, A. *Advances in Kernel Methods — Support Vector Learning.* MIT Press, Cambridge, MA, 1999.

[8] Vapnik, V. *Statistical Learning Theory.* Wiley, 1998.