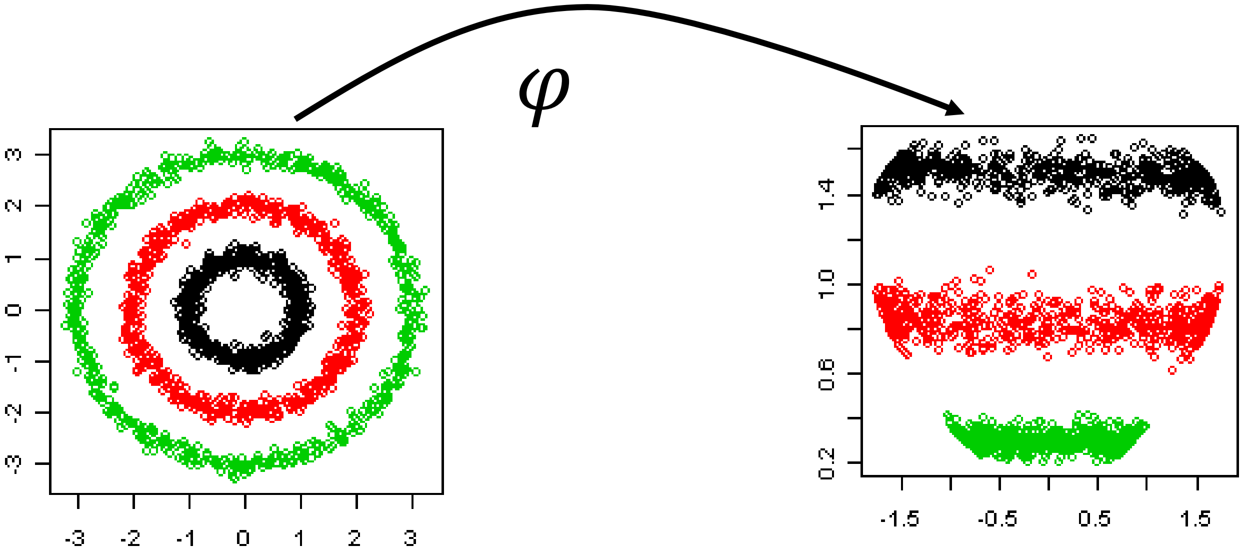
Course 495: Advanced Statistical Machine Learning/Pattern Recognition

- Goal (Lecture): To present Kernel Principal Component Analysis (KPCA) (and give a small flavour of Auto-encoders).

Materials

- Pattern Recognition & Machine Learning by C. Bishop Chapter 12
- **KPCA:** Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Müller. "Nonlinear component analysis as a kernel eigenvalue problem." Neural computation 10.5 (1998): 1299-1319.
- **Auto-Encoder:** Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. "Reducing the dimensionality of data with neural networks." *Science* 313.5786 (2006): 504-507.

Non-linear Component Analysis



$$\boldsymbol{x}_i \in R^F \longrightarrow \varphi(\boldsymbol{x}_i) \in H$$

H can be of arbitrary dimensionality

(could be even infinite)

Kernel Principal Component Analysis

- $\varphi(.)$ may not be explicitly known or is extremely expensive to compute and store.

- What is explicitly known is the dot product in H

(also known as kernel k)

$$\varphi(\boldsymbol{x}_i)^T\varphi\left(\boldsymbol{x}_j\right) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

$$(\boldsymbol{x}_i, \boldsymbol{x}_j) \in R^F \mathrm{X}\, R^F \longrightarrow k(.,.) \in R$$

- All positive (semi)-definite functions can be used as kernels

KPCA-Kernel Matrix

- Given a training population of n samples $[\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]$ we compute the training kernel matrix (also called Gram matrix).

$$\boldsymbol{K} = \left[\varphi(\boldsymbol{x}_i)^T \varphi\left(\boldsymbol{x}_j\right)\right] = \left[k\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)\right]$$

- All the computations are performed via the use of the kernel or the centralized kernel matrix.

$$\bar{\mathbf{K}} = \left(\varphi(\boldsymbol{x}_i) - \boldsymbol{m}^{\Phi}\right)^T \left(\varphi\left(\boldsymbol{x}_j\right) - \boldsymbol{m}^{\Phi}\right) \qquad \boldsymbol{m}^{\Phi} = \frac{1}{n}\sum_{i=1}^{n} \varphi(\boldsymbol{x}_i)$$

KPCA- Popular Kernels

Gaussian Radial Basis Function (RBF) kernel:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-||\boldsymbol{x}_i - \boldsymbol{x}_j||^2 / r^2}$$

Polynomial kernel: $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i{}^T \boldsymbol{x}_j + b)^n$

Hyperbolic Tangent kernel:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(\boldsymbol{x}_i{}^T \boldsymbol{x}_j + b)$$

KPCA-kernel Matrix

Input space: $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]$

Feature space: $\boldsymbol{X}^{\Phi} = [\varphi(\boldsymbol{x}_1), \dots, \varphi(\boldsymbol{x}_n)]$

Centralised:
$$\bar{\boldsymbol{X}}^{\Phi} = [\varphi(\boldsymbol{x}_1) - \boldsymbol{m}^{\Phi}, \dots, \varphi(\boldsymbol{x}_n) - \boldsymbol{m}^{\Phi}]$$
$$= \boldsymbol{X}^{\Phi}(\boldsymbol{I} - \boldsymbol{E}) = \boldsymbol{X}^{\Phi}\boldsymbol{M}, \quad \boldsymbol{E} = \frac{1}{n}\,\boldsymbol{1}\,\boldsymbol{1}^T$$

Kernel:
$$\boldsymbol{K} = [\varphi(\boldsymbol{x}_i)^T \varphi(\boldsymbol{x}_j)] = [k(\boldsymbol{x}_i, \boldsymbol{x}_j)] = {\boldsymbol{X}^{\Phi}}^T \boldsymbol{X}^{\Phi}$$

Centralised Kernel:
$$\bar{\boldsymbol{K}} = [(\varphi(\boldsymbol{x}_i) - \boldsymbol{m}^{\Phi})^T (\varphi(\boldsymbol{x}_j) - \boldsymbol{m}^{\Phi})] = (\boldsymbol{I} - \boldsymbol{E}){\boldsymbol{X}^{\Phi}}^T \boldsymbol{X}^{\Phi}(\boldsymbol{I} - \boldsymbol{E})$$
$$= (\boldsymbol{I} - \boldsymbol{E})\boldsymbol{K}(\boldsymbol{I} - \boldsymbol{E}) = \boldsymbol{K} - \boldsymbol{E}\boldsymbol{K} - \boldsymbol{K}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{K}\boldsymbol{E}$$

KPCA-Optimization problem

- KPCA cost function

$$\begin{aligned}\boldsymbol{U}^{\Phi}{}_{o} &= \arg\max_{\boldsymbol{U}^{\Phi}} \mathrm{tr}[\boldsymbol{U}^{\Phi^{T}} \boldsymbol{S}_{t}{}^{\Phi} \boldsymbol{U}^{\Phi}] \\ &= \arg\max_{\boldsymbol{U}^{\Phi}} \mathrm{tr}[\boldsymbol{U}^{\Phi^{T}} \bar{\boldsymbol{X}}^{\Phi} \bar{\boldsymbol{X}}^{\Phi^{T}} \boldsymbol{U}^{\Phi}]\end{aligned}$$

subject to $\boldsymbol{U}^{\Phi^{T}} \boldsymbol{U}^{\Phi} = \boldsymbol{I}$

- The solution is given by the d eigenvectors that correspond to the d largest eigenvalues

$$\boldsymbol{S}_{t}{}^{\Phi} \boldsymbol{U}^{\Phi}{}_{o} = \boldsymbol{U}^{\Phi}{}_{o} \boldsymbol{\Lambda}$$

KPCA- Computing Principal Components

- Do you see any problem with that?

- How can we compute the eigenvectors of $\boldsymbol{S_t}^{\Phi}$?

  We do not even know φ!!!!

- Remember our Lemma that links the eigenvectors and eigenvalues of matrices of the form $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$

$$\bar{\boldsymbol{K}} = \bar{\boldsymbol{X}}^{\Phi^T}\bar{\boldsymbol{X}}^{\Phi} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T \quad \text{then } \boldsymbol{U}^{\Phi}{}_{o} = \bar{\boldsymbol{X}}^{\Phi}\boldsymbol{V}\boldsymbol{\Lambda}^{-\frac{1}{2}}$$

- All computations are performed via the use of $\bar{\boldsymbol{K}}$ (so-called kernel trick)

KPCA- Computing Principal Components

- Still $\boldsymbol{U}^{\Phi}{}_{o} = \bar{\boldsymbol{X}}^{\Phi}\boldsymbol{V}\boldsymbol{\Lambda}^{-\frac{1}{2}}$ cannot be analytically computed.

- But we do not want to compute $\boldsymbol{U}^{\Phi}{}_{o}$.

- What we want is to compute latent features.

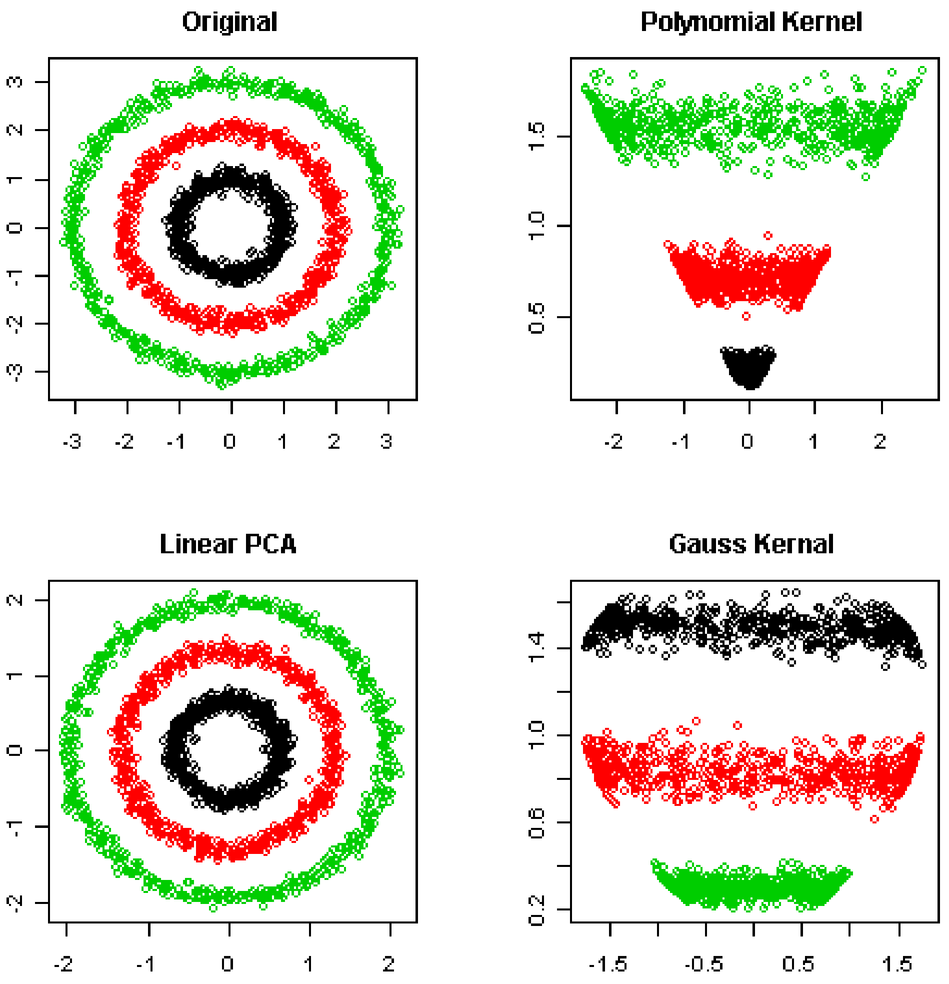
- That is, given a test sample $\boldsymbol{x}_t$ we want to compute $\boldsymbol{y} = \boldsymbol{U}^{\Phi}{}_{o}{}^{T}\varphi(\boldsymbol{x}_t)$ (this can be performed via the kernel trick)

KPCA- Extracting Latent Features

$$
\begin{aligned}
\boldsymbol{y} &= \boldsymbol{U}^{\Phi}{}_{o}{}^{T}(\varphi(\boldsymbol{x_t}) - \boldsymbol{m}^{\Phi}) \\
&= \boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{V}^{T}\bar{\boldsymbol{X}}^{\Phi^{T}}(\varphi(\boldsymbol{x_t}) - \boldsymbol{m}^{\Phi}) \\
&= \boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{V}^{T}(\boldsymbol{I} - \boldsymbol{E})\boldsymbol{X}^{\Phi^{T}}\left(\varphi(\boldsymbol{x_t}) - \frac{1}{n}\boldsymbol{X}^{\Phi}\boldsymbol{1}\right) \\
&= \boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{V}^{T}(\boldsymbol{I} - \boldsymbol{E})(\boldsymbol{X}^{\Phi^{T}}\varphi(\boldsymbol{x_t}) - \frac{1}{n}\boldsymbol{X}^{\Phi^{T}}\boldsymbol{X}^{\Phi}\boldsymbol{1}) \\
&= \boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{V}^{T}(\boldsymbol{I} - \boldsymbol{E})(g(\boldsymbol{x_t}) - \frac{1}{n}\boldsymbol{K}\boldsymbol{1})
\end{aligned}
$$

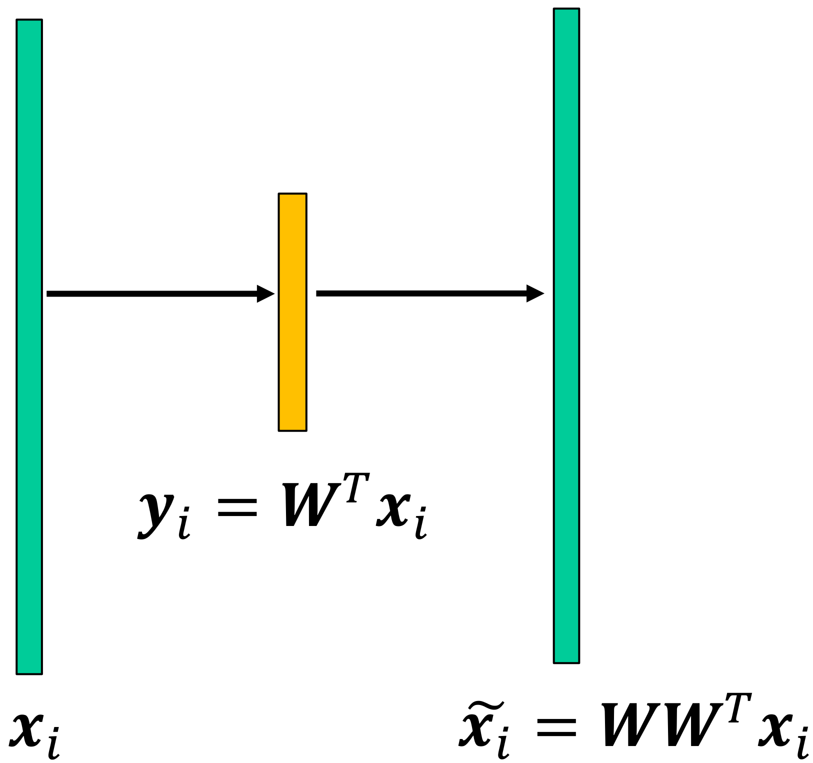
$$
g(\boldsymbol{x_t}) = \boldsymbol{X}^{\Phi^{T}}\varphi(\boldsymbol{x_t}) = \begin{bmatrix} \varphi(\boldsymbol{x}_1)^T\varphi(\boldsymbol{x}_t) \\ \dots \\ \varphi(\boldsymbol{x}_n)^T\varphi(\boldsymbol{x}_t) \end{bmatrix} = \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_t) \\ \dots \\ k(\boldsymbol{x}_n, \boldsymbol{x}_t) \end{bmatrix}
$$

KPCA- Example

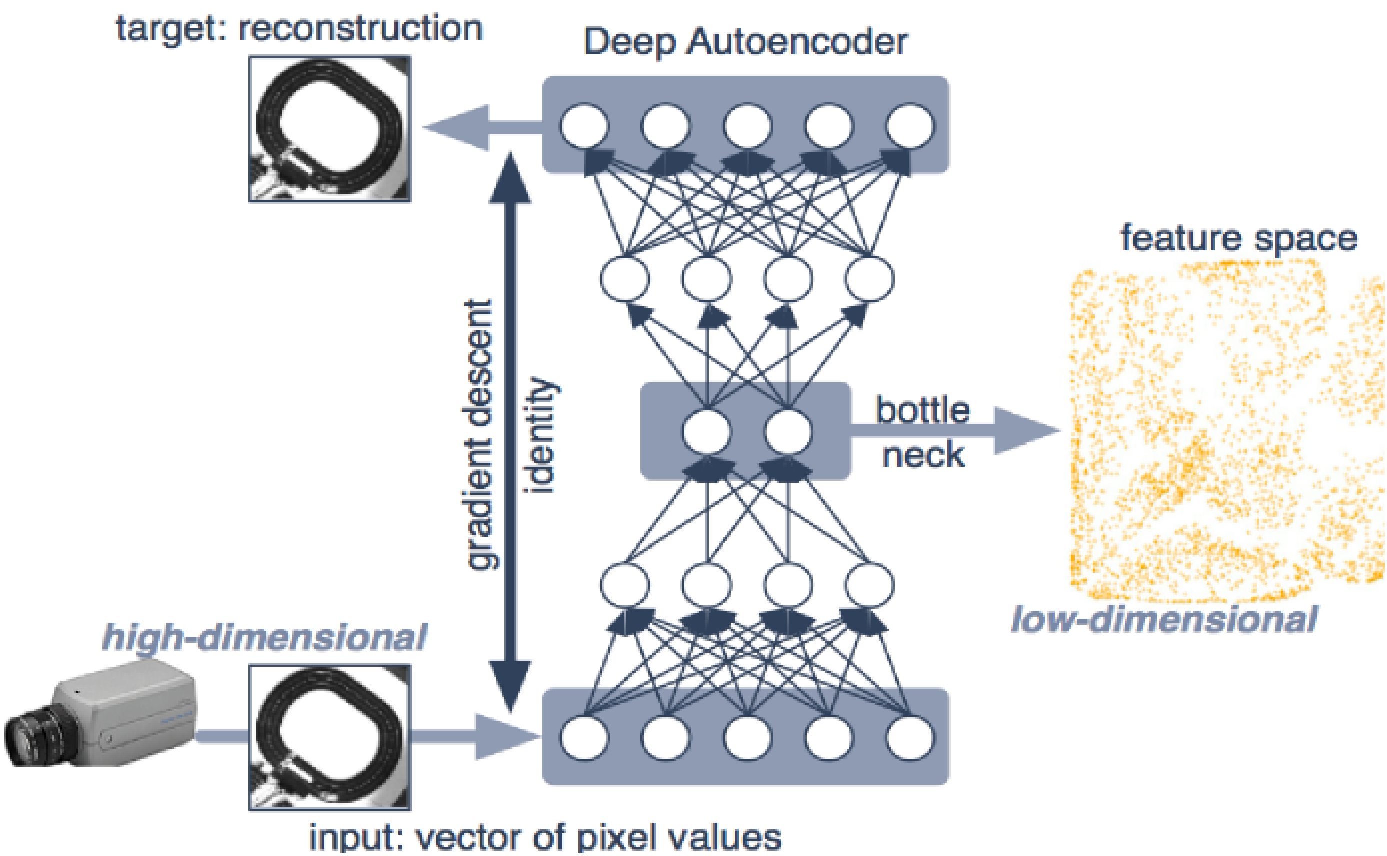


Latent Feature Extraction with Neural Networks

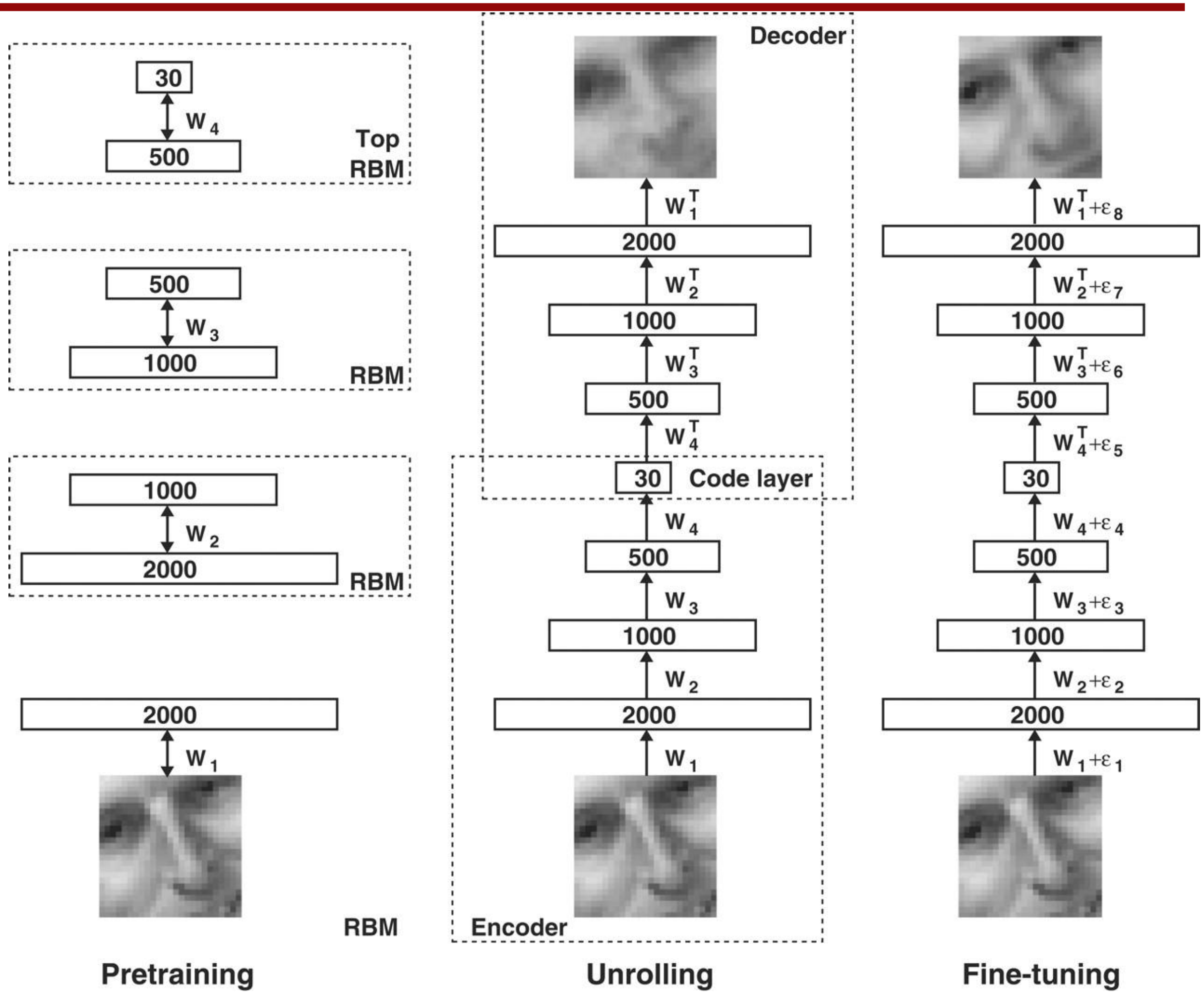
Remember the PCA model?



Latent Feature Extraction with Neural Networks



Latent Feature Extraction with Neural Networks



•G E Hinton, and R R Salakhutdinov Science 2006;313:504-507

Toolboxes on Component Analysis

Matlab Toolbox for Dimensionality Reduction

http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html

Matlab 2013b has PPCA implemented

http://www.mathworks.co.uk/help/stats/ppca.html

Toolboxes on Component Analysis

ICA toolboxes for image and signal processing:

http://www.bsp.brain.riken.jp/ICALAB/

ICA for EEG Analysis:

http://mialab.mrn.org/software/

FastICA

http://research.ics.aalto.fi/ica/fastica/