Dictionary Learning

CS 754

Problem statement

- Images or small patches from images can often be expressed as a linear combination of columns (or “atoms”) of a matrix.

$$\mathbf{y_i} = \mathbf{D}\mathbf{s_i} = \sum_{k=1}^{K} \mathbf{d_k} S_{ik}$$

Index for image/image-patch = i

- This matrix is called a **dictionary** and the coefficients of the linear combination are called **dictionary coefficients**.

Problem statement

- We have seen previously that these columns could belong to the DCT or wavelet basis matrix – these are called universal bases.

- But we can go one step beyond – we can infer (or learn) the dictionary from the data. This is called **dictionary learning**.

Problem statement

- The dictionary can be forced to obey a variety of constraints:

1. it may be orthonormal
2. it may be overcomplete (i.e. the number of columns exceeds the number of rows)
3. it may be non-negative.

- The learned dictionary can be useful in a variety of applications – compression, denoising, deblurring, inpainting as well as compressive recovery.

Dictionary learning algorithms

- We will study the following dictionary learning algorithms:

1. Principal Components Analysis (PCA)
2. Non-negative sparse coding
3. Method of optimal directions (MOD)
4. Union of orthonormal bases
5. K-SVD

PCA (Principal Components Analysis)

Principal Components Analysis (PCA)

- Consider *N* vectors (or points) $\mathbf{x_i}$, each containing *d* elements, each represented as a **column vector**.
- We say: $\forall i, \mathbf{x_i} \in R^d$. *d* could be very large – like 250,000 or more.
- Our aim is to extract some *k* features from each $\mathbf{x_i}$, $k << d$.
- Effectively we are projecting the original vectors from a *d*-dimensional space to a *k*-dimensional space. This is called **dimensionality reduction**. PCA is **one method** of dimensionality reduction.

PCA

- How do we pick the '*k*' appropriate features?

- We look into a notion of "compressibility" – how much can the data be compressed, allowing for some small errors.

PCA: Algorithm

1. Compute the mean of the given points:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i, \mathbf{x}_i \in R^d, \bar{\mathbf{x}} \in R^d$$

2. Deduct the mean from each point:

$$\bar{\mathbf{x}}_\mathbf{i} = \mathbf{x}_\mathbf{i} - \bar{\mathbf{x}}$$

3. Compute the covariance matrix of these mean-deducted points:

$$\mathbf{C} = \frac{1}{N-1}\sum_{i=1}^{N}\bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_\mathbf{i} - \bar{\mathbf{x}})(\mathbf{x}_\mathbf{i} - \bar{\mathbf{x}})^T, \mathit{Note}: \mathbf{C} \in R^{d\times d}$$

$\mathit{Note}: \mathbf{C}$ is a symmetric matrix, and it is positive-semidefinite

PCA: algorithm

4. Find the eigenvectors of **C**:

$\mathbf{CV} = \mathbf{V\Lambda}, \mathbf{V} \in R^{d \times d}, \mathbf{\Lambda} \in R^{d \times d}$

$\mathbf{V}$ – matrix of eigenvectors (each column is an eigenvector),

$\mathbf{\Lambda}$ – diagonal matrix of eigenvalues

Note: $\mathbf{V}$ is an orthonormal matrix (i.e. $\mathbf{VV}^T = \mathbf{V}^T\mathbf{V} = \mathbf{I}$), as $\mathbf{C}$ is a covariance matrix and hence it is symmetric.

Note: $\mathbf{\Lambda}$ contains non-negative values on the diagonal (eigen-values)

5. Extract the *k* eigenvectors corresponding to the *k* largest eigenvalues. This is called the extracted **eigenspace:**

$\hat{\mathbf{V}}_\mathbf{k} = \mathbf{V}(:,1:k)$

There is an implicit assumption here that the first *k* indices indeed correspond to the *k* largest eigenvalues. If that is not true, you would need to pick the appropriate indices.

PCA: algorithm

6. Project each point onto the eigenspace, giving a vector of *k* **eigen-coefficients** for that point.

$$\boldsymbol{\alpha}_{ik} = \hat{\mathbf{V}}_{\mathbf{k}}^{\mathbf{T}} \bar{\mathbf{x}}_{\mathbf{i}}, \boldsymbol{\alpha}_{ik} \in R^k; \boldsymbol{\alpha}_i = \mathbf{V}^T \bar{x}_i, \boldsymbol{\alpha}_i \in R^d$$

As $\mathbf{V}$ is orthonormal, we have

$$\bar{\mathbf{x}}_{\mathbf{i}} = \mathbf{V}\boldsymbol{\alpha}_{\mathbf{i}} = \mathbf{V}(:,1)\boldsymbol{\alpha}_i(1) + \mathbf{V}(:,2)\boldsymbol{\alpha}_i(2) + \ldots + \mathbf{V}(:,d)\boldsymbol{\alpha}_i(d)$$

$$\approx \hat{\mathbf{V}}_{\mathbf{k}}\boldsymbol{\alpha}_{ik} = \hat{\mathbf{V}}_{\mathbf{k}}(:,1)\boldsymbol{\alpha}_{ik}(1) + \hat{\mathbf{V}}_{\mathbf{k}}(:,2)\boldsymbol{\alpha}_{ik}(2) + \ldots + \hat{\mathbf{V}}_{\mathbf{k}}(:,k)\boldsymbol{\alpha}_{ik}(k)$$

We are representing each face as a linear combination of the *k* eigenvectors corresponding to the *k* largest eigenvalues. The coefficients of the linear combination are the eigen-coefficients.

Note that $\boldsymbol{\alpha}_{\mathbf{ik}}$ is a vector of the eigencoefficients of the *i*-th sample point, and it has *k* elements. The j-th element of this vector is denoted as $\boldsymbol{\alpha}_{\mathbf{ik}}$ (j).

PCA and Face Recognition: Eigen-faces

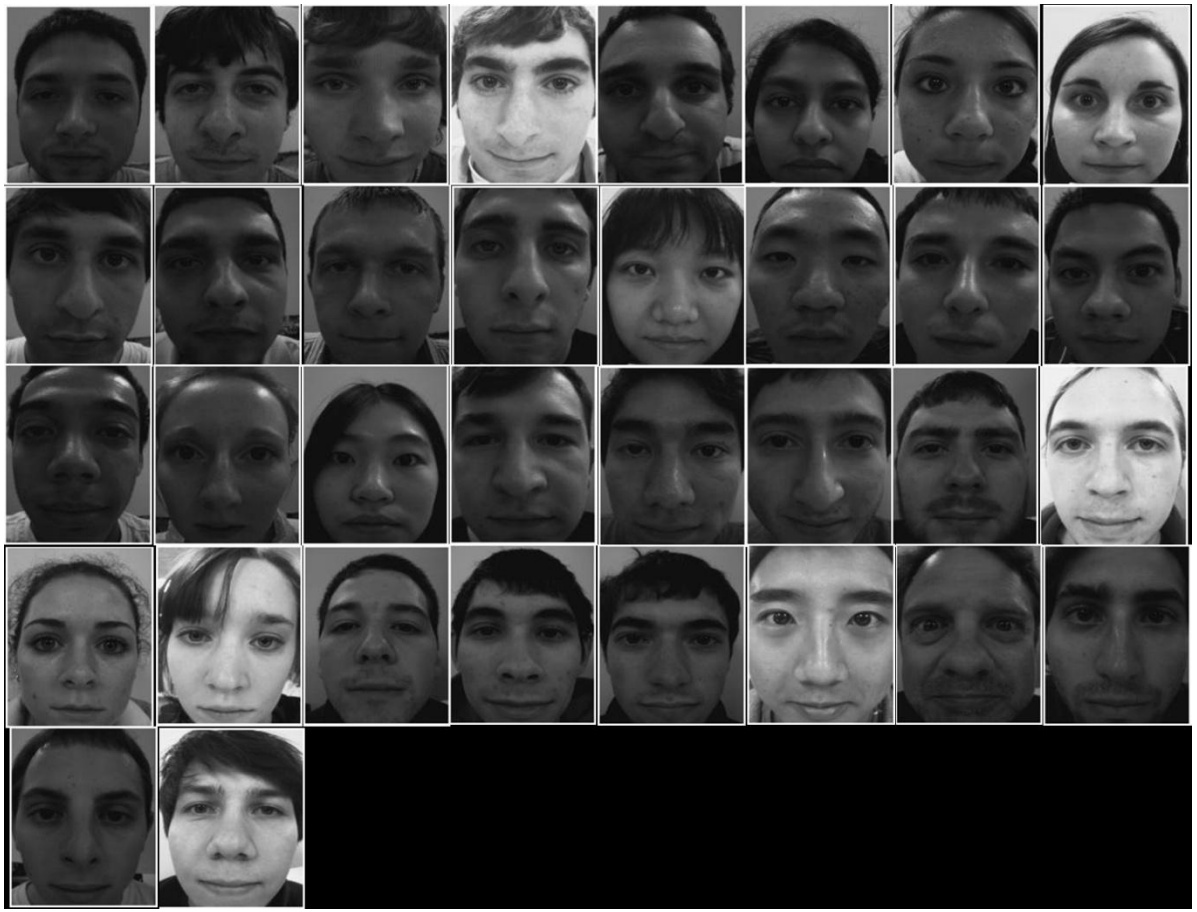
- Consider a database of cropped, frontal face images (which we will assume are aligned and under the same illumination). These are the **gallery images**.

- We will reshape each such image (a 2D array of size *H x W* after cropping) to form a column vector of $d = HW$ elements. Each image will be a vector $\mathbf{x}_i$, as per the notation on the previous two slides.

- And then carry out the six steps mentioned before.

- The eigenvectors that we get in this case are called **eigenfaces**. Each eigenvector has *d* elements. If you reshape those eigenvectors to form images of size *H x W*, those images **look like (filtered!) faces.**

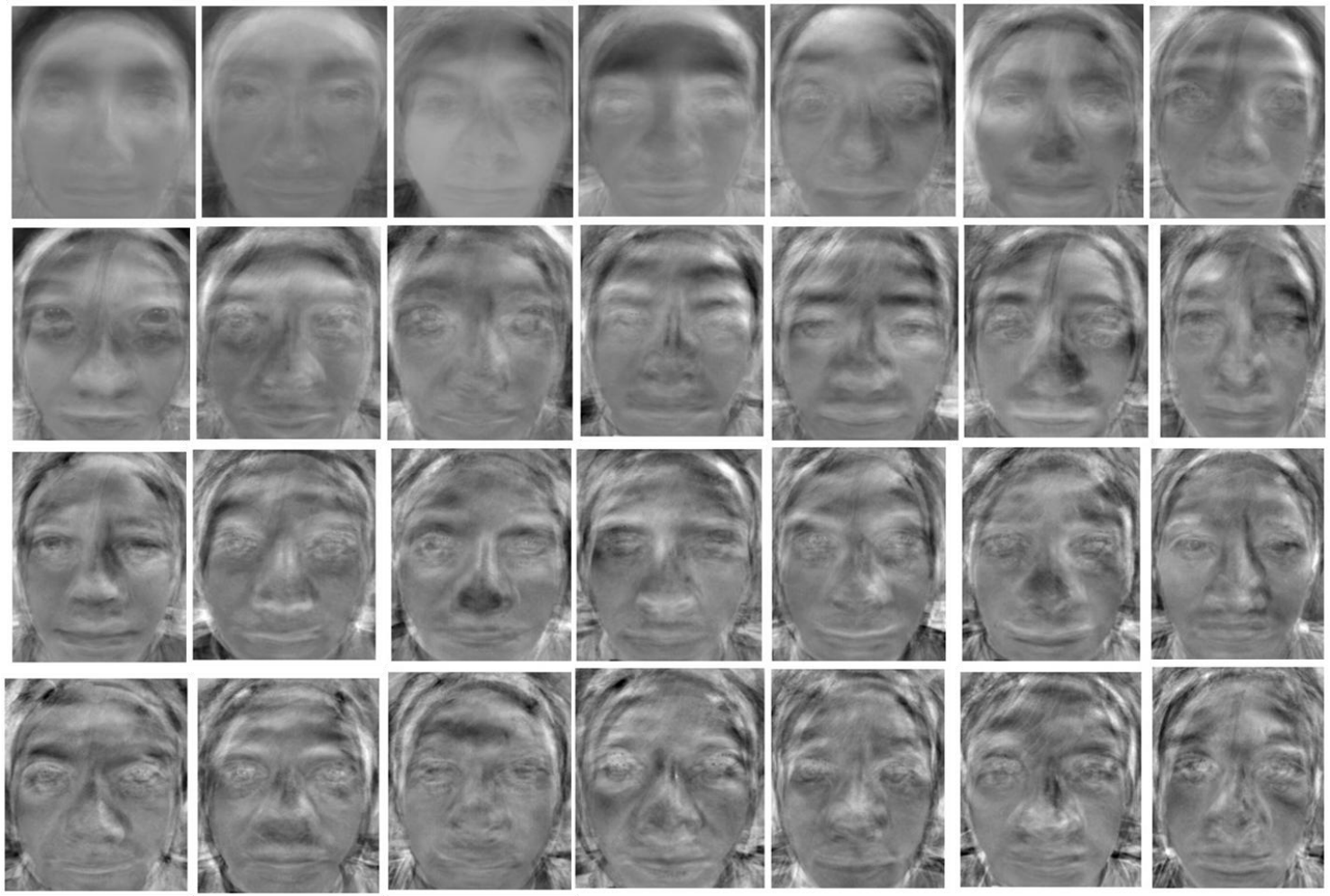
Example 1



A face database

http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/

Top 25 Eigen-faces for this database!



http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/

PCA and Face recognition: Eigenfaces

- For each gallery image, you compute the eigen-coefficients. You then store the eigen-coefficients and the identity of the person in a database.
- You also store $\hat{\mathbf{V}}_k, \bar{\mathbf{x}}$ in the database.
- During the testing phase, you are given a probe image (say) $\mathbf{z}_p$ in the form of a column vector of *HW* elements.
- You deduct the mean image from $\mathbf{z}_p$:

$$\bar{\mathbf{z}}_p = \mathbf{z}_p - \bar{\mathbf{x}}$$

PCA and Face recognition: Eigenfaces

- You then project the mean-deducted face image onto the eigen-space:

$$\boldsymbol{\alpha}_p = \hat{\mathbf{V}}_k^T \bar{\mathbf{z}}_p$$

Eigen-coefficients of the probe image $\mathbf{z}_p$.

- Now, compare $\boldsymbol{\alpha}_p$ with all the $\boldsymbol{\alpha}_{ik}$ (eigen-coefficients of the gallery images) in the database.

- Find the closest match in terms of the squared distance between the eigen-coefficients. That gives you the identity (see next slide).

PCA and Face recognition: Eigenfaces

$$j_p = \arg\min_l \left\| \boldsymbol{\alpha}_p - \boldsymbol{\alpha}_l \right\|_2^2$$

Eigen-coefficients of the probe image $\mathbf{z}_p$.

Eigen-coefficients of the l-th gallery image $\mathbf{x}_l$.

Note: other distance measures (different from sum of squared differences) may also be employed. One example is sum of absolute differences, given as follows: $\left\| \boldsymbol{\alpha}_p - \boldsymbol{\alpha}_l \right\|_1$

Another could be normalized dot product (and this distance measure should be maximized!): $\dfrac{\boldsymbol{\alpha}_p \bullet \boldsymbol{\alpha}_l}{\left\| \boldsymbol{\alpha}_p \right\|_2 \left\| \boldsymbol{\alpha}_l \right\|_2}$

PCA and Face recognition: eigenfaces

- The eigen-face images contain more and more high frequency information as the corresponding eigen-values decrease.
- Although PCA is a technique known for a long time, it's application in face recognition was pioneered by Turk and Pentland in a classic paper in 1991.

  M. Turk and A. Pentland (1991). Eigenfaces for recognition, *Journal of Cognitive Neuroscience*, 3(1): 71–86.

PCA and Face recognition: eigenfaces

- We can regard the *k* eigenfaces as key **signatures**.
- We express each face image as a **linear combination** of these eigenfaces, i.e. the average face + (say) 3 times eigenface 1 + (say) 5 times eigenface 2 + (say) -1 times eigenface 3 and so on. (note: 3,5,-1 here are the eigen-coefficients, and some of them can be negative).

$$\bar{\mathbf{x}}_\mathbf{i} = \mathbf{V}\boldsymbol{\alpha}_\mathbf{i} = \mathbf{V}(:,1)\boldsymbol{\alpha}_\mathbf{i}(1) + \mathbf{V}(:,2)\boldsymbol{\alpha}_\mathbf{i}(2) + \ldots + \mathbf{V}(:,d)\boldsymbol{\alpha}_\mathbf{i}(d)$$

$$\approx \hat{\mathbf{V}}_\mathbf{k}\boldsymbol{\alpha}_\mathbf{ik} = \hat{\mathbf{V}}_\mathbf{k}(:,1)\boldsymbol{\alpha}_\mathbf{ik}(1) + \hat{\mathbf{V}}_\mathbf{k}(:,2)\boldsymbol{\alpha}_\mathbf{ik}(2) + \ldots + \hat{\mathbf{V}}_\mathbf{k}(:,d)\boldsymbol{\alpha}_\mathbf{ik}(k)$$

One word of caution: Eigen-faces

- The algorithm described earlier is computationally infeasible for face images (or eigenfaces), as it requires storage of a d x d Covariance matrix (d – the number of image pixels - could be more than 10,000). And the computation of the eigen-vectors of such a matrix is a $O(d^3)$ operation!

- We will study a modification to this that will bring down the computational cost drastically.

Eigen-faces: reducing computational complexity.

- Consider the covariance matrix:

$$\mathbf{C} = \frac{1}{N-1}\sum_{i=1}^{N}\bar{\mathbf{x}}_i\bar{\mathbf{x}}_i^T = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i-\bar{\mathbf{x}})(\mathbf{x}_i-\bar{\mathbf{x}})^T, \; Note: \mathbf{C}\in R^{d\times d}$$

$$Note: \mathbf{C} \text{ is a symmetric matrix, and it is positive-semidefinite}$$

- It will require too much memory if *d* is large, and computing its eigenvectors will be a horrendous task!

- Consider the case when *N* is much less than *d*. This is very common in face recognition applications. The number of training images is usually much smaller than the size of the image.

Eigen-faces: reducing computational complexity.

- In such a case, the rank of **C** is at the most *N-1*. So **C** will have at the most *N-1* non-zero eigen-values.

- We can write **C** in the following way:

$$\mathbf{C} = \frac{1}{N-1}\sum_{i=1}^{N} \bar{\mathbf{x}}_i \bar{\mathbf{x}}_i^T \propto \mathbf{X}\mathbf{X}^T, \text{where}$$

$$\mathbf{X} = [\bar{\mathbf{x}}_1 \mid \bar{\mathbf{x}}_2 \mid \ldots \mid \bar{\mathbf{x}}_N] \in R^{d \times N}$$

Back to Eigen-faces: reducing computational complexity.

- Consider the matrix $\mathbf{X}^T\mathbf{X}$ (size N x N) instead of $\mathbf{X}\mathbf{X}^T$ (size d x d). Its eigenvectors are of the form:

$$\mathbf{X}^T\mathbf{X}\mathbf{w} = \lambda\mathbf{w}, \mathbf{w} \in R^N$$

$$\rightarrow \mathbf{X}\mathbf{X}^T(\mathbf{X}\mathbf{w}) = \lambda(\mathbf{X}\mathbf{w}) [\text{pre} - \text{multiplying by } \mathbf{X}]$$

Xw is an eigenvector of $\mathbf{C}=\mathbf{X}\mathbf{X}^T$! Computing all eigenvectors of **C** will now have a complexity of only $O(N^3)$ for computation of the eigenvectors of $\mathbf{X}^T\mathbf{X}$ + *O(N* x *dN)* for computation of **Xw** from each **w** = total of $O(N^3 + dN^2)$ which is much less than $O(d^3)$. Note that **C** has at most only *min(N-1,d)* eigenvectors corresponding to non-zero eigen-values (why?).

Eigenfaces: Algorithm (N << d case)

1. Compute the mean of the given points:

$$\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i, \mathbf{x}_i \in R^d, \bar{\mathbf{x}} \in R^d$$

2. Deduct the mean from each point:

$$\bar{\mathbf{x}}_\mathbf{i} = \mathbf{x}_\mathbf{i} - \bar{\mathbf{x}}$$

3. Compute the following matrix:

$$\mathbf{L} = \mathbf{X}^T\mathbf{X}, \mathbf{L} \in R^{N\times N}, \mathbf{X} = [\bar{\mathbf{x}}_\mathbf{1} \mid \bar{\mathbf{x}}_\mathbf{2} \mid \ldots \mid \bar{\mathbf{x}}_\mathbf{N}] \in R^{d\times N}$$

$$Note: \mathbf{L} \text{ is a symmetric matrix, and it is positive-semidefinite}$$

Eigen-faces: Algorithm (*N* << *d* case)

4. Find the eigenvectors of **L**:
   $\mathbf{LW} = \mathbf{W\Gamma}, \mathbf{W} - eigenvectors, \mathbf{\Gamma} - eigenvalues,$

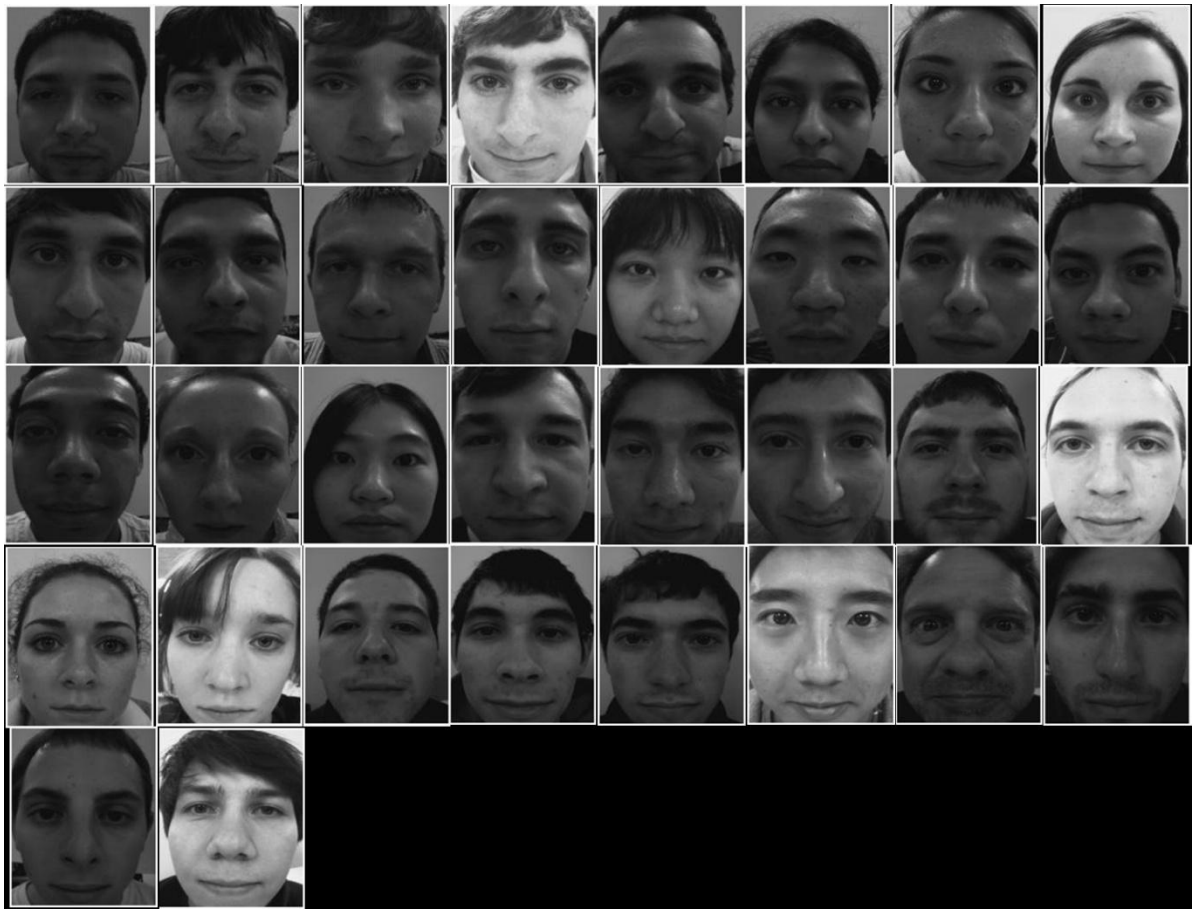
   $\mathbf{WW}^T = \mathbf{I}$
5. Obtain the eigenvectors of **C** from those of **L**:

   $\mathbf{V} = \mathbf{XW}, \mathbf{X} \in R^{d \times N}, \mathbf{W} \in R^{N \times N}, \mathbf{V} \in R^{N \times N}$

6. Unit-normalize the columns of **V**.
7. **C** will have at most only *N* eigenvectors corresponding to non-zero eigen-values*. Out of these you pick the top *k* (*k* < *N*) corresponding to the largest eigen-values.

* Actually this number is at most *N*-1 – this is due to the mean subtraction, else it would have been at most *N*.

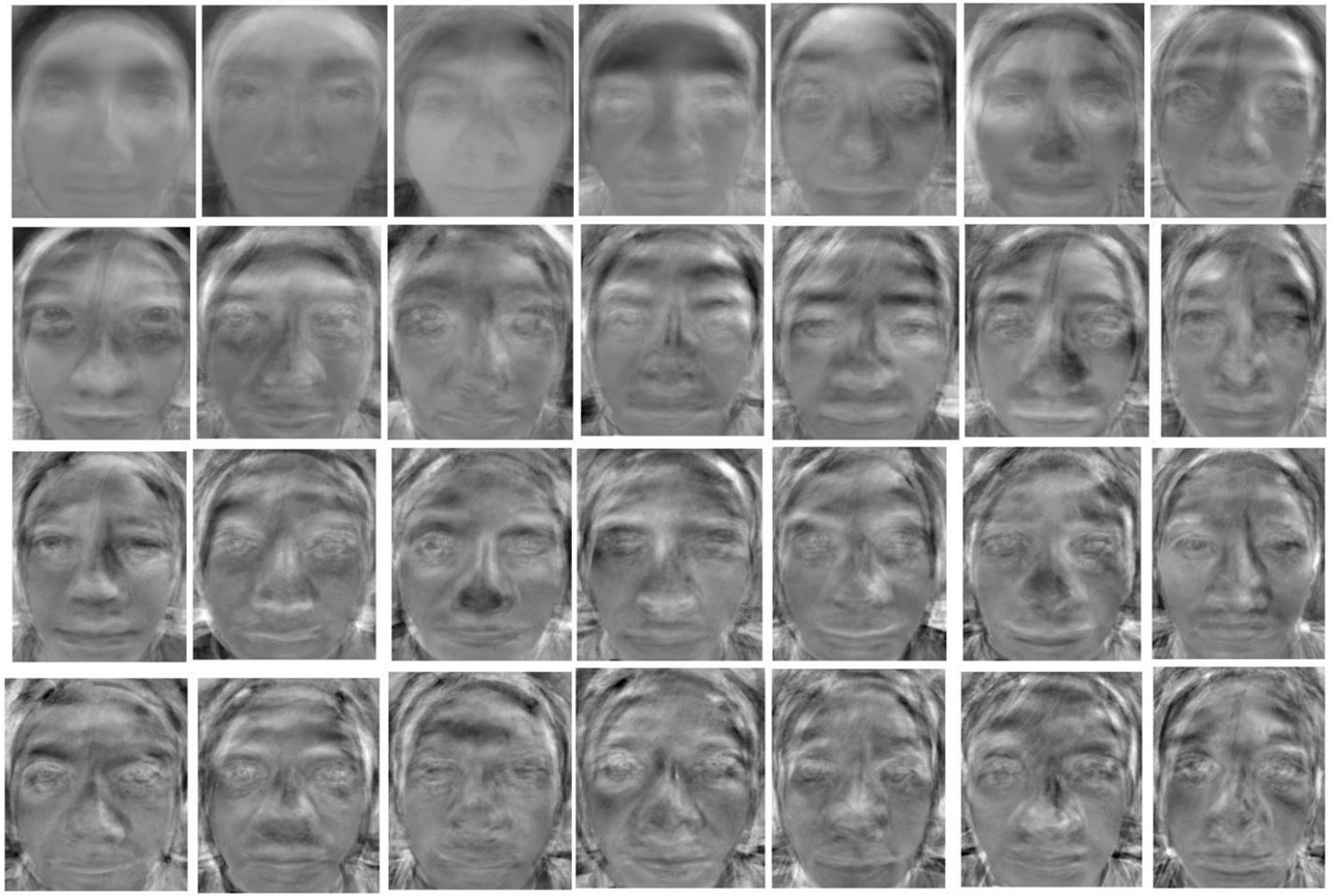
Example 1



A face database

http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/

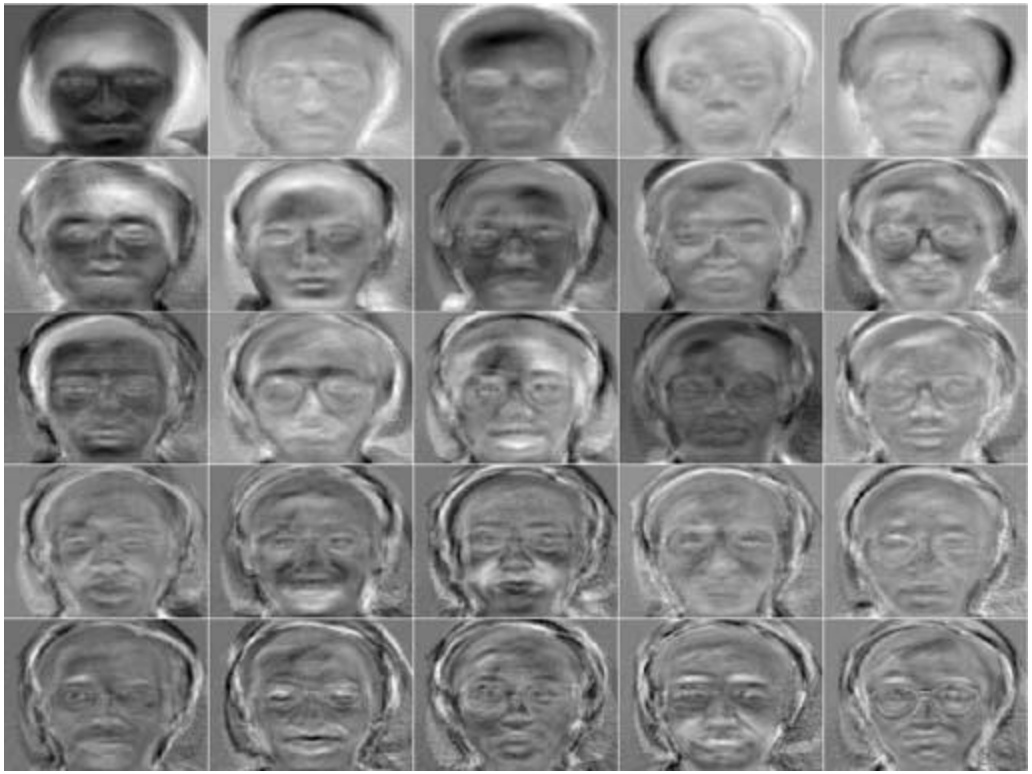
Top 25 Eigen-faces for this database!



http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/bjh78_caj65/bjh78_caj65/

Example 2

The Yale Face database



Top 25 eigenfaces from the previous database

Reconstruction of a face image using the top 1,8,16,32,…,104 eigenfaces (i.e. *k* varied from 1 to 104 in steps of 8)

$$\mathbf{x_i} = \bar{\mathbf{x}} + \hat{\mathbf{V}}\boldsymbol{\alpha}_\mathbf{i} \approx \bar{\mathbf{x}} + \sum_{l=1}^{k} \hat{\mathbf{V}}(:,l)\boldsymbol{\alpha}_{\mathbf{ik}}(l)$$

What if both *N* and *d* are large?

- This can happen, for example, if you wanted to build an eigenspace for face images of all people in Mumbai.
- Divide people into coherent groups based on some visual attributes (eg: gender, age group etc) and build separate eigenspaces for each group.

PCA: A closer look

- PCA has many applications – apart from face/object recognition – in image processing/computer vision, statistics, econometrics, finance, agriculture, and you name it!

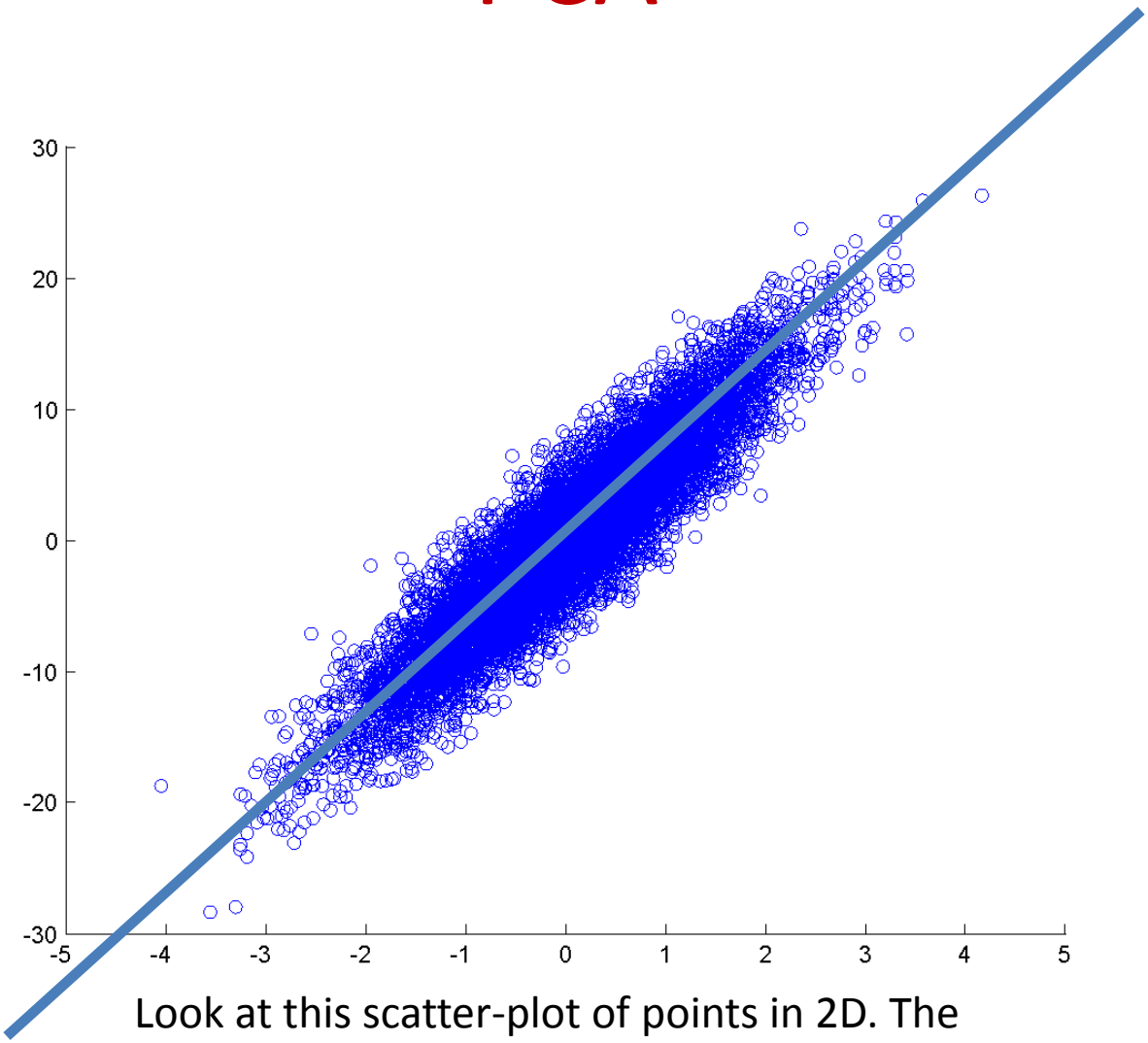
- Why PCA? What's special about PCA? See the next slides!

PCA: what does it do?

- It finds '*k*' perpendicular directions (all passing through the mean vector) such that the original data are approximated as **accurately** as possible when projected onto these '*k*' directions.

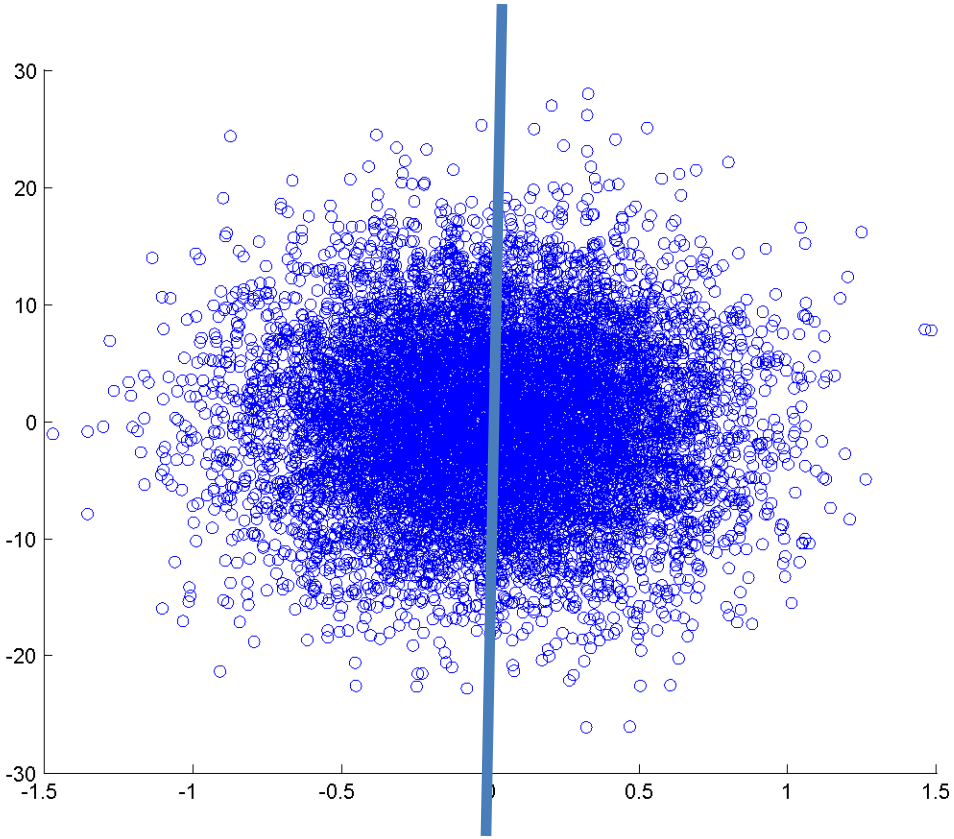
- We will see soon why these '*k*' directions are eigenvectors of the covariance matrix of the data!

PCA



Look at this scatter-plot of points in 2D. The points are highly spread out in the direction of the light blue line.

PCA



This is how the data would look if they were rotated in such a way that the major axis of the ellipse (the light blue line) now coincided with the Y axis. As the spread of the X coordinates is now relatively insignificant (**observe the axes!**), we can approximate the rotated data points by their projections onto the Y-axis (i.e. their Y coordinates alone!). This was not possible prior to rotation!

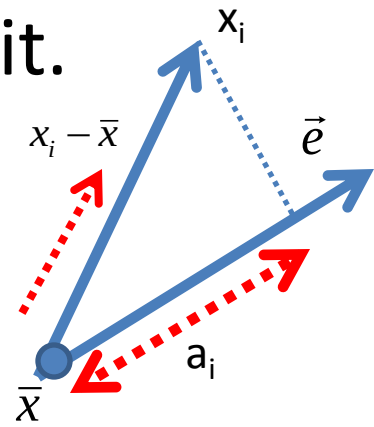
PCA

- As we could ignore the X-coordinates of the points post rotation and represent them just by the Y-coordinates, we have performed some sort of lossy data compression – or dimensionality reduction.

- The job of PCA is to perform such a rotation as shown on the previous two slides!

PCA

- **Aim of PCA:** Find the line $\vec{e}$ passing through the sample mean (i.e. $\bar{x}$), such that the projection of any mean-deducted point $x_i - \bar{x}$ onto $\vec{e}$, most accurately approximates it.



$$\text{Projection of } x_i - \bar{x} \text{ onto } \vec{e} \text{ is} = a_i\vec{e}, a_i \in R,$$

$$a_i = \vec{e}^T(x_i - \bar{x})$$

$$\text{Error of approximation} = \|(a_i\vec{e}) - (x_i - \bar{x})\|^2$$

Note: Here $\vec{e}$ is a **unit vector.**

PCA

- Summing up over all points, we get:

$$\text{Sum total error of approximation} = J(\vec{\mathbf{e}}) = \sum_{i=1}^{N} \left\| (a_i \vec{\mathbf{e}}) - (\mathbf{x_i} - \bar{\mathbf{x}}) \right\|^2$$

$$= \sum_{i=1}^{N} \left\| (a_i \vec{\mathbf{e}}) \right\|^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2 - 2 \sum_{i=1}^{N} a_i \vec{\mathbf{e}}^T (\mathbf{x_i} - \bar{\mathbf{x}})$$

$$= \sum_{i=1}^{N} \left\| (a_i \vec{\mathbf{e}}) \right\|^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2 - 2 \sum_{i=1}^{N} a_i \vec{\mathbf{e}}^T (\mathbf{x_i} - \bar{\mathbf{x}})$$

$$= \sum_{i=1}^{N} a_i^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2 - 2 \sum_{i=1}^{N} a_i \vec{\mathbf{e}}^T (\mathbf{x_i} - \bar{\mathbf{x}})$$

$$= \sum_{i=1}^{N} a_i^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2 - 2 \sum_{i=1}^{N} a_i^2, (\because a_i = \vec{\mathbf{e}}^T (\mathbf{x_i} - \bar{\mathbf{x}}))$$

$$= -\sum_{i=1}^{N} a_i^2 + \sum_{i=1}^{N} \left\| \mathbf{x_i} - \bar{\mathbf{x}} \right\|^2$$

PCA

$$\text{Sum total error of approximation} = J(\vec{\mathbf{e}}) = -\sum_{i=1}^{N} a_i^2 + \sum_{i=1}^{N} \|\mathbf{x_i} - \bar{\mathbf{x}}\|^2$$

$$= -\sum_{i=1}^{N} (\vec{\mathbf{e}}^t(\mathbf{x_i} - \bar{\mathbf{x}}))^2 + \sum_{i=1}^{N} \|\mathbf{x_i} - \bar{\mathbf{x}}\|^2$$

$$= -\sum_{i=1}^{N} \vec{\mathbf{e}}^t(\mathbf{x_i} - \bar{\mathbf{x}})(\mathbf{x_i} - \bar{\mathbf{x}})^t \vec{\mathbf{e}} + \sum_{i=1}^{N} \|\mathbf{x_i} - \bar{\mathbf{x}}\|^2$$

$$= -\vec{\mathbf{e}}^t \mathbf{S} \vec{\mathbf{e}} + \sum_{i=1}^{N} \|\mathbf{x_i} - \bar{\mathbf{x}}\|^2 \quad (\text{where } \mathbf{S} = (N-1)\mathbf{C})$$

This term is proportional to the variance of the data points when projected onto the direction **e**.

PCA

$$J(\vec{\mathbf{e}}) = -\vec{\mathbf{e}}^{\mathrm{t}}\mathbf{S}\vec{\mathbf{e}} + \sum_{i=1}^{N} \|\mathbf{x}_i - \bar{\mathbf{x}}\|^2$$

Independent of the direction e

Minimizing $J(\vec{\mathbf{e}})$ w.r.t. $\vec{\mathbf{e}}$ is equivalent to maximizing $\vec{\mathbf{e}}^{\mathrm{t}}\mathbf{S}\vec{\mathbf{e}}$ w.r.t. $\vec{\mathbf{e}}$.

We use the method of Lagrange multipliers to do so, while simultaneously imposing the constraint that $\vec{\mathbf{e}}^{t}\vec{\mathbf{e}} = 1$.

See appendix for details

So we have to take the derivative of the following modified function w.r.t. $\vec{\mathbf{e}}$ (and set it to 0)

$$\tilde{J}(\vec{e}) = \vec{\mathbf{e}}^{\mathrm{t}}\mathbf{S}\vec{\mathbf{e}} - \lambda(\vec{\mathbf{e}}^{t}\vec{\mathbf{e}} - 1)$$

Taking derivative of $\tilde{J}(\vec{\mathbf{e}})$ w.r.t. $\vec{\mathbf{e}}$ and setting it to 0, we get

$$\mathbf{S}\vec{\mathbf{e}} = \lambda\vec{\mathbf{e}},$$

so $\vec{\mathbf{e}}$ is an eigen - vector of $\mathbf{S}$.

As $\vec{\mathbf{e}}^{t}\mathbf{S}\vec{\mathbf{e}} = \lambda$ and we wish to maximize $\vec{\mathbf{e}}^{t}\mathbf{S}\vec{\mathbf{e}}$, we choose $\vec{\mathbf{e}}$ to be the eigen - vector corresponding to the maximum eigenvalue of $\mathbf{S}$.

PCA

- PCA thus projects the data onto that direction that minimizes the total squared difference between the data-points and their respective projections along that direction.

- This **equivalently** yields the direction along which the spread (or variance) will be maximum.

- Why? Note that the eigenvalue of a covariance matrix tells you the variance of the data when projected along that particular eigenvector:

$$\mathbf{S}\vec{\mathbf{e}} = \lambda\vec{\mathbf{e}} \rightarrow \vec{\mathbf{e}}^t\mathbf{S}\vec{\mathbf{e}} = \lambda$$

$$\vec{\mathbf{e}}^t\mathbf{S}\vec{\mathbf{e}} = \sum_{i=1}^{N}(\vec{\mathbf{e}}^t(\mathbf{x_i} - \bar{\mathbf{x}}))^2$$

This term is proportional to the variance of the data when projected along **e**.

PCA

- But for most applications (including face recognition), just a single direction is absolutely insufficient!

- We will need to project the data (from the high-dimensional, i.e. *d*-dimensional space) onto *k* (*k* << *d*) different mutually perpendicular directions.

- What is the criterion for deriving these directions?

- We seek those *k* directions for which the total reconstruction error of all the *N* images when projected on those directions is minimized.

PCA

- We seek those *k* directions for which the total reconstruction error of all the *N* images when projected on those directions is minimized.

$$J(\{\mathbf{e_j}\}_{j=1}^{k}) = \sum_{i=1}^{N} \left\| (\mathbf{x_i} - \bar{\mathbf{x}}) - \sum_{j=1}^{k} (\mathbf{e_j}^t (\mathbf{x_i} - \bar{\mathbf{x}}))\mathbf{e_j} \right\|_2^2$$

- One can prove that these *k* directions will be the eigenvectors of the **S** matrix (equivalently covariance matrix of the data) corresponding to the *k*-largest eigenvalues. These *k* directions form the eigen-space.

- If the eigenvalues of **S** are distinct, these *k* directions are defined uniquely (up to a sign factor)

PCA

- One can prove that these *k* directions will be the eigenvectors of the **S** matrix (equivalently covariance matrix of the data) corresponding to the *k*-largest eigenvalues. These *k* directions form the eigen-space.

- **Sketch of the proof:**

✓ Assume we have found $\mathbf{e}_1$ and are looking for $\mathbf{e}_2$ (where $\mathbf{e}_2$ is perpendicular to $\mathbf{e}_1$ and $\mathbf{e}_2$ has unit magnitude).

✓ Write out the objective function with the two constraints.

✓ Take the derivative of the objective function, set it to 0, and do some algebra to see that $\mathbf{e}_2$ is the eigenvector of **S** with the second largest eigenvalue.

✓ Proceed similarly for other directions.

Some observations about PCA for face images

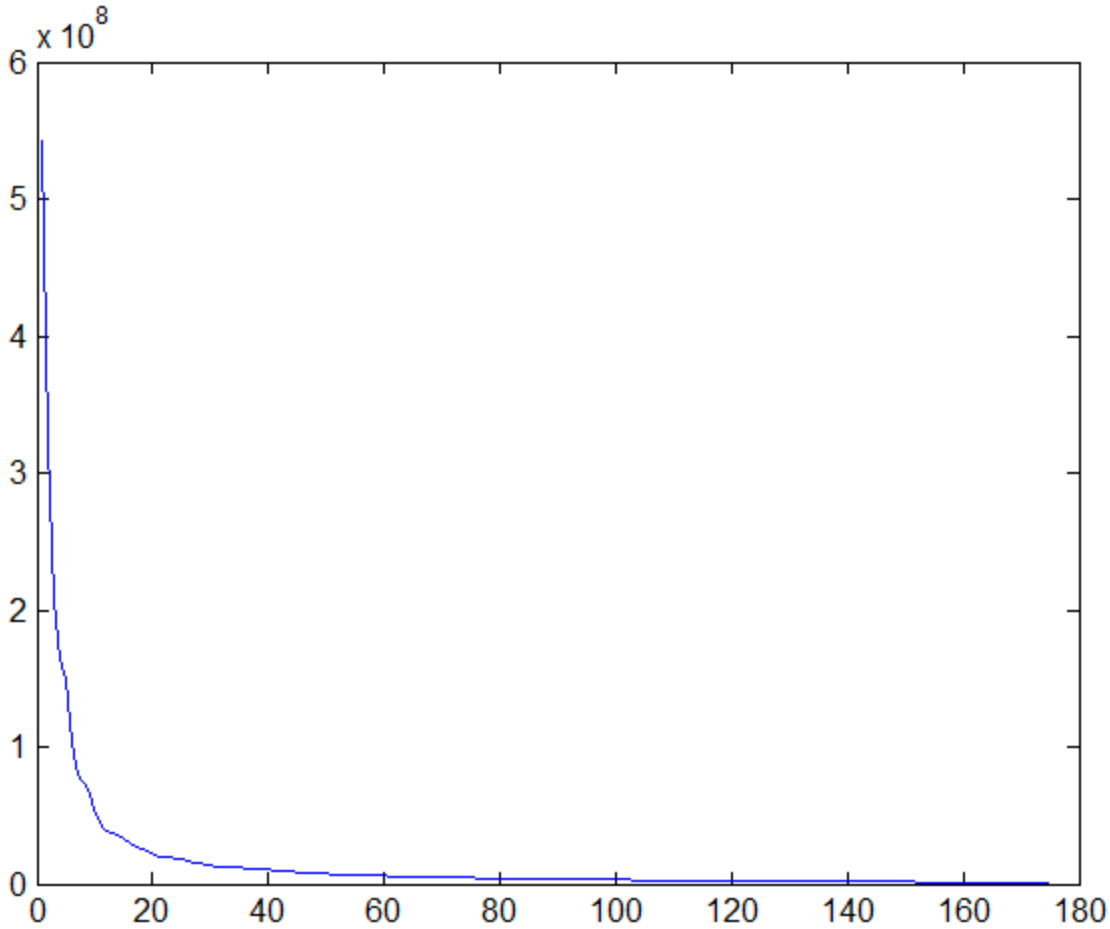
- The matrix **V** (with all columns) is orthonormal. Hence the squared error between any image and its approximation using just top *k* eigenvectors is given by:

$$\left\|\mathbf{x}_i - \tilde{\mathbf{x}}_i\right\|^2 = \left\|V(\boldsymbol{\alpha}_i - \tilde{\boldsymbol{\alpha}}_i)\right\|^2 (why?)$$

$$= \left\|\boldsymbol{\alpha}_i - \tilde{\boldsymbol{\alpha}}_i\right\|^2 (why?)$$

$$= \sum_{j=k+1}^{d} \boldsymbol{\alpha}_i^2 (why?)$$

This error is small ***on an average for a well-aligned group of face images*** – we will see why on the next slide.



The eigenvalues of the covariance matrix typically decay fast in value (if the faces were properly normalized). Note that the *j*-th eigenvalue is proportional to the variance of the j-th eigencoefficient, i.e.

$$\lambda_j = \mathbf{e}_\mathbf{j}^\mathbf{t}\mathbf{S}\mathbf{e}_\mathbf{j} = \sum_{i=1}^{N} \mathbf{e}_\mathbf{j}^\mathbf{t}(\mathbf{x}_\mathbf{i} - \bar{\mathbf{x}})(\mathbf{x}_\mathbf{i} - \bar{\mathbf{x}})^t\mathbf{e}_\mathbf{j} = (N-1)E(\alpha_{ij}^2)$$

What this means is that the data have low variance when projected along most of the eigenvectors, i.e. effectively the data are concentrated in a lower-dimensional subspace of the d-dimensional space.

PCA: Compression of a set of images

- Consider a database of *N* images that are "similar" (eg: all are face images, all are car images, etc.)
- Build an eigen-space from some subset of these images (could be all images, as well)
- We know that these images can often be reconstructed very well (i.e. with low error) using just a few eigenvectors.

PCA: Compression of a set of images

- Use this fact for image compression.
- Original data storage = d pixels x N images = Nd bytes (assume one byte per pixel intensity) = $8Nd$ bits.
- After PCA: Nk x number of bits to store each eigen-coefficient = $32Nk$ bits (remember $k << d$, example: $d \sim 250{,}000$ and $k \sim 100$).
- Plus storage of eigenvectors = $32dk$ bits (remember $k << M$ as well).
- Plus mean image = $8d$ bits.
- Total: $32(N+d)k + 8d$ bits

PCA: Compression of a set of images

- Example: N= 5000, d = 250000, k = 100
- Original size/(size after PCA compression) ~ 12.2.
- Note: we allocated 32 bits for every element of the eigen-vector. This is actually very conservative and you can have further savings using several tricks.

PCA: Compression of a set of images

- This differs a lot from JPEG compression.
- JPEG uses discrete cosine transform (DCT) as the basis. PCA tunes the basis to the underlying training data! If you change the training data, the basis changes!
- The performance of the PCA compression algorithm will depend on how compactly the eigen-space can represent the data.
- We will study image compression using JPEG and other standards later on in the course.

Which is the best orthonormal basis? Relationship between PCA and DCT

- Consider a set of *M* data-points (e.g. image patches in a vectorized form) represented as a linear combination of column vectors of an ortho-normal basis matrix:

$$\mathbf{q}_i = \mathbf{U}\boldsymbol{\theta}_i, \mathbf{q}_i \in R^{N\times 1}, \boldsymbol{\theta}_i \in R^{N\times 1}, \mathbf{U}\mathbf{U}^T = \mathbf{U}^T\mathbf{U} = \mathbf{I}$$

- Suppose we reconstruct each patch using only a subset of some *k* coefficients as follows:

$\tilde{\mathbf{q}}_i^{(k)} = \mathbf{U}\tilde{\boldsymbol{\theta}}_i$, where $\tilde{\boldsymbol{\theta}}_i$ is obtained by setting to 0 all except k coefficients (the same k coefficients are retained for all patches)

Which is the best orthonormal basis? Relationship between PCA and DCT

- For which orthonormal basis **U** is the following error the lowest:

$$E(\mathbf{U}) = \sum_{i=1}^{M} \left\| \tilde{\mathbf{q}}_i^{(k)} - \mathbf{q}_i \right\|^2$$

Which is the best orthonormal basis? Relationship between PCA and DCT

- The answer is the PCA basis, i.e. the set of *k* eigenvectors of the correlation matrix **C**, corresponding to the *k* largest eigen-values. Here is **C** is defined as:

$$\mathbf{C} = \frac{1}{M-1}\sum_{i=1}^{M}\mathbf{q}_i\mathbf{q}_i^T,$$

$$C_{kl} = \frac{1}{M-1}\sum_{i=1}^{M}q_{ikl}q_{ilk}$$

PCA: separable 2D version

- Find the correlation matrix $\mathbf{C}_R$ of row vectors from the patches.
- Find the correlation matrix $\mathbf{C}_C$ of column vectors from the patches.
- The final PCA basis is the Kronecker product of the individual bases:

$$\mathbf{C}_R = \frac{1}{M-1}\sum_{i=1}^{M}\sum_{j=1}^{n} q_i(j,:)'\, q_i(j,:);$$

$$[\mathbf{V}_R, \mathbf{D}_R] = eig(\mathbf{C}_R); q_i(j,:) \in R^{1\times n} - j^{th} \text{ row vector of } \mathbf{q_i}$$

$$\mathbf{C}_C = \frac{1}{M-1}\sum_{i=1}^{M}\sum_{j=1}^{n} q_i(:,j) q_i(:,j)';$$

$$[\mathbf{V}_C, \mathbf{D}_C] = eig(\mathbf{C}_C); q_i(:,j) \in R^{n\times 1} - j^{th} \text{ column vector of } \mathbf{q_i}$$

$$\mathbf{V} = \mathbf{V}_R \otimes \mathbf{V}_C; \mathbf{V}\mathbf{V}^T = I; \mathbf{V} \in R^{n^2\times n^2}, \mathbf{V}_R \in R^{n\times n}, \mathbf{V}_C \in R^{n\times n}, \mathbf{q_i} \in R^{n\times n}$$

But PCA is not used in JPEG, because…

- It is image-dependent, and the basis matrix would need to be computed afresh for each image.
- The basis matrix would need to be **stored** for each image.
- It is **expensive** to compute – $O(n^3)$ for a vector with n elements.

The DCT is used instead!

DCT and PCA

- DCT can be computed very fast using fft.
- It is universal – no need to store the DCT bases explicitly.
- DCT has very good energy compaction properties, only slightly worse than PCA.

Experiment

- Suppose you extract M ~ 100,000 small-sized (8 x 8) patches from a set of images.
- Compute the column-column and row-row correlation matrices.

$$\mathbf{C_C} = \frac{1}{M-1}\sum_{i=1}^{M}\mathbf{P}_i\mathbf{P}_i^T = \frac{1}{M-1}\sum_{i=1}^{M}\sum_{j=1}^{8}P_i(:,j)P_i(:,j)';$$

$$\mathbf{C_R} = \frac{1}{M-1}\sum_{i=1}^{M}\mathbf{P}_i^T\mathbf{P}_i = \frac{1}{M-1}\sum_{i=1}^{M}\sum_{j=1}^{8}P_i(j,:)'P_i(j,:);$$

See code here.

- Compute their eigenvectors $\mathbf{V_R}$ and $\mathbf{V_C}$.
- The eigenvectors will be very similar to the columns of the 1D-DCT matrix! (as evidenced by dot product values).
- Now compute the Kronecker product of $\mathbf{V_R}$ and $\mathbf{V_C}$ and call it $\mathbf{V}$. Reshape each column of $\mathbf{V}$ to form an image. These images will appear very similar to the DCT bases.

0.3536	0.4904	0.4619	0.4157	0.3536	0.2778	0.1913	0.0975
0.3536	0.4157	0.1913	-0.0975	-0.3536	-0.4904	-0.4619	-0.2778
0.3536	0.2778	-0.1913	-0.4904	-0.3536	0.0975	0.4619	0.4157
0.3536	0.0975	-0.4619	-0.2778	0.3536	0.4157	-0.1913	-0.4904
0.3536	-0.0975	-0.4619	0.2778	0.3536	-0.4157	-0.1913	0.4904
0.3536	-0.2778	-0.1913	0.4904	-0.3536	-0.0975	0.4619	-0.4157
0.3536	-0.4157	0.1913	0.0975	-0.3536	0.4904	-0.4619	0.2778
0.3536	-0.4904	0.4619	-0.4157	0.3536	-0.2778	0.1913	-0.0975

DCT matrix: dctmtx command from MATLAB (see code on website)

0.3517	-0.4493	-0.4278	0.4230	0.3754	0.3247	-0.2250	-0.1245
0.3534	-0.4366	-0.2276	-0.0110	-0.3078	-0.4746	0.4732	0.2975
0.3543	-0.3101	0.1728	-0.4830	-0.3989	0.0498	-0.4299	-0.4109
0.3546	-0.1115	0.4799	-0.3005	0.3342	0.4102	0.1856	0.4761
0.3547	0.1141	0.4823	0.2944	0.3301	-0.4182	0.1745	-0.4771
0.3543	0.3104	0.1771	0.4834	-0.3977	-0.0322	-0.4308	0.4103
0.3535	0.4357	-0.2319	0.0143	-0.3009	0.4656	0.4851	-0.2975
0.3520	0.4468	-0.4328	-0.4204	0.3686	-0.3253	-0.2342	0.1261

$\mathbf{V_C}$: Eigenvectors of column-column correlation matrix

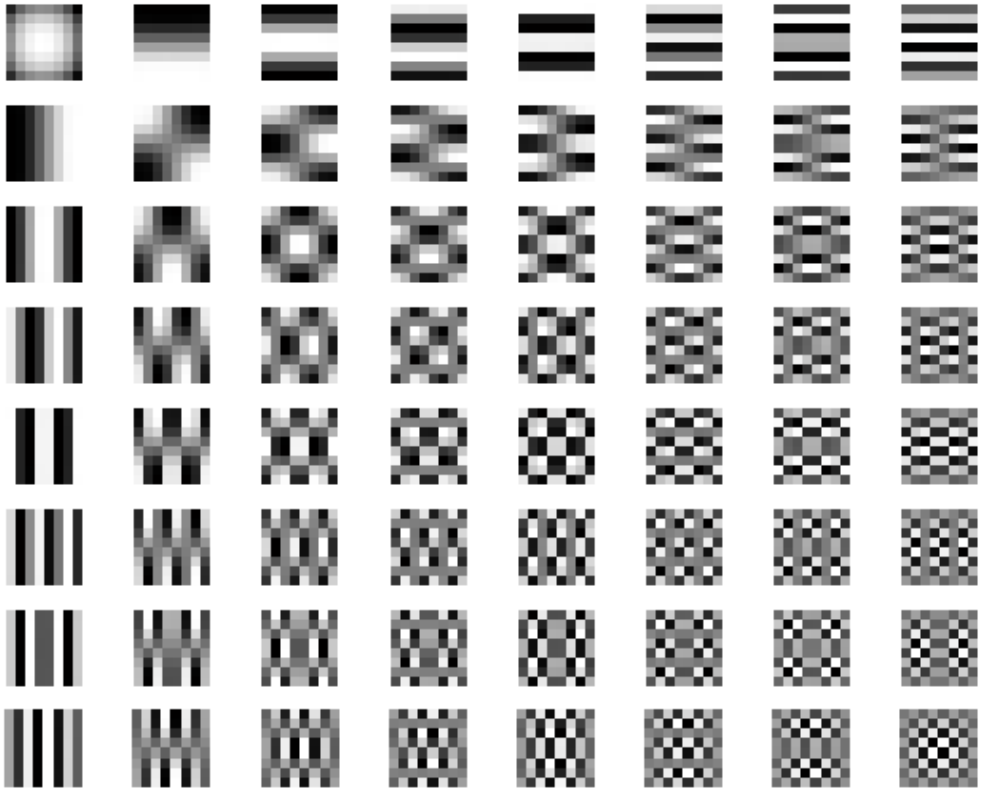
0.3520	-0.4461	-0.4305	0.4224	0.3696	0.3247	0.2342	0.1283
0.3537	-0.4338	-0.2345	-0.0114	-0.3000	-0.4671	-0.4814	-0.3028
0.3545	-0.3086	0.1662	-0.4896	-0.4007	0.0359	0.4261	0.4102
0.3548	-0.1145	0.4763	-0.3031	0.3339	0.4198	-0.1800	-0.4713
0.3548	0.1056	0.4839	0.2926	0.3349	-0.4194	-0.1766	0.4733
0.3543	0.3043	0.1863	0.4833	-0.4028	-0.0354	0.4269	-0.4097
0.3532	0.4389	-0.2269	0.0180	-0.3008	0.4654	-0.4811	0.3037
0.3512	0.4562	-0.4300	-0.4126	0.3694	-0.3242	0.2335	-0.1319

$\mathbf{V_R}$: Eigenvectors of row-row correlation matrix

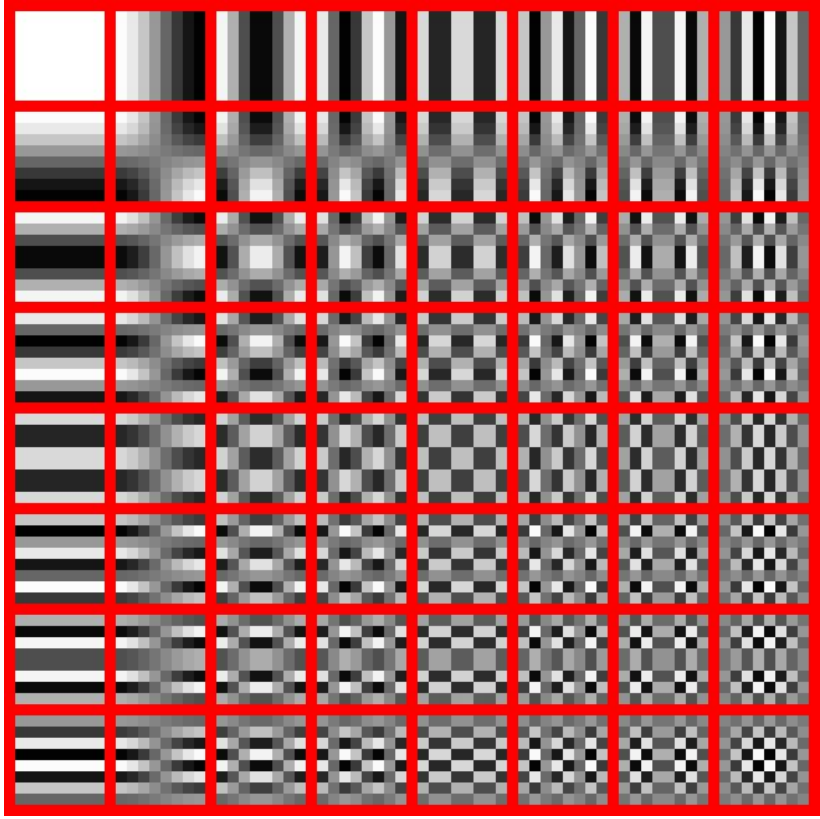
Absolute value of dot products between the columns of DCT matrix and columns of V_R (left) and V_C (right)

1.0000	0.0007	0.0032	0.0002	0.0013	0.0001	0.0005	0.0000
0.0007	**0.9970**	0.0097	0.0689	0.0009	0.0322	0.0003	0.0110
0.0033	0.0106	**0.9968**	0.0118	0.0713	0.0004	0.0334	0.0025
0.0002	0.0718	0.0124	**0.9926**	0.0007	0.0927	0.0017	0.0276
0.0010	0.0001	0.0737	0.0004	**0.9942**	0.0008	0.0780	0.0010
0.0000	0.0261	0.0015	0.0962	0.0005	**0.9934**	0.0011	0.0569
0.0003	0.0007	0.0276	0.0021	0.0802	0.0010	**0.9964**	0.0013
0.0000	0.0076	0.0026	0.0227	0.0012	0.0596	0.0015	**0.9979**

1.0000	0.0002	0.0029	0.0001	0.0010	0.0000	0.0004	0.0000
0.0002	**0.9965**	0.0028	0.0766	0.0005	0.0314	0.0009	0.0107
0.0029	0.0025	**0.9969**	0.0046	0.0728	0.0017	0.0304	0.0013
0.0001	0.0795	0.0044	**0.9923**	0.0029	0.0916	0.0015	0.0243
0.0008	0.0003	0.0747	0.0026	**0.9948**	0.0061	0.0696	0.0004
0.0000	0.0246	0.0021	0.0949	0.0069	**0.9940**	0.0131	0.0452
0.0003	0.0004	0.0252	0.0003	0.0715	0.0137	**0.9970**	0.0002
0.0000	0.0076	0.0013	0.0207	0.0001	0.0476	0.0009	**0.9986**



64 columns of **V** – each reshaped to form an 8 x 8 image, and rescaled to fit in the 0-1 range. Notice the similarity between the DCT bases and the columns of **V**. Again, **V** is the Kronecker product of $\mathbf{V}_R$ and $\mathbf{V}_C$.



DCT bases

DCT and PCA

- DCT is very close to PCA when the patches come from what is called as a **stationary first order Markov process**, i.e.

$$q_i = \rho q_{i-1} + \eta_i, \eta_i \sim N(0, \sigma_\eta^2), \rho < 1$$

$$E(q_i q_{i-1}) = \rho\sigma_q^2, E(q_i q_{i-2}) = \rho^2\sigma_q^2, \ldots, E(q_i q_{i-n+1}) = \rho^{n-1}\sigma_q^2,$$

$$\mathbf{C} = \sigma_q^2 \begin{pmatrix} 1 & \rho & \rho^2 & . & \rho^{n-1} \\ \rho & 1 & . & . & . \\ \rho^2 & . & . & . & . \\ . & . & . & . & . \\ \rho^{n-1} & . & . & . & 1 \end{pmatrix}, C_{ij} = E(q_i q_j)$$

DCT and PCA

- One can show that the eigenvectors of the covariance matrix of the form seen on the previous slide are the DCT basis vectors!

- Natural images approximate this first order Markov model, and hence DCT is almost as good as PCA for compression of a large ensemble of image patches.

- DCT has the advantage of being a universal basis and also the DCT coefficients are more efficiently computable than PCA coefficients (because DCT computation uses FFT).

More results from the previous experiment. See code here.

1.0000	0.9902	0.9795	0.9733	0.9682	0.9639	0.9604	0.9570
0.9902	1.0005	0.9908	0.9795	0.9734	0.9684	0.9643	0.9605
0.9795	0.9908	1.0010	0.9908	0.9796	0.9735	0.9689	0.9646
0.9733	0.9795	0.9908	1.0005	0.9904	0.9793	0.9735	0.9686
0.9682	0.9734	0.9796	0.9904	1.0004	0.9903	0.9794	0.9734
0.9639	0.9684	0.9735	0.9793	0.9903	1.0001	0.9903	0.9793
0.9604	0.9643	0.9689	0.9735	0.9794	0.9903	1.0004	0.9904
0.9570	0.9605	0.9646	0.9686	0.9734	0.9793	0.9904	1.0002

CR/CR(1,1) -
Notice it can be approximated by the form shown two slides before, with ρ~0.99

1.0000	0.9888	0.9770	0.9704	0.9648	0.9599	0.9554	0.9510
0.9888	1.0004	0.9891	0.9768	0.9703	0.9646	0.9596	0.9548
0.9770	0.9891	1.0004	0.9886	0.9764	0.9698	0.9640	0.9587
0.9704	0.9768	0.9886	0.9994	0.9878	0.9755	0.9687	0.9627
0.9648	0.9703	0.9764	0.9878	0.9986	0.9870	0.9746	0.9676
0.9599	0.9646	0.9698	0.9755	0.9870	0.9978	0.9861	0.9734
0.9554	0.9596	0.9640	0.9687	0.9746	0.9861	0.9967	0.9847
0.9510	0.9548	0.9587	0.9627	0.9676	0.9734	0.9847	0.9951

CC/CC(1,1) -
Notice it can be approximated by the form shown two slides before, with ρ~0.9888

Non-negative matrix factorization (NMF) and non-negative sparse coding (NNSC)

NMF

- Consider a matrix $\mathbf{Y} \in R^{n \times N}$, each column of which is a vectorised patch of n elements, i.e. $\mathbf{Y} = [\mathbf{y_1}|\mathbf{y_2}|\ldots|\mathbf{y_N}]$.

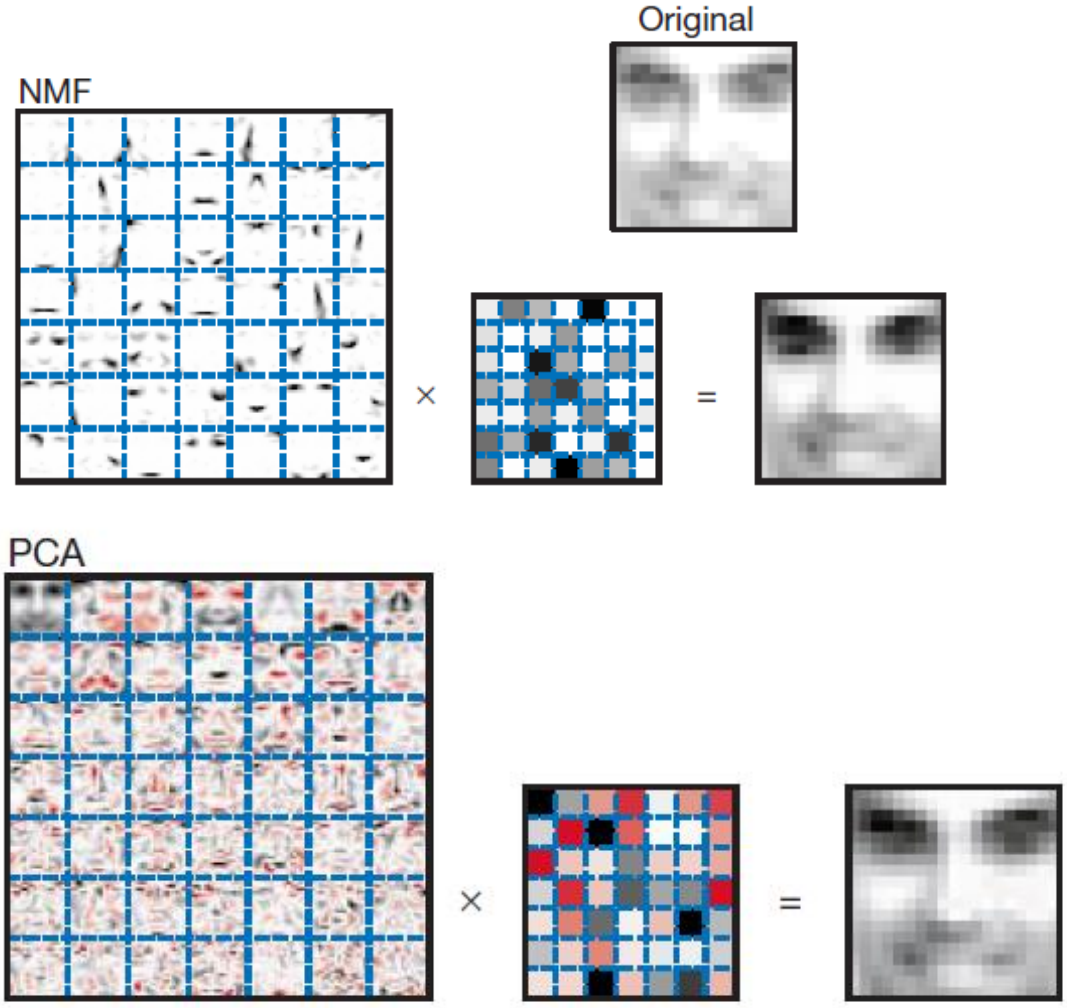
- Then NMF seeks to factorize **Y** into the product of matrices **W** (size n x r) and **H** (size r x N) which are both non-negative, such that $\mathbf{Y} \approx \mathbf{WH}$.

- Note: non-negativity means that every element of **W**, **H** must be non-negative.

NMF

- The columns of **W** are basis vectors and the values in the *i*-th column of **H** are coefficients.
- We seek to approximate each vector $\mathbf{y}_i$ with a linear combination of the columns of **W**.
- Usually, *r* is chosen to be smaller than *n* or *N*, which gives good model compression.
- The speciality of NMF is that it allows only additive combinations of the columns of **W** to approximate **Y**.
- This is in tune with the intuition of combining different parts to form a whole.

PCA versus NMF bases

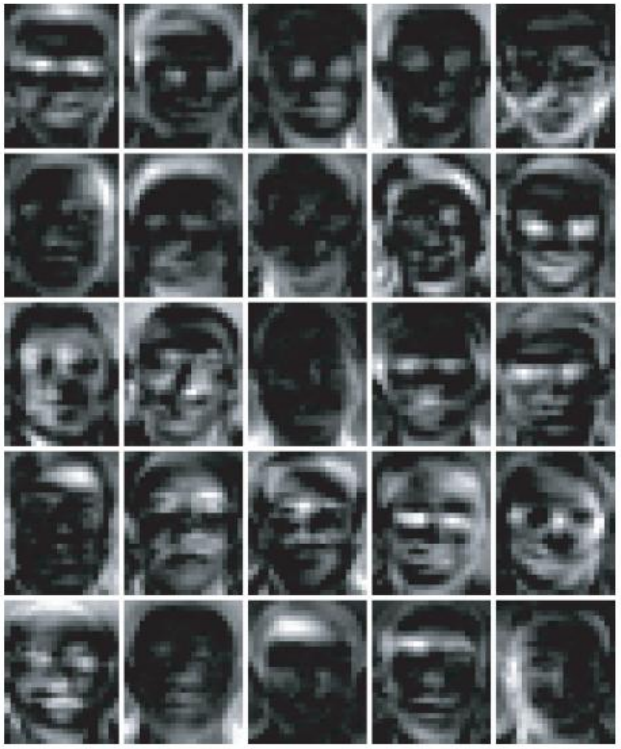


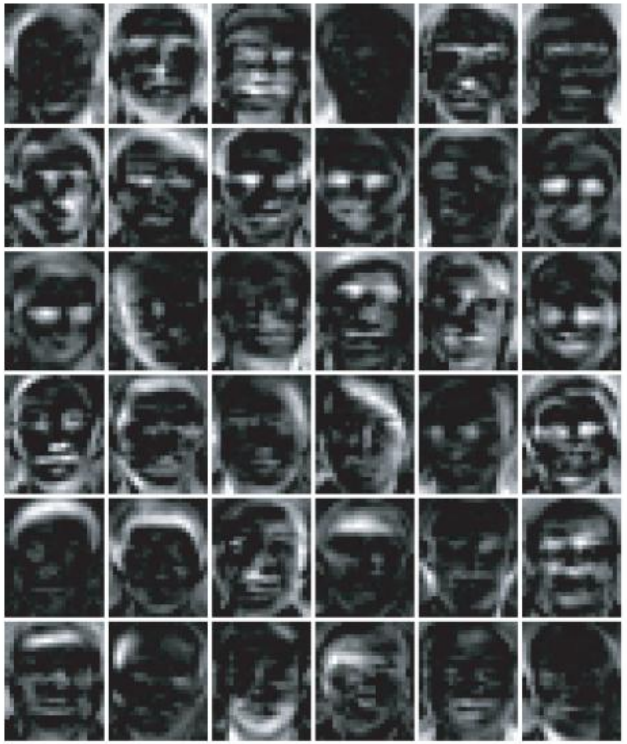
Source:
Lee and Seung, "Learning the parts of objects by non-negative matrix factorization", http://lsa.colorado.edu/LexicalSemantics/seung-nonneg-matrix.pdf

NMF bases are non-negative unlike eigenfaces. In that sense, each basis image from NMF is more like a model face than from the eigenfaces.

In practice they do not appear as sparse as seen in this figure, but more like what you will see on the next slide.

Source:
Wang et al, NMF framework for face recognition, http://www.cs.ucsb.edu/~mturk/pubs/IJPRAI05.pdf





NMF: Objective function

- The objective function to be minimized is:

$$E(\boldsymbol{W}, \boldsymbol{H}) = \left\| \boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H} \right\|_F^2 \text{ such that } \boldsymbol{W} \geq \boldsymbol{0}, \boldsymbol{H} \geq \boldsymbol{0}$$

- Method of alternating optimization using projected gradient descent with adaptive step size:

$$\text{Fix } \boldsymbol{H}, \boldsymbol{W} \leftarrow \boldsymbol{W} + \delta 2(\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H})(\boldsymbol{H}^T) \text{ such that } \boldsymbol{W} \geq \boldsymbol{0}$$

$$\text{Fix } \boldsymbol{W}, \boldsymbol{H} \leftarrow \boldsymbol{H} + \delta 2\boldsymbol{W}^T(\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H}) \text{ such that } \boldsymbol{H} \geq \boldsymbol{0}$$

NMF: Multiplicative updates

- The objective function to be minimized is:

$$E(\boldsymbol{W}, \boldsymbol{H}) = \left\| \boldsymbol{Y} - \boldsymbol{WH} \right\|_F^2 \text{ such that } \boldsymbol{W} \geq \boldsymbol{0}, \boldsymbol{H} \geq \boldsymbol{0}$$

- However there are also multiplicative updates of the following form:

$$H_{ab} = H_{ab} \frac{(W^T Y)_{ab}}{(W^T WH)_{ab}}, 1 \leq a \leq r, 1 \leq b \leq N$$

$$W_{ia} = W_{ia} \frac{(YH^T)_{ia}}{(WHH^T)_{ia}}, 1 \leq a \leq r, 1 \leq b \leq N$$

NMF: Multiplicative updates

- The multiplicative updates maintain the non-negativity of **W** and **H** and converge to a stationary point of E(**W**,**H**).

- The multiplicative updates also guarantee decrease of the cost function. They are based on majorization-minimization.

Non-negative sparse coding

- An extension to NMF with sparseness penalty imposed on the coefficients.
- The objective function to be minimized is:

$$E(\boldsymbol{W},\boldsymbol{H}) = \|\boldsymbol{Y} - \boldsymbol{W}\boldsymbol{H}\|_F^2 + \lambda\|\boldsymbol{H}\|_1$$

$$\text{such that } \boldsymbol{W} \geq 0, \boldsymbol{H} \geq 0, \boldsymbol{w}_i^t \boldsymbol{w}_i = 1 \text{ for } 1 \leq i \leq r$$

- Need to be careful to avoid multiple solutions – also impose unit norm constraint on all columns of **W**.

Non-negative sparse coding

- Algorithm:

Fix H,

$$W \leftarrow W + \delta 2(Y - WH)(H^T)$$

Set negative values of W to 0 and

unit normalize its columns so that $w_i^t w_i = 1$ for $1 \le i \le r$

Fix W,

$$H \leftarrow H + \delta 2W^T(Y - WH) + \lambda$$

Set negative values of H to 0

Results

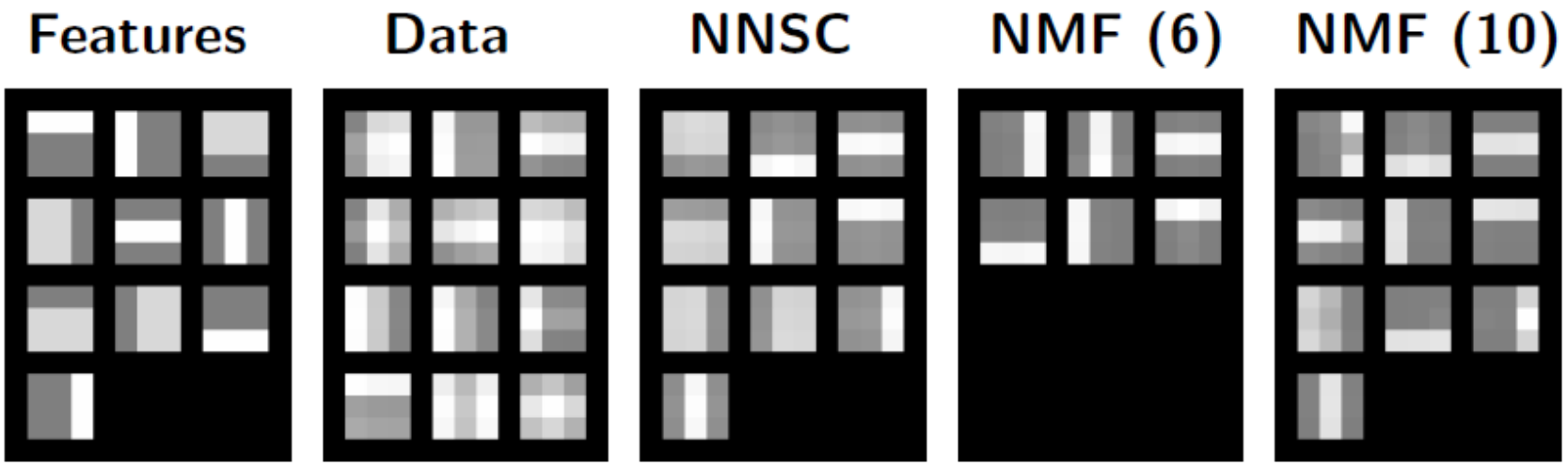


Figure 1: Experiments on bars data. Features: **The 10 original features that were used to construct the dataset.** Data: **A random sample of 12 data vectors. These constitute superpositions of the original features.** NNSC: **Features learned by NNSC, with dimensionality of the hidden representation equal to 10, starting from random initial values.** NMF (6): **Features learned by NMF, with dimensionality 6.** NMF (10): **Features learned by NMF, with dimensionality 10. See main text for discussion.**

https://arxiv.org/pdf/cs/0202009.pdf

NNSC and Poisson Data

- The noise affecting real-world images is known to follow the Poisson distribution.
- The Poisson distribution is given as follows:

$$P(X = i) = \frac{\lambda^i}{i!} e^{-\lambda}$$

X is a Poisson random variable, *i* is a non-negative integer, λ is the mean of the Poisson distribution

- For such a distribution, the mean is equal to the variance.

NNSC and Poisson Data

- Consider an image I, a Poisson-corrupted version of this image is obtained as $J_i \sim \text{Poisson}(I_i)$ where I_i is the average intensity (photon flux) at pixel i.
- Note that the noisy intensity at any pixel location i is always a non-negative integer, drawn from a Poisson distribution having mean (hence variance) I_i.

Image quality at 9 different average rates of photon emission.
https://en.wikipedia.org/wiki/Shot_noise

NNSC and Poisson denoising

- NNSC lends itself naturally to Poisson denoising due to the non-negativity constraints – on measurements as well as data (mean of Poisson = photon rate).
- Consider **Y** ~ Poisson(**WH**), i.e. y_{kl} ~ Poisson($[WH]_{kl}$), 1 <= *k* <= *n*, 1 <= *l* <= *N*. We have $[WH]_{kl} = [Wh_l]_k$.
- The objective function to be minimized is as follows:

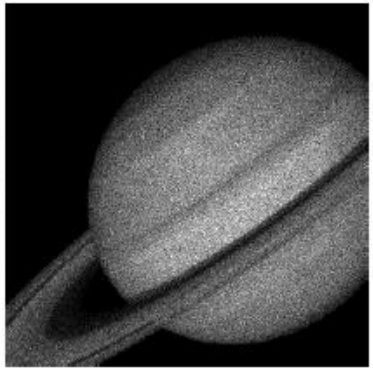
$$E(W,H) = \sum_{k=1}^{n}\sum_{l=1}^{N} - y_{kl}\log(Wh_l)_k + (Wh_l)_k + \rho\sum_{l=1}^{N}\sum_{c=1}^{r} h_{cl}$$

$$\text{such that } W \geq 0, H \geq 0, w_i^t w_i = 1 \text{ for } 1 \leq i \leq r$$

Non-negative Poisson likelihood where y_{kl} is a noisy pixel

Sparsity on coefficients

(a) Original image



(b) Noisy image, PSNR = 21.39

(c) Our algorithm, PSNR = 32.88

Figure 3.1: Saturn image, Peak = 50

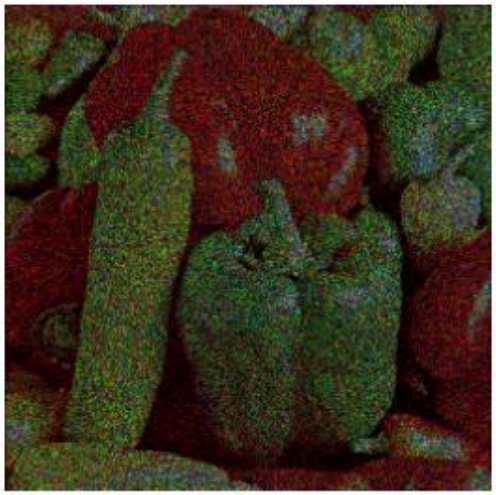
(a) Original image

(b) Noisy image, PSNR = 19.71

(c) Our algorithm, PSNR = 28.63

Figure 3.2: House image, Peak = 50

(a) Original image

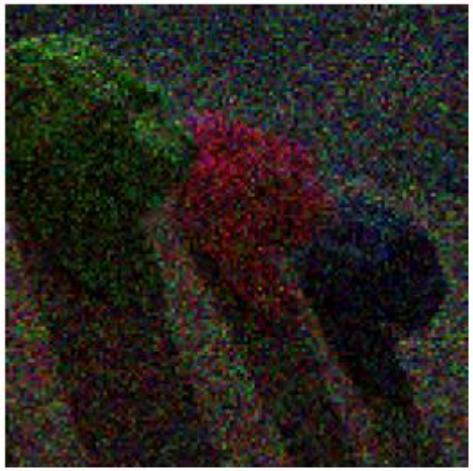


(b) Noisy image, PSNR = 10.37

(c) Our algorithm, PSNR = 21.40

Figure 3.3: Pepper image, Peak = 5

(a) Original image



(b) Noisy image, PSNR = 11.68

(c) Our algorithm, PSNR = 23.15

A comment on image processing under Poisson noise

- The relative error due to Poisson noise is equal to standard deviation divided by mean = $(\lambda)^{-0.5}$.

- At low photon counts , i.e. low λ, the noise variance is less, but its relative effect is more, and hence the problem is more challenging!

- This is unlike the signal independent additive iid Gaussian noise!

Learning Overcomplete Dictionaries

Introduction: Complete and over-complete dictionaries

- Signals are often represented as a linear combination of basis functions (e.g. Fourier or wavelet representation).
- The basis functions always have the same dimensionality as the (discrete) signals they represent.
- The number of basis vectors is traditionally the same as the dimensionality of the signals they represent.
- These bases may be orthonormal (Fourier, wavelet, PCA) or may not be orthonormal (in fact any full rank matrix can be used to represent signals – albeit not compactly).

Introduction: Complete and over-complete bases

- A more general representation for signals uses so called "over-complete dictionaries", where the number of basis functions is MORE than the dimensionality of the signals.
- Complete and over-complete bases:

$$\mathbf{x} = \mathbf{A}\mathbf{s}$$

$$\mathbf{x} \in R^n, \mathbf{A} \in R^{n \times n}$$

$$\mathbf{s} \in R^n$$

$$\mathbf{x} = \mathbf{A}\mathbf{s}$$

$$\mathbf{x} \in R^n, \mathbf{A} \in R^{n \times m}$$

$$\mathbf{s} \in R^m (m > n)$$

Introduction: Construction of over-complete bases

- Over-complete bases can be created by union of multiple sets of complete bases.
- Example 1: A signal with *n* values can be represented using a union of *n* x *n* Fourier and *n* x *n* Haar wavelet bases, yielding a *n* x *2n* basis matrix.
- Example 2: A signal with *n* values can be represented by adding sinusoids of more frequencies to an existing Fourier basis matrix with *n* vectors.

Introduction: uniqueness? ☹

- With complete bases, the representation of the signal is always unique.
- Example: Signals are uniquely defined by their wavelet or Fourier transforms, the eigen-coefficients of any signal (given PCA basis) are uniquely defined.
- This uniqueness is LOST with over-complete basis.

Introduction: compactness! ☺

- Advantage: over-complete bases afford much greater compactness in signal representation.
- Example: Consider two types of audio-signals – whistles and claps. Signals of either type can be represented in a complete Fourier or wavelet basis (power-law of compressibility will apply).
- BUT: imagine two complete bases respectively learned for whistles and claps – $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$.

Introduction: compactness! ☺

- Suppose $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ are such that a whistle (resp. clap) signal will likely have a compact representation in the whistle (resp. clap) basis and not in the other one.
- A whistle+clap signal will NOT have a compact representation in either basis - $\boldsymbol{B_1}$ or $\boldsymbol{B_2}$!
- But the whistle+clap signal WILL have a compact representation in the overcomplete basis $\boldsymbol{B = [B_1\ B_2]}$.
- Another (and simpler) example: cosines and spikes!

More problems ☹

- Since a signal can have many representations in an over-complete basis, which one do we pick?
- **Pick the sparsest one**, i.e. the one with least number of non-zero elements which either perfectly reconstructs the signal, or reconstructs the signal up to some error.
- Finding the sparsest representation for a signal in an over-complete basis is an NP-hard problem (i.e. the best known algorithm for this task has exponential time complexity ☹)

More problems ☹

- In other words, the following problem is NP-hard:

$$x = As$$

$$x \in R^n, A \in R^{n \times m}$$

$$s \in R^m (m > n)$$

$$s^* = \min \|s\|_0 \text{ subject to } x = As$$

Solving (?) those problems

- The NP-hard problem has several methods for approximation – basis pursuit (BP), matching pursuit (MP), orthogonal matching pursuit (OMP) and many others.
- None of them will give you the sparsest solution always – but they will yield the sparsest solutions under certain conditions on the overcomplete dictionary **A**.

Learning the dictionaries

- So far we assumed that the basis (i.e. **A**) was fixed, and optimized for the sparse representation.
- Now, we need to learn **A** as well – along with the sparse codes!
- We've learned about PCA – but it gives an orthonormal basis.

Learning the dictionary: analogy with K-means

- In K-means, we start with a bunch of *K* cluster centers and assign each point in the dataset to the nearest cluster center.
- The cluster centers are re-computed by taking the mean of all points assigned to a cluster.
- The assignment and cluster-center computation problems are iterated until a convergence criterion is met.

Learning the bases: analogy with K-means

- K-means is a special sparse coding problem where each point is represented by strictly one of K dictionary elements.
- Our dictionary (or bases) learning problem is more complex: we are trying to express each point as a linear combination of a ***subset*** of dictionary elements (or a ***sparse*** linear combination of dictionary elements).

Learning the bases: Method 1 (Olshausen and Field, 1996)

$x_k = k^{th}$ signal; $x_k = As_k + \text{noise}$

$$\{A, s_k\} = \arg\max_{A,s_k} \prod_k p(A, s_k \mid x_k) = \arg\max_A \prod_k p(x_k \mid A, s_k)\, p(A) p(s_k \mid A)$$

$$= \arg\min_{A,s_k} \sum_k \log p(s_k \mid A) + \log\, p(x_k \mid A, s_k)$$

Keeping s_k fixed, optimize on A using the foll. estimator

$$\arg\min_A \sum_{k=1}^{N} (\lambda \|s_k\|_1 + \|x_k - As_k\|^2)$$

Gradient descent

$$A^{(t+1)} = A^{(t)} - \eta \sum_{k=1}^{N} (A^{(t)} s_k - x_k) s_k^T$$

Uniform prior on **A**

Gaussian likelihood, i.e. assuming Gaussian noise

Laplacian prior on $\mathbf{s_k}$ (given any **A**)

$$p(s_k) \propto e^{-\lambda \|s_k\|_1};\; p(x_k \mid A, s_k) \propto e^{-\|x_k - As_k\|_2^2 / (2\sigma^2)}$$

Two-step iterative procedure

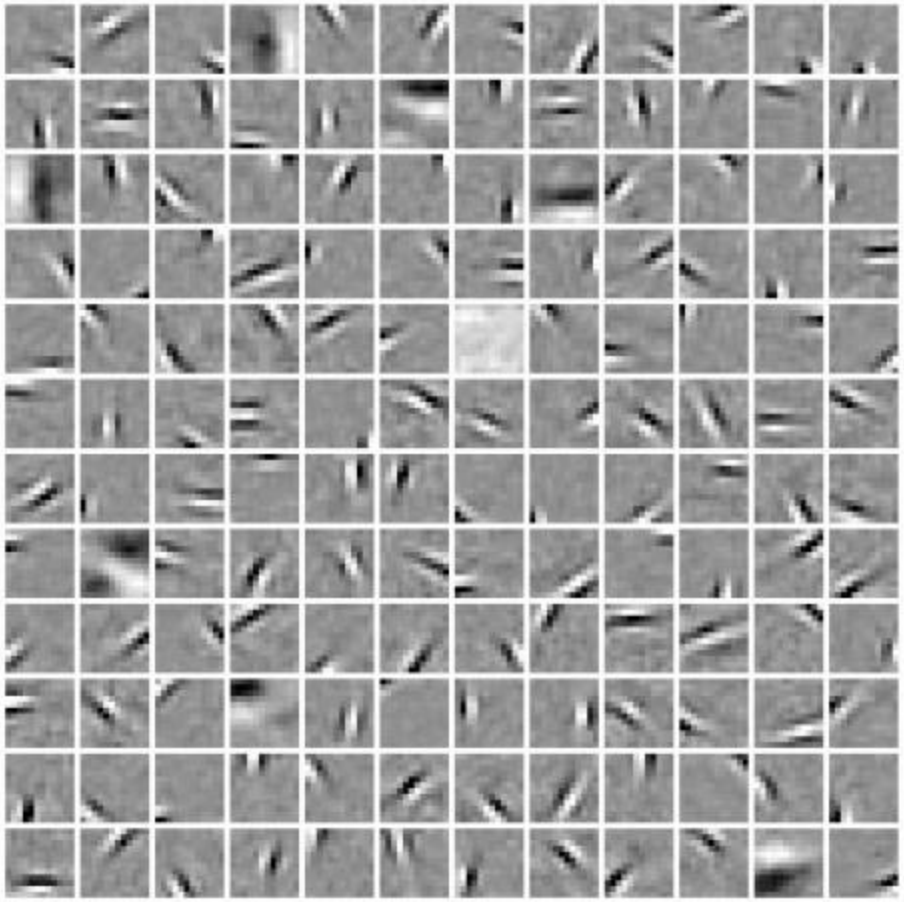
$$E(\mathbf{A}, \{\mathbf{s_k}\}) = \sum_{k=1}^{N} (\lambda \left\| \mathbf{s_k} \right\|_1 + \left\| \mathbf{x_k} - \mathbf{A}\mathbf{s_k} \right\|^2)$$

Gradient descent

$$\mathbf{A}^{(t+1)} = \mathbf{A}^{(t)} - \eta \sum_{k=1}^{N} (\mathbf{A}^{(t)}\mathbf{s_k} - \mathbf{x_k})\mathbf{s_k}^t$$

- Fix the basis ***A*** and obtain the sparse coefficients for each signal using MP, OMP or BP (some papers – like the one by Olshausen and Field - use gradient descent for this step!).
- Now fix the coefficients, and update the basis vectors (using various techniques, one of which was described on the previous slide).
- Normalize each basis vector to unit norm (so we may use *projected* gradient descent).
- Repeat the previous two steps until some error criterion is met.

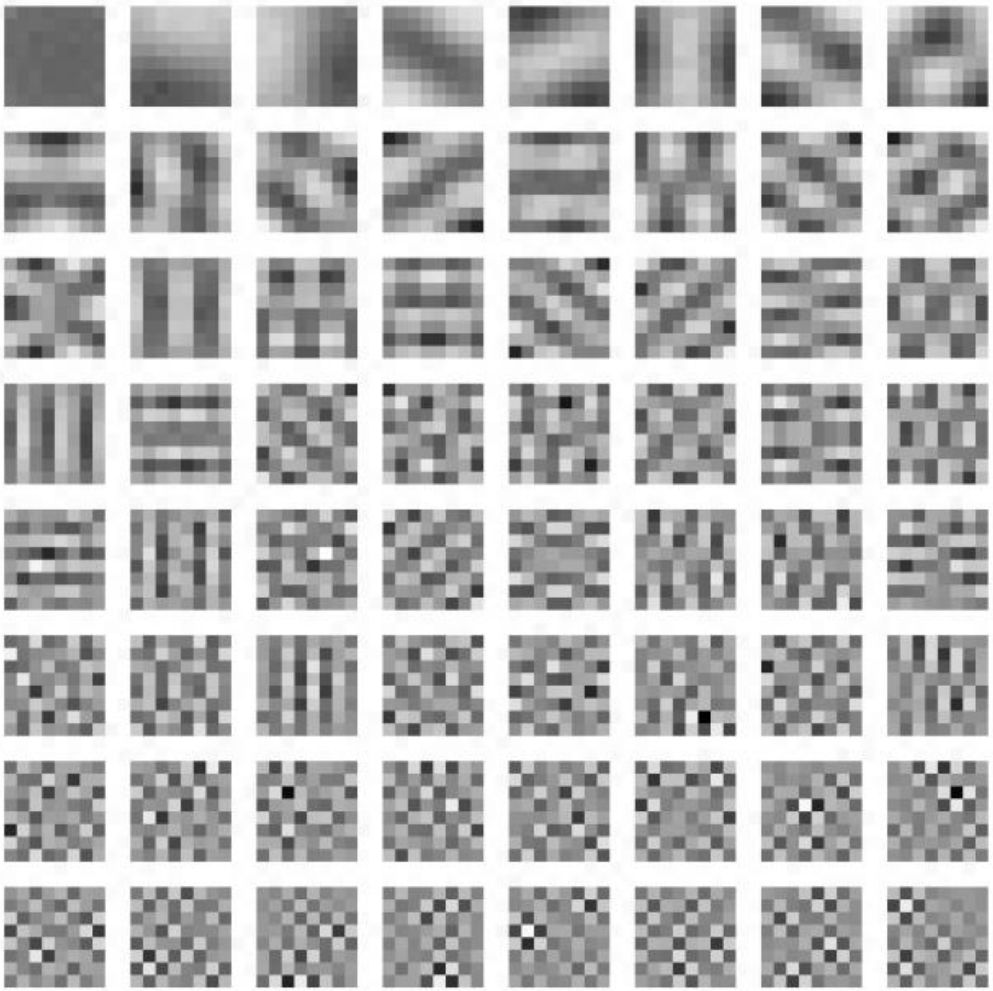
Toy Experiment 1



Result of basis learning (dictionary with 144 elements) with sparsity constraints on the codes. Training performed on 12 x 12 patches extracted from natural images.

It is conjectured that the simple cells of the V1 area of the visual cortex in the brain have responses similar in shape to these learned dictionary elements.

Ref: Olshausen and Field, “Natural image statistics and efficient coding”
https://pdfs.semanticscholar.org/e309/e441a38ccee6456bd02e0f1e894e44180d53.pdf



These are the results one obtains upon performing PCA on the 12 x 12 patches.

Ref: Olshausen and Field, “Natural image statistics and efficient coding”
https://pdfs.semanticscholar.org/e309/e441a38ccee6456bd02e0f1e894e44180d53.pdf

Toy Experiment 2

- Data-points generated as a (Laplacian/super-Laplacian) random linear combination of some arbitrarily chosen basis vectors. Note: the coefficients are chosen from a (super)-Laplacian distribution.
- In the no-noise situation, the aim is to extract the basis vectors and the coefficients of the linear combination.
- The fitting results are shown on the next slide. The true and estimated directions agree quite well.

Ref: Lewicki and Sejnowski, “Learning overcomplete representations”

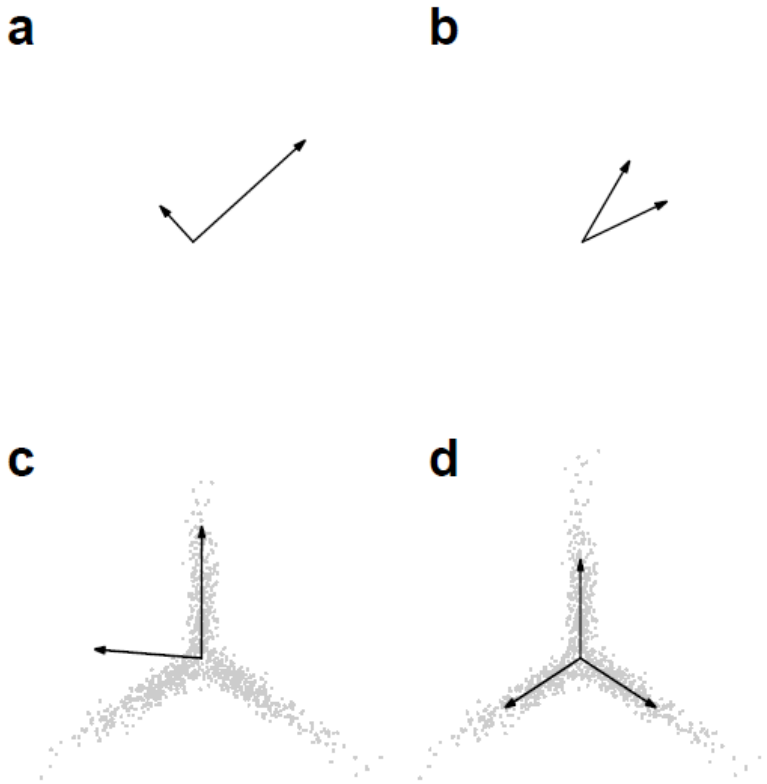
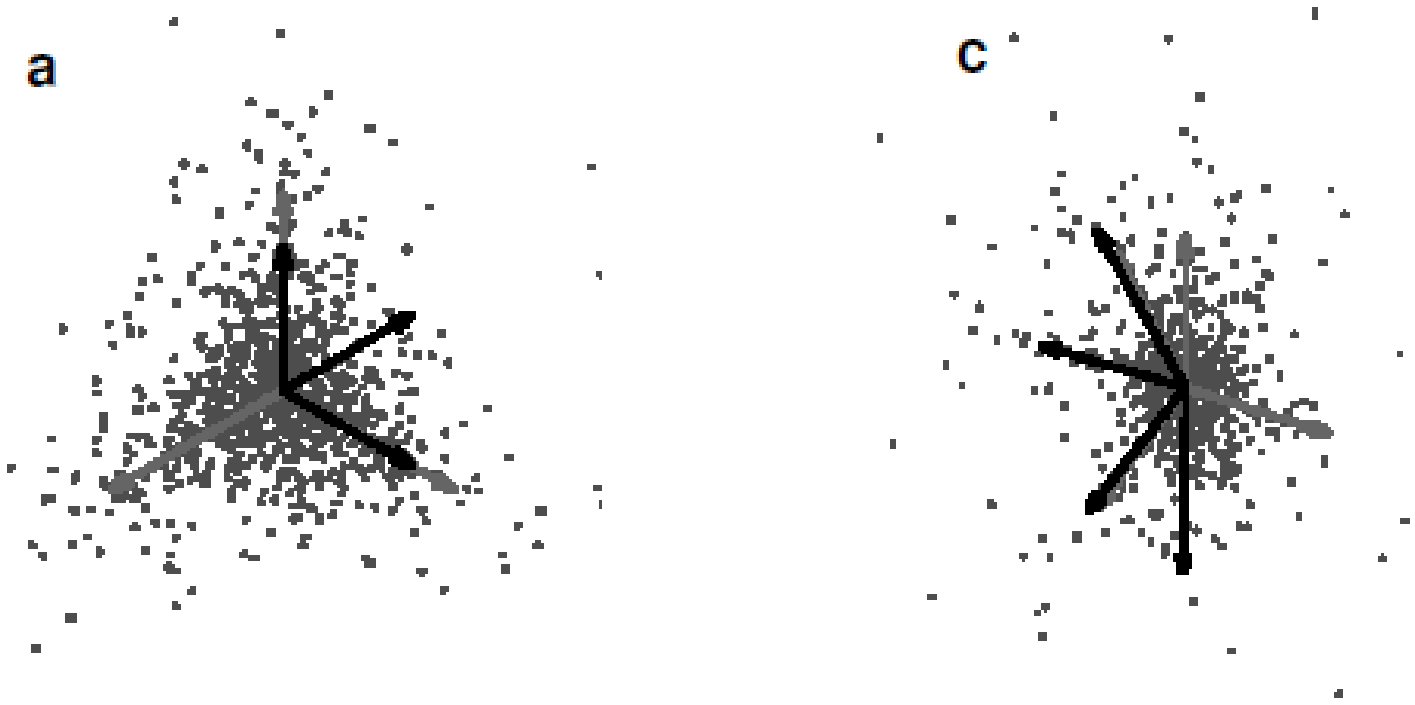


Figure 2: Fitting two-dimensional data with different bases. (**a**) PCA makes the assumption that the data have a Gaussian distribution. The optimal basis vectors are orthogonal and are not efficient at representing non-orthogonal density distributions. (**b**) ICA does not require that the vectors be orthogonal and can fit more general types of densities. (**c**) Some data distributions, however, cannot be modeled adequately by either PCA or ICA. (**d**) Allowing the basis to be overcomplete allows the three-armed distribution to be fit with simple and independent distributions for the basis vector coefficients.

Ref: Lewicki and Sejnowski, "Learning overcomplete representations"



Ref: Lewicki and Sejnowski, "Learning overcomplete representations"

Learning the Bases: Method of Optimal Directions (MOD) - Method 2

- Given a fixed dictionary **A**, assume sparse codes for every signal are computed using OMP, MP etc.
- The overall error for dictionary update is now given as

$$E(\mathbf{A}) = \sum_{i=1}^{n} \left\| \mathbf{y_i} - \mathbf{A}\mathbf{s_i} \right\|^2 = \left\| \mathbf{Y} - \mathbf{A}\mathbf{S} \right\|_F^2$$

- We want to find dictionary **A** that minimizes this error (assuming fixed sparse codes).

Ref: Engan et al, "Method of optimal directions for frame design"

Learning the Bases: Method of Optimal Directions (MOD) - Method 2

- Take the derivative of E(**A**) w.r.t. **A** and set it to **0**. This gives us the following update:

$$(\mathbf{Y} - \mathbf{AS})\mathbf{S}^T = \mathbf{0} \Rightarrow \mathbf{A}^{(t+1)} = \mathbf{Y}\mathbf{S}^{(t)^T}(\mathbf{S}^{(t)}\mathbf{S}^{(t)^T})^{-1}$$

- Following the update of **A**, each column in **A** is independently rescaled to unit norm.
- The updates of **A** and **S** alternate with each other till some convergence criterion is reached.
- This method is more efficient than the one by Olshausen and Field.

Learning the Bases: Method 3- Union of Orthonormal Bases

- Like before, we represent a signal in the following way:

$$\mathbf{X} = \mathbf{AS} + \varepsilon$$

$$(\mathbf{A}, \mathbf{S}) = \min_{\mathbf{A},\mathbf{S}} \|\mathbf{X} - \mathbf{AS}\|^2 + \lambda \|\mathbf{S}\|_1$$

- **A** is an over-complete dictionary, but let us assume that it is a union of ortho-normal bases, in the form

$$\mathbf{A} = [\mathbf{A_1} \mid \mathbf{A_2} \mid \ldots \mid \mathbf{A_M}]$$

$$\forall i, 1 \le i \le M, \mathbf{A_i} \mathbf{A_i^T} = \mathbf{I}$$

Learning the Bases: Method 3- Union of Ortho-normal Bases

- The coefficient matrix **S** can now be written as follows (M subsets, each corresponding to a single orthonormal basis):

$$\mathbf{S} = \begin{pmatrix} \mathbf{S_1} \\ \mathbf{S_2} \\ . \\ . \\ \mathbf{S_M} \end{pmatrix}$$

- Assuming we have fixed bases stored in **A**, the coefficients in **S** can be estimated using block coordinate descent, described on the following slide.

Learning the Bases: Method 3- Union of Ortho-normal Bases

for $t = 1:T\{$

$\lambda_t = \lambda_0(1 - t/T);$

Update $\mathbf{S}$ using BCR with parameter λ_t

$\}$

$BCR(\lambda_t, \mathbf{S})\{$

for $m = 1:M\{$

$$\mathbf{X_m} = \mathbf{X} - \sum_{j \neq m} \mathbf{A_j S_j}$$

Update $\mathbf{S_m}$ as follows : $\mathbf{S_m} = \arg\min_{\mathbf{S*}} \left\| \mathbf{X_m} - \mathbf{A_m S*} \right\|^2 + \lambda_t \left\| \mathbf{S*} \right\|_1$

$\}\}$

There is a quick way of performing this optimization given an ortho-normal basis – SOFT THRESHOLDING (could be replaced by hard thresholding if you had a stronger sparseness prior than an L1 norm

Learning the Bases: Method 3- Union of Ortho-normal Bases

- Given the coefficients, we now want to update the dictionary which is done as follows:

$$\text{for } m = 1 : M$$
$$\{$$
$$\mathbf{X_m} = \mathbf{X} - \sum_{j \neq m} \mathbf{A_j S_j};$$
$$\mathbf{S_m X_m^T} = \mathbf{U \Lambda V}^T;$$
$$\mathbf{A_m} = \mathbf{V U}^T;$$
$$\}$$

Why are we doing this? It is related to the so-called orthogonal Procrustes problem - a well-known application of SVD. We will see this on the next slide.

The specific problem we are solving is given below. Note that it cannot be solved using a pseudo-inverse as that will not impose orthonormality constraint, if there is noise in the data or if the coefficients are perturbed or thresholded.

$$\min_{\mathbf{A}} \left\| \mathbf{X_m} - \mathbf{A S_m} \right\|^2 \text{s.t. } \mathbf{A A}^T = \mathbf{I}$$

$$A^* = \min_A \|X - AS\|_F^2 \; s.t. \; A^T A = I$$

$$\min_A \|X - AS\|_F^2 = \min_A trace((X - AS)^T (X - AS))$$

$$= \min_A trace(X^T X - 2X^T AS + S^T S)$$

$$= \max_A trace(X^T AS)$$

$$= \max_A trace(ASX^T) -- \because trace(FG) = trace(GF)$$

--Let $SX^T = Q$, SVD of Q gives $Q = UDV^T$ --

$$= \max_A trace(AUDV^T)$$

$$= \max_A trace(V^T AUD)$$

$$= \max_A trace(Z(A)D) \text{ where } Z(A) = V^T AU$$

$$= \max_A \sum_i z_{ii} d_{ii} \le \sum_i d_{ii} \; (\because Z(A)^T Z(A) = I)$$

The maximum is achieved for $Z(A) = I$, i.e.

$$V^T AU = I \Rightarrow A = VU^T$$

Learning the Bases: Method 3- Union of Ortho-normal Bases

- Keeping all bases in ***A*** fixed, update the coefficients in ***S*** using a known sparse coding technique.
- Keeping the coefficients in ***S*** fixed, update the bases in ***A*** using the aforementioned SVD-based method.
- Repeat the above two steps until a convergence criterion is reached.

Learning the Bases: Method 4 – K-SVD

- Recall: we want to learn a dictionary and ***sparse*** codes on that dictionary given some data-points:

$$\min_{A,S} \|Y - AS\|^2 \text{ subject to } \forall i, \|s_i\|_0 \leq T_0$$

- Starting with a fixed dictionary, sparse coding follows as usual – OMP, BP, MP etc. The criterion could be based on reconstruction error or L0-norm of the sparse codes.
- The dictionary is updated one column at a time.

$$\|Y - AS\|_F^2 = \left\|Y - \sum_{j=1}^{K} a_j s^j\right\|_F^2 = \left\|Y - \sum_{j\neq k} a_j s^j - a_k s^k\right\|_F^2$$

$$= \|E_k - a_k s^k\|_F^2$$

Does NOT depend on the k-th dictionary column

Row 'k' (NOT column) of matrix S

How to find $\boldsymbol{a_k}$, given the above expression? We have decomposed the original error matrix, i.e. ***Y-AS***, into a sum of rank-1 matrices, out of which only the last term depends on $\boldsymbol{a_k}$. So we are trying to find a rank-1 approximation for $\boldsymbol{E_k}$, and this can be done by computing the SVD of $\boldsymbol{E_k}$, and using the singular vectors corresponding to the largest singular value.

$$E_k = U\Lambda V^T, a_k = u_1, s^k = \Lambda(1,1)v_1^T$$

Problem! The dictionary codes may no more be sparse! SVD does not have any in-built sparsity constraint in it! So, we proceed as follows:

$$\omega_k = \{i \mid s^k(i) \neq 0\}$$

Considers only those columns of Y (i.e. only those data-points) that actually **USE** the k-th dictionary atom, effectively yielding a smaller matrix, of size p by $|\omega_k|$

Consider the error matrix defined as follows:

$$\left\| Y_{\omega_k} - AS_{\omega_k} \right\|_F^2 = \left\| \left(Y - \sum_{j \neq k} a_j s^j \right)_{\omega_k} - a_k s^k_{\omega_k} \right\|_F^2 = \left\| E_{k,\omega_k} - (a_k s^k_{\omega_k}) \right\|_F^2$$

$$E_{k,\omega_k} = U \Lambda V^T$$

$$a_k \leftarrow u_1 \,,\, s^k_{\omega_k} \leftarrow \Lambda(1,1) v_1^T$$

This update affects the sparse codes of only those data-points that used the k-th dictionary element, i.e. those that are part of ω_k.

Task: Find the best dictionary to represent the data samples $\{\mathbf{y}_i\}_{i=1}^N$ as sparse compositions, by solving

$$\min_{\mathbf{D},\mathbf{X}} \{\|\mathbf{Y}-\mathbf{D}\mathbf{X}\|_F^2\} \quad \text{subject to} \quad \forall i,\ \|\mathbf{x}_i\|_0 \le T_0.$$

Initialization : Set the dictionary matrix $\mathbf{D}^{(0)} \in \mathbb{R}^{n\times K}$ with ℓ^2 normalized columns. Set $J=1$.
Repeat until convergence (stopping rule):

- *Sparse Coding Stage*: Use any pursuit algorithm to compute the representation vectors $\mathbf{x}_i$ for each example $\mathbf{y}_i$, by approximating the solution of

$$i=1,2,\ldots,\ N, \quad \min_{\mathbf{x}_i} \{\|\mathbf{y}_i-\mathbf{D}\mathbf{x}_i\|_2^2\} \quad \text{subject to} \quad \|\mathbf{x}_i\|_0 \le T_0.$$

- *Codebook Update Stage*: For each column $k=1,2,\ldots,K$ in $\mathbf{D}^{(J-1)}$, update it by
  - Define the group of examples that use this atom, $\omega_k = \{i|\ 1\le i\le N,\ \mathbf{x}_T^k(i)\neq 0\}$.
  - Compute the overall representation error matrix, $\mathbf{E}_k$, by

$$\mathbf{E}_k = \mathbf{Y} - \sum_{j\neq k} \mathbf{d}_j \mathbf{x}_T^j.$$

  - Restrict $\mathbf{E}_k$ by choosing only the columns corresponding to ω_k, and obtain $\mathbf{E}_k^R$.
  - Apply SVD decomposition $\mathbf{E}_k^R = \mathbf{U}\mathbf{\Delta}\mathbf{V}^T$. Choose the updated dictionary column $\tilde{\mathbf{d}}_k$ to be the first column of $\mathbf{U}$. Update the coefficient vector $\mathbf{x}_R^k$ to be the first column of $\mathbf{V}$ multiplied by $\mathbf{\Delta}(1,1)$.
- Set $J=J+1$.

http://www.cs.technion.ac.il/~elad/publications/journals/2004/32_KSVD_IEEE_TSP.pdf

KSVD and K-means

- Limit the sparsity factor $T_0 = 1$.
- Enforce all the sparse codes to be either 0 or 1.
- Then you get the K-means algorithm! ☺

Implementation issues

- KSVD is a popular and effective method. But some implementation issues haunt K-SVD (life is never easy ☺).

- KSVD is susceptible to **local minima and over-fitting** if K is too large (just like you can get meaningless clusters if your number of cluster is too small, or get meaningless densities if the number of histogram bins is too many).

- KSVD convergence is not fully guaranteed. The dictionary updates given fixed sparse codes ensure the error decreases. However the sparse codes given fixed dictionary may not decrease the error – it is affected by the behaviour of the sparse coding approximation algorithms.
- You can speed up the algorithm by removing extremely "unpopular" dictionary elements, or removing duplicate (or near-duplicate) columns of the dictionary.

(Many) Applications of KSVD

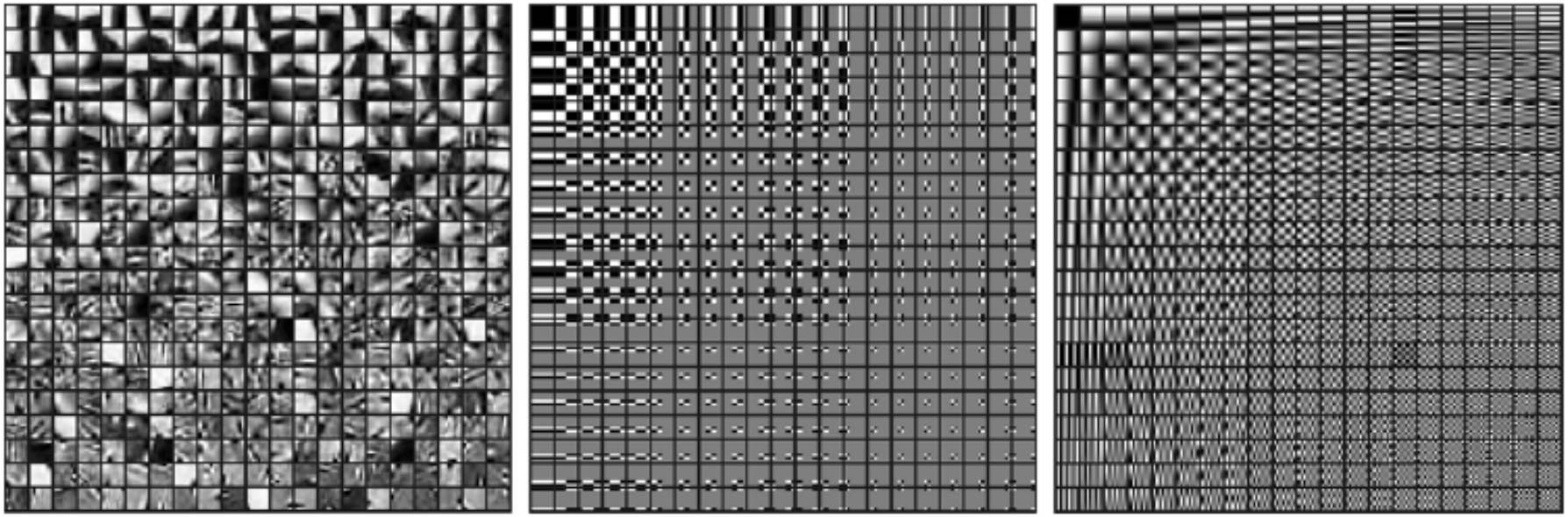
- Image denoising
- Image inpainting
- Image deblurring
- Blind compressive sensing
- Classification
- Compression
- And **you** can work on many more ☺

Application: Image Compression

- Recall JPEG algorithm
- It divides the image to be compressed into non-overlapping patches of size 8 x 8.
- It computes the 2D-DCT coefficients of each patch, quantizes them and stores only the non-zero coefficients – using Huffman encoding.
- KSVD can be used in image compression by replacing the orthonormal 2D-DCT dictionary by an overcomplete dictionary trained on a database of similar images.

Application: Image Compression (Training Phase)

- Training set for dictionary learning: a set of 11000 patches of size 8 x 8 – taken from a face image database. Dictionary size K = 441 atoms (elements).
- OMP used in the sparse coding step during training – stopping criterion is a fixed number of coefficients $T_0 = 10$.
- Over-complete Haar and DCT dictionaries – of size 64 x 441 – and ortho-normal DCT basis of size 64 x 64 (JPEG), also used for comparison.



(a) (b) (c)

Fig. 5. (a) The learned dictionary. Its elements are sorted in an ascending order of their variance and stretched to maximal range for display purposes. (b) The overcomplete separable Haar dictionary and (c) the overcomplete DCT dictionary are used for comparison.

http://www.cs.technion.ac.il/~elad/publications/journals/2004/32_KSVD_IEEE_TSP.pdf

Application: Image Compression (Testing Phase)

- A lossy image compression algorithm is evaluated using an ROC curve – the X axis contains the average number of bits (BPP) to store the signal.
- The Y-axis is the compression error or PSNR. Normally, the acceptable error *e* is fixed and the number of bits is calculated.
- The test image is divided into non-overlapping patches of size 8 x 8.
- Each patch is projected onto the trained dictionary and its sparse code is obtained using OMP given a fixed error *e*.

Application: Image Compression (Testing Phase)

- The encoded image contains the following:

1. Sparse-code coefficients for each patch and the indices of each non-zero coefficient (in the dictionary).
2. The number of coefficients used to represent each patch (different patches will need different number of coefficients).

Application: Image Compression (Testing Phase)

- The average number of bits per pixel (BPP) is calculated as:

$$BPP = \frac{a \bullet \#\ patches + \#\ coeffs \bullet (b + Q)}{\#\ pixels}$$

Sum total of the number of coefficients representing each patch

Number of bits required to store the number of coefficients for each patch

Number of bits required to store the dictionary index for each coefficient

Number of bits required to code each coefficient (quantization level). **The quantization makes this a lossy compression algorithm.**

Huffman encoding

http://www.cs.technion.ac.il/~elad/publications/journals/2004/32_KSVD_IEEE_TSP.pdf

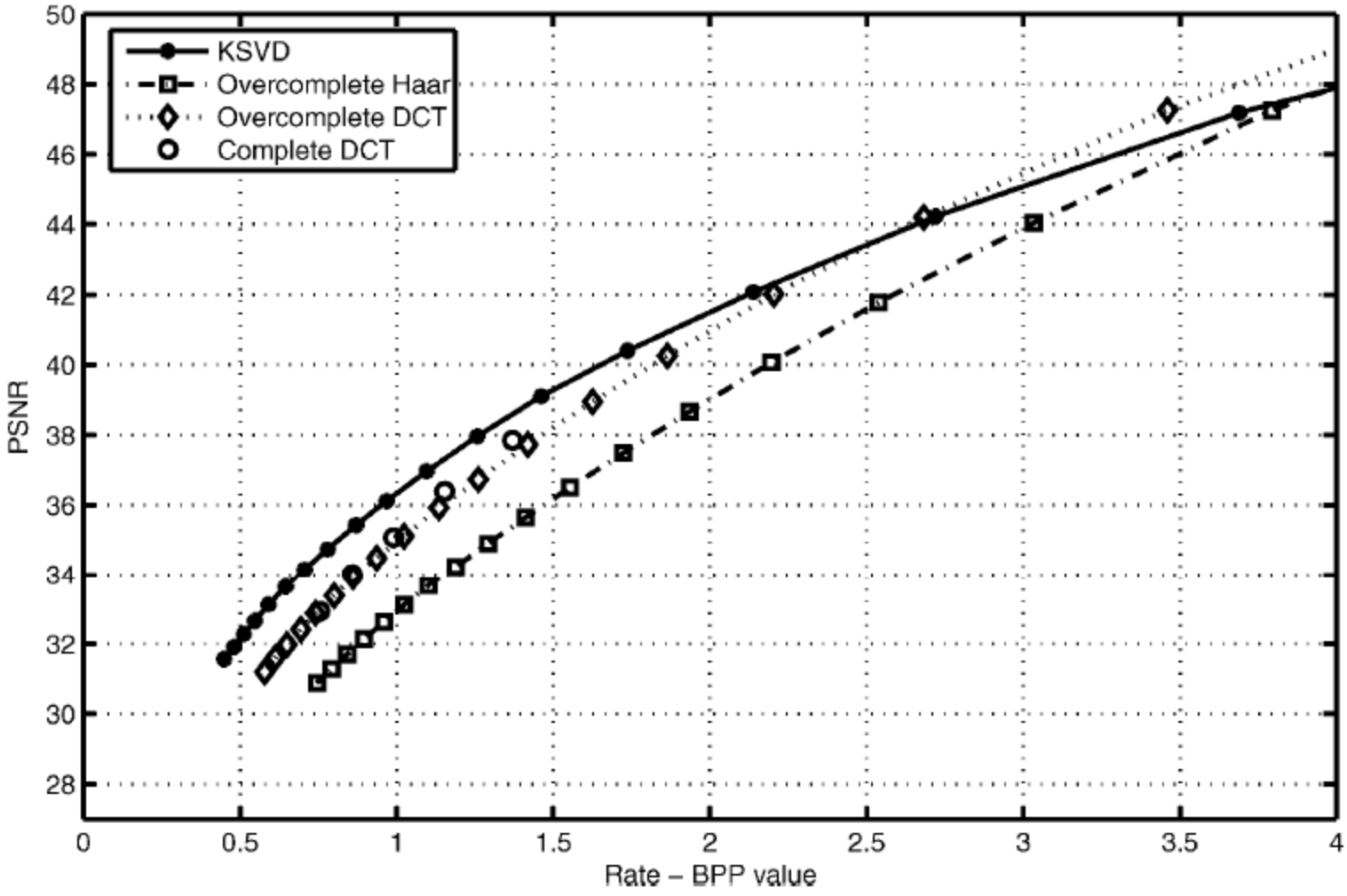


Fig. 8. Compression results: rate-distortion graphs.

Even if the error ‘e’ for the OMP was fixed, we need to compute the total MSE between the true and the compressed image. This is due to effects of quantization while storing the sparse coefficient values for each patch.

http://www.cs.technion.ac.il/~elad/publications/journals/2004/32_KSVD_IEEE_TSP.pdf

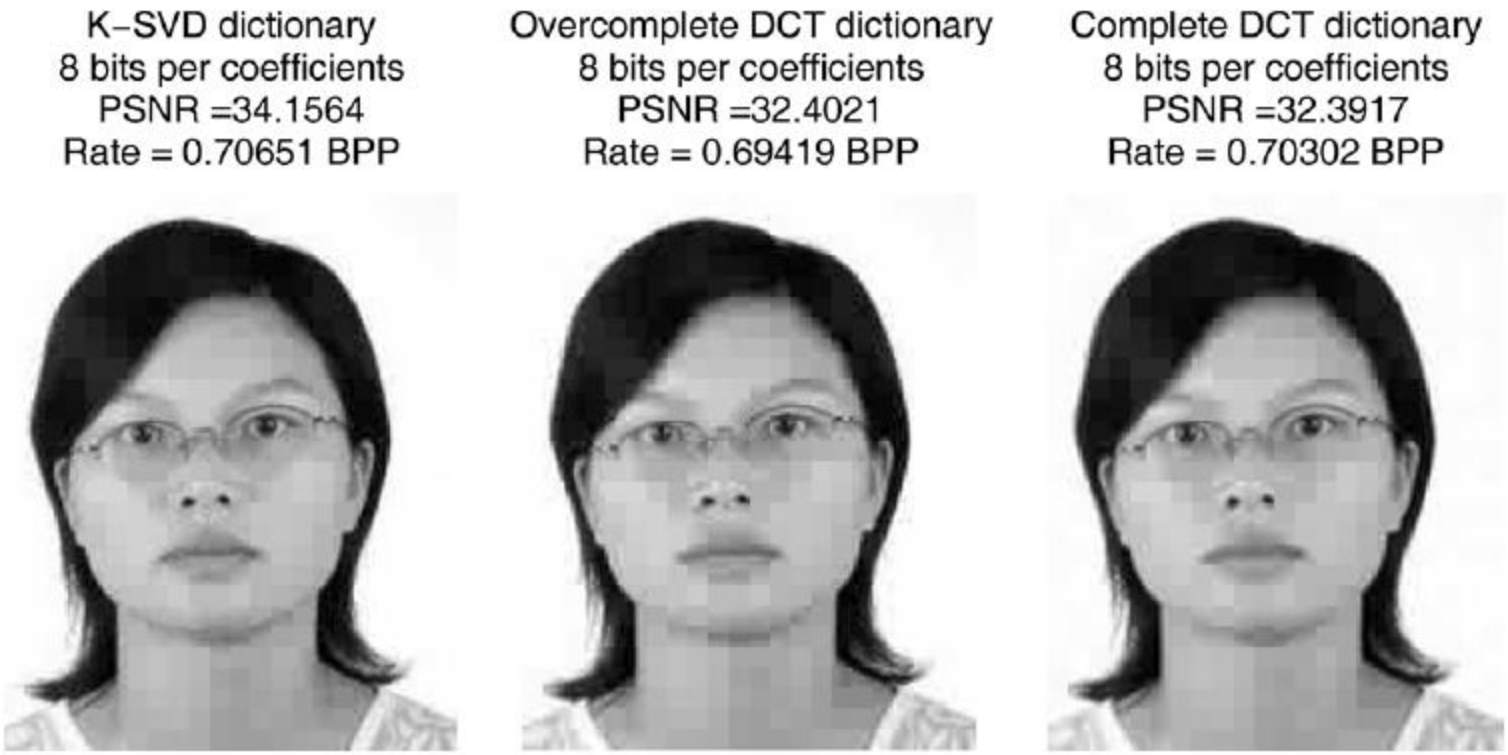


Fig. 9. Sample compression results.

Application: Image Denoising

- KSVD for denoising seeks to minimize the following objective function:

$$(X, D, \{\alpha_{ij}\}) = \arg\min_{X, D, \{\alpha_{ij}\}} \lambda \|X - Y\|^2 + \sum_{i,j} \mu_{ij} \|\alpha_{ij}\|_0 + \sum_{i,j} \|D\alpha_{ij} - R_{ij} X\|^2$$

Y = noisy image

X = underlying clean image (to be estimated)

α_{ij} = sparse dictionary coefficients for patch at location (i,j)

D = dictionary

R_{ij} = matrix that extracts the patch x_{ij} from image X, i.e.

$x_{ij} = R_{ij} X$

Application: Image Denoising

- Note: The dictionary may be learned a priori from a corpus of image patches. The patches from the noisy image can then be denoised by mere sparse coding.

- The more preferable method is to train the dictionary directly on the noisy image in tandem with the sparse coding step (as in the previous slide).

- This avoids having to depend on the training set and allows for tuning of the dictionary to the underlying image structure (as opposed to the structure of some other images).

KSVD Algorithm for Denoising (Dictionary learned on the noisy image)

- Set X = Y, D = overcomplete DCT
- Until some "convergence" criterion is satisfied, repeat the following:

1. Obtain the sparse codes for every patch (typically using OMP) as follows:

$$\forall i,\ j, \min \left\| \alpha_{ij} \right\|_0 \text{ s.t. } \left\| R_{ij} X - D\alpha_{ij} \right\|^2 \leq C\sigma^2$$

2. Perform the dictionary learning update typical for KSVD.

- Estimate the final image X by averaging the reconstructed overlapping patches, OR estimate X given D and α_{ij}:

$$X = \arg\min_X \lambda \left\| X - Y \right\|^2 + \sum_{ij} \left\| D\alpha_{ij} - R_{ij} X \right\|^2$$

$$\Rightarrow X = (\lambda I + \sum_{ij} R_{ij}^T R_{ij})^{-1} (\lambda Y + \sum_{ij} R_{ij}^T D\alpha_{ij})$$

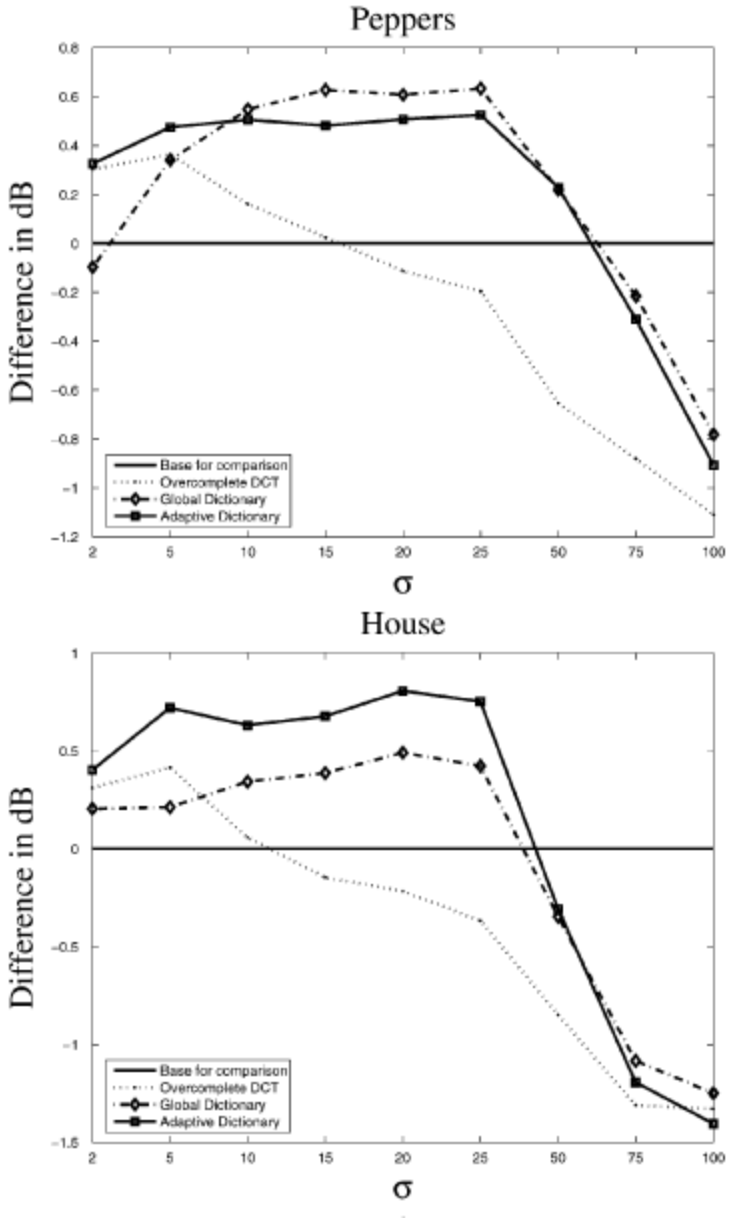
KSVD Algorithm for Denoising (Dictionary learned on the noisy image)

$$X = \arg\min_X \lambda \|X - Y\|^2 + \sum_{ij} \|D\alpha_{ij} - R_{ij}X\|^2$$

$$\Rightarrow X = (\lambda I + \sum_{ij} R_{ij}^T R_{ij})^{-1} (\lambda Y + \sum_{ij} R_{ij}^T D\alpha_{ij})$$

$$\lambda \text{ set to } \frac{30}{\sigma}$$

This equation is a mathematically rigorous way to show how X was reconstructed by averaging overlapping denoised patches together with the noisy image as well.



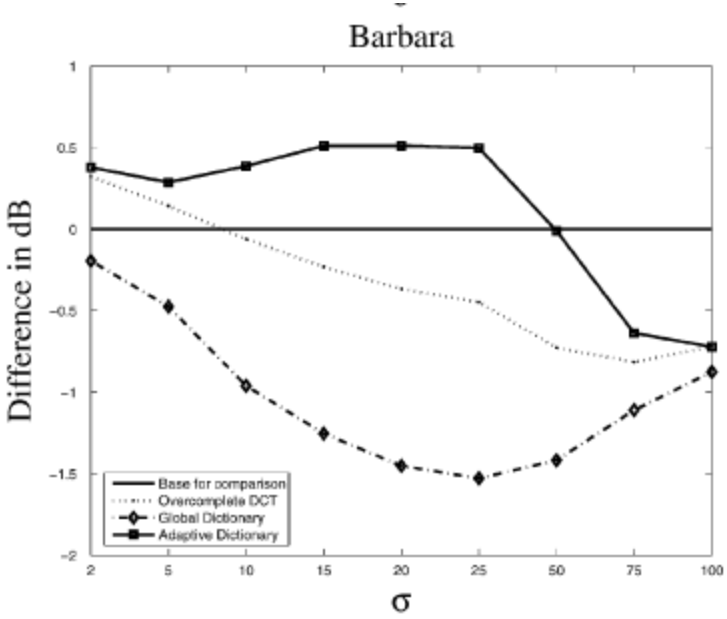


Fig. 4. Comparison between the three presented methods (overcomplete DCT, global trained dictionary, and adaptive dictionary trained on patches from the noisy image) and the results achieved recently in [23] for three test images.

Baseline for comparison: method by Portilla et al, "Image denoising using scale mixture of Gaussians in the wavelet domain", IEEE TIP 2003.

http://www.cs.technion.ac.il/~elad/publications/journals/2005/KSVD_Denoising_IEEE_TIP.pdf

http://www.cs.technion.ac.il/~elad/publications/journals/2005/KSVD_Denoising_IEEE_TIP.pdf

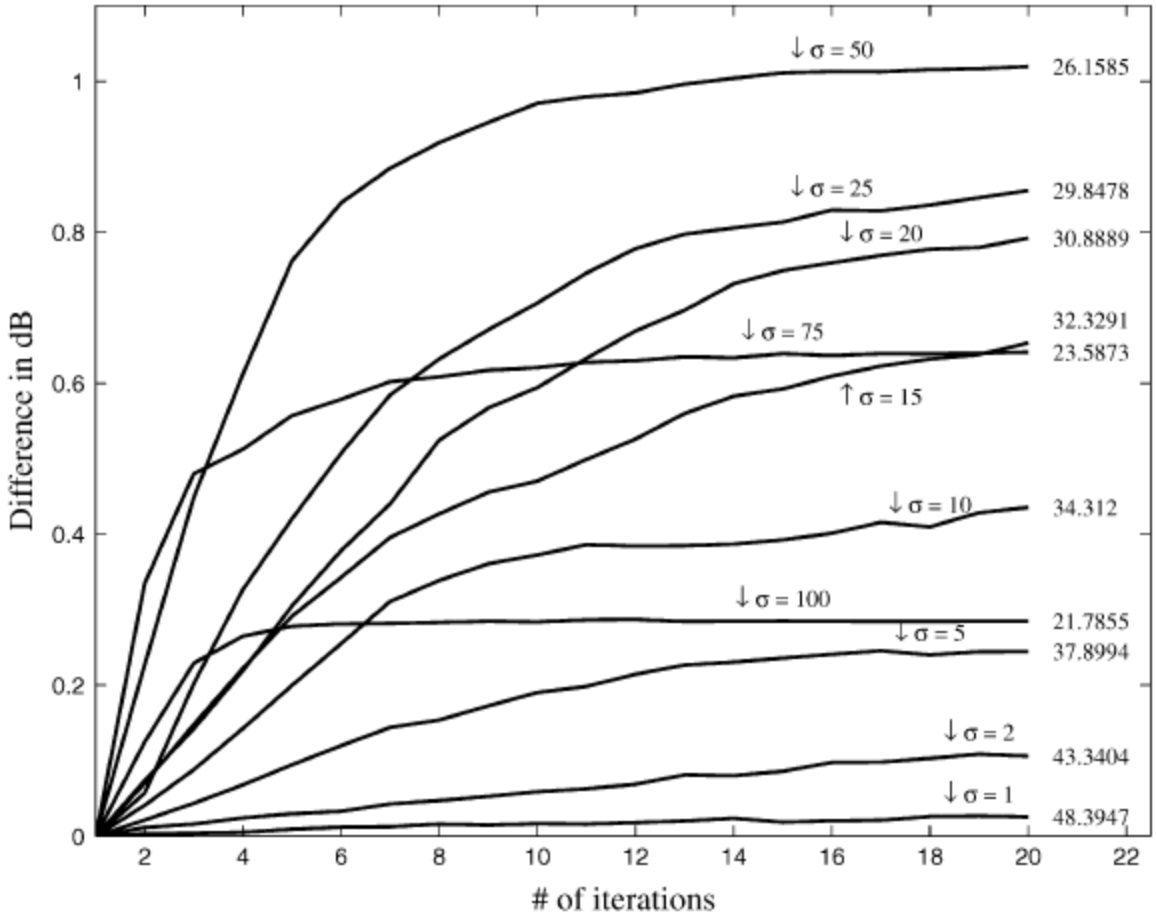


Fig. 5. Improvement in the denoising results after each iteration of the K-SVD algorithm, executed on noisy patches of the image "Peppers."

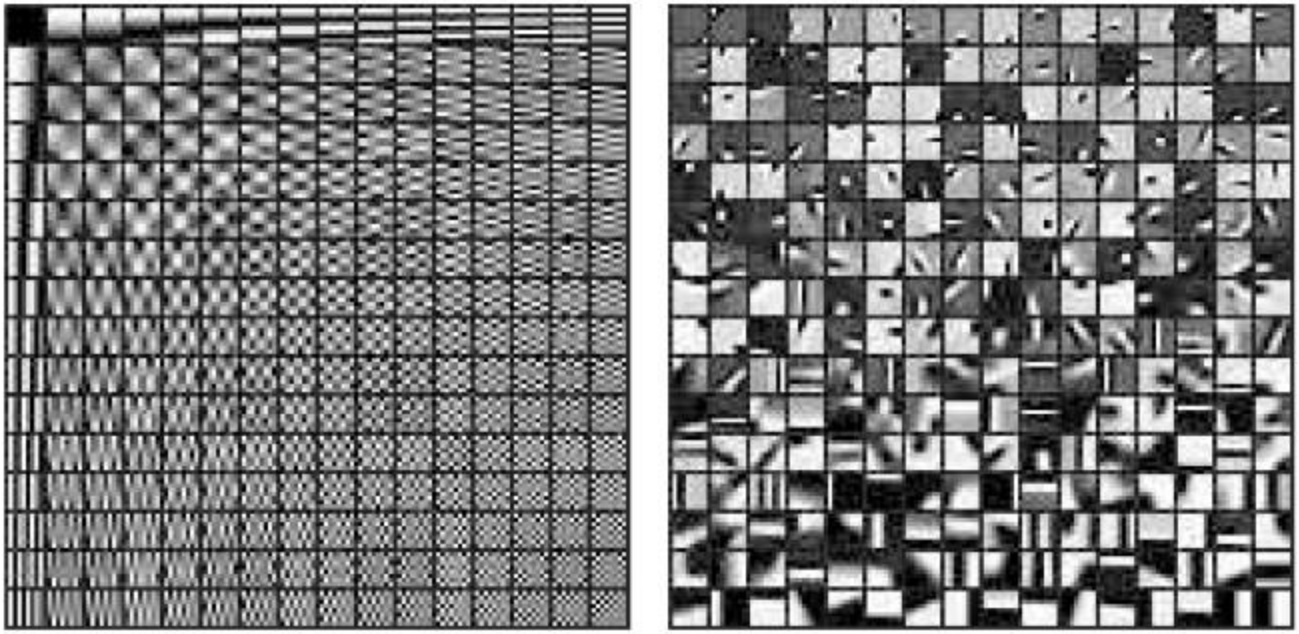


Fig. 2. Left: Overcomplete DCT dictionary. Right: Globally trained dictionary.

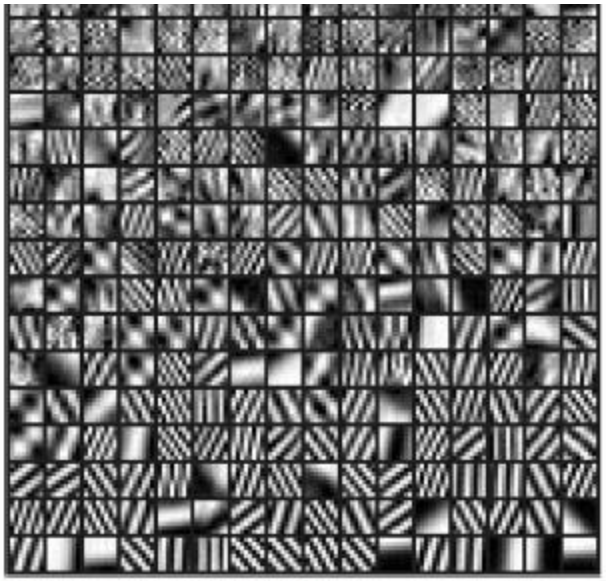


Fig. 7. Example of the denoising results for the image "Barbara" with σ = 20—the adaptively trained dictionary.

http://www.cs.technion.ac.il/~elad/Various/KSVD_Matlab_ToolBox.zip

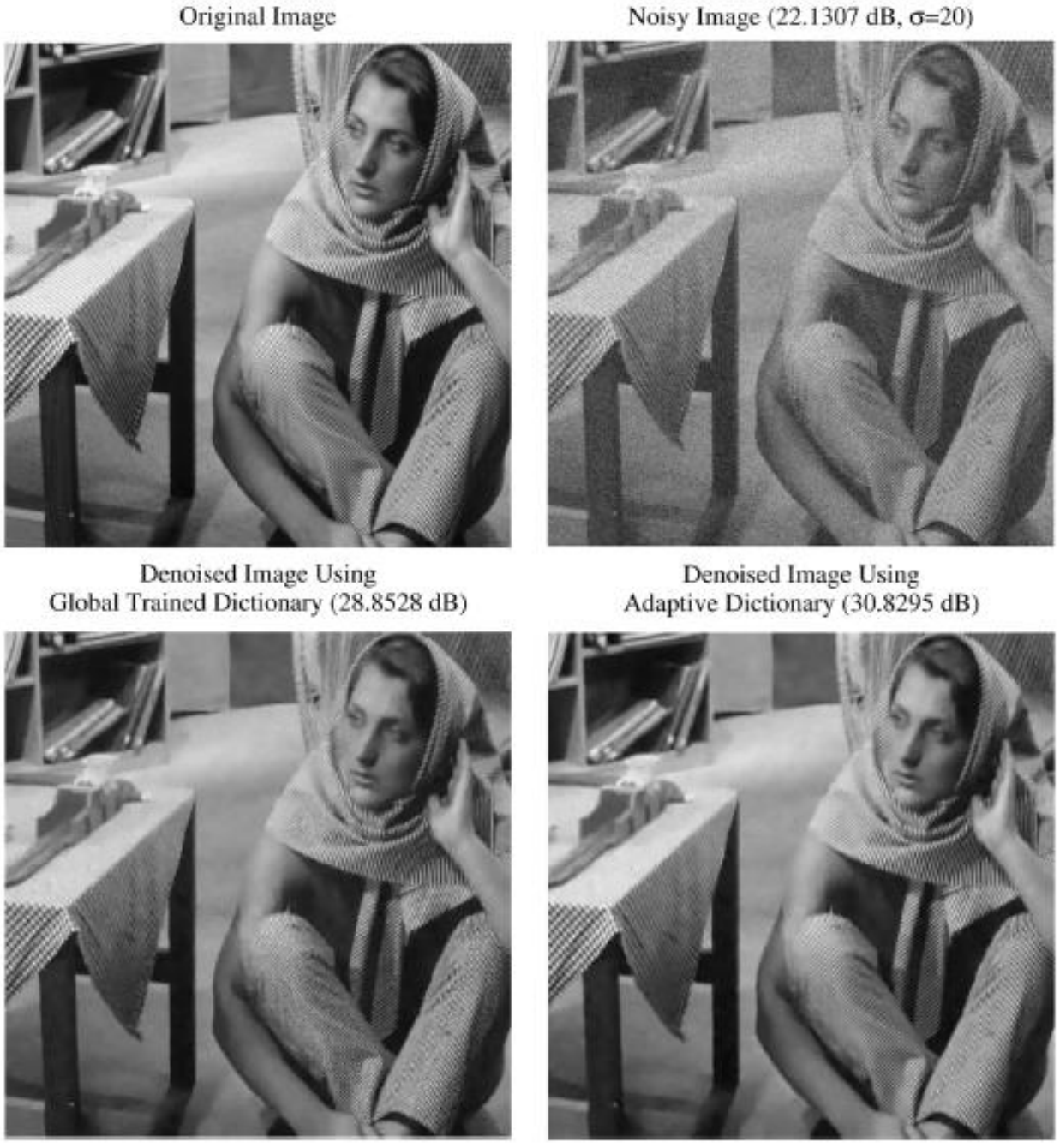


Fig. 6. Example of the denoising results for the image "Barbara" with $\sigma = 20$—the original, the noisy, and two restoration results.

http://www.cs.technion.ac.il/~elad/publications/journals/2005/KSVD_Denoising_IEEE_TIP.pdf

Application of KSVD: Filling in Missing Pixels

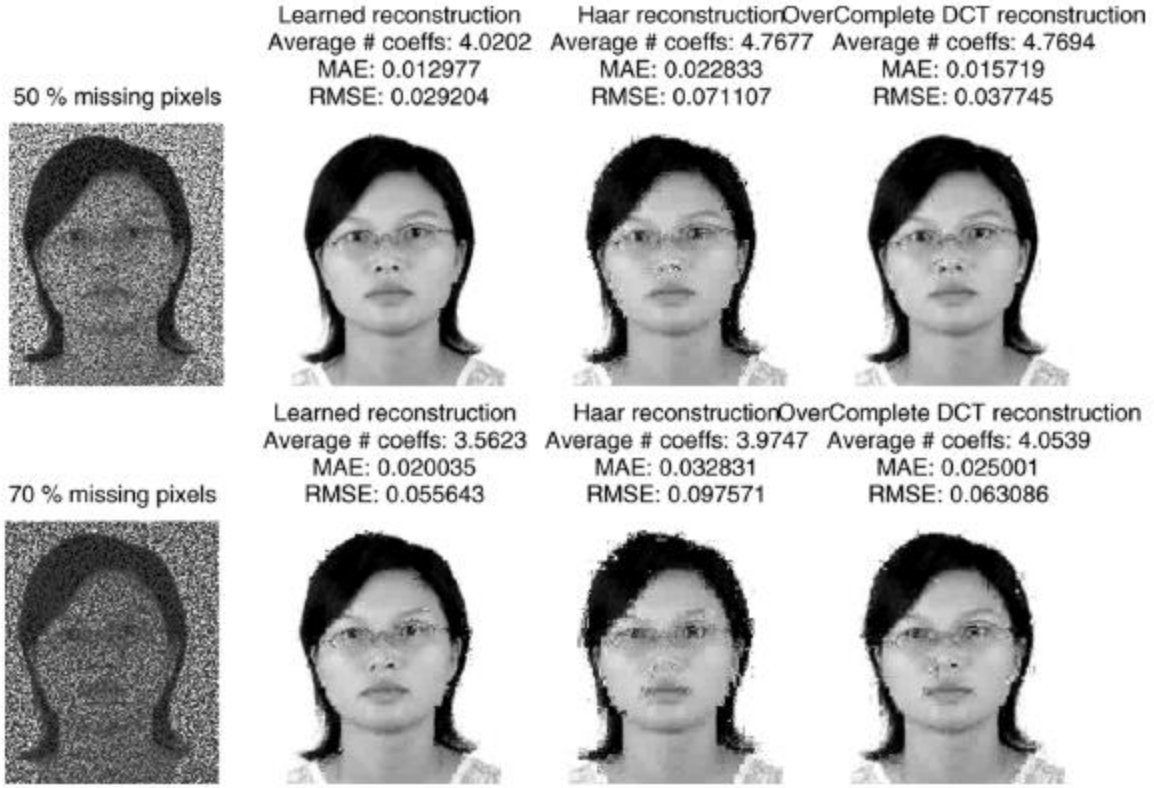


Fig. 7. The corrupted image (left) with the missing pixels marked as points and the reconstructed results by the learned dictionary, the overcomplete Haar dictionary, and the overcomplete DCT dictionary, respectively. The different rows are for 50% and 70% of missing pixels.

http://www.cs.technion.ac.il/~elad/publications/journals/2004/32_KSVD_IEEE_TSP.pdf

Application of KSVD: Filling in Missing Pixels

- An over-complete dictionary is trained a priori on a set of face images.
- A test face image (not part of the training set) is synthetically degraded by masking out 50 to 70 percent of the pixels.
- Patches from the degraded image are sparse coded by projection onto the trained dictionary using OMP.
- OMP is modified so that only the non-degraded pixels are considered during any error computation (the dictionary elements are therefore appropriately re-scaled to unit norm).

Application of KSVD: Filling in Missing Pixels

$y_{ij} = n^2 \times 1$ patch in corrupted image at location (i, j)

$\tilde{y}_{ij} = |P| \times 1$ vector containing the P non-corrupted intensities from y_{ij}

$$\therefore \tilde{y}_{ij} = \Phi_{ij} D s_{ij} \text{ where } \Phi_{ij} \in R^{P \times n^2}, D \in R^{n^2 \times K}, s_{ij} \in R^{K \times 1}$$

$$E(\{s_{ij}\}) = \sum_{i,j} \left\| \tilde{y}_{ij} - \Phi_{ij} D s_{ij} \right\|^2 \text{ s.t.} \left\| s_{ij} \right\|_0 \leq T_0$$

D learned a-priori

Sparse codes determined using OMP or ISTA

Application of KSVD: Filling in Missing Pixels

- The patches are reconstructed in the following manner:

$$y_{ij} = Ds_{ij}$$

NOTE: Dictionary elements without masking!!

KSVD Application: Coded Exposure Image

It is a coded superposition (i.e. summation) of *T* sub-frames within a unit integration time of the video camera.

Coded exposure image (captured in one unit integration time of the camera)

$$I(x,y)=\sum_{t=1}^{T} S(x,y,t)\bullet E(x,y,t)$$

Binary code at time instant *t*

Sub-frame at time instant *t*

Conventional capture (simple integration across time, without modulation by binary codes)

$$\forall x,y,t\ S(x,y,t)=1$$

KSVD Application: Sparse Coding in Video CS

$$I(x,y) = \sum_{t=1}^{T} S(x,y,t) \bullet E(x,y,t) \rightarrow \mathbf{y} = \mathbf{\Phi f}$$

$$\mathbf{f} = \mathbf{\Psi \theta}$$

$$\therefore \mathbf{y} = \mathbf{\Phi \Psi \theta}$$

y is a vectorized form of the snapshot image **I** of *n* pixels.
f is a vectorized form of the underlying video with *T* frames and each frame having *n* pixels, hence **f** has *nT* pixels in total.
$\mathbf{\Psi}$ is a 3D DCT basis – the basis can also be "learned offline on a representative set of videos"
$\mathbf{\Phi}$ Is a matrix of size *n* x *nT* containing values from the binary code **S**. See below:

$$\mathbf{\Phi} = (diag(S_1) \mid diag(S_2) \mid \ldots \mid diag(S_T))$$

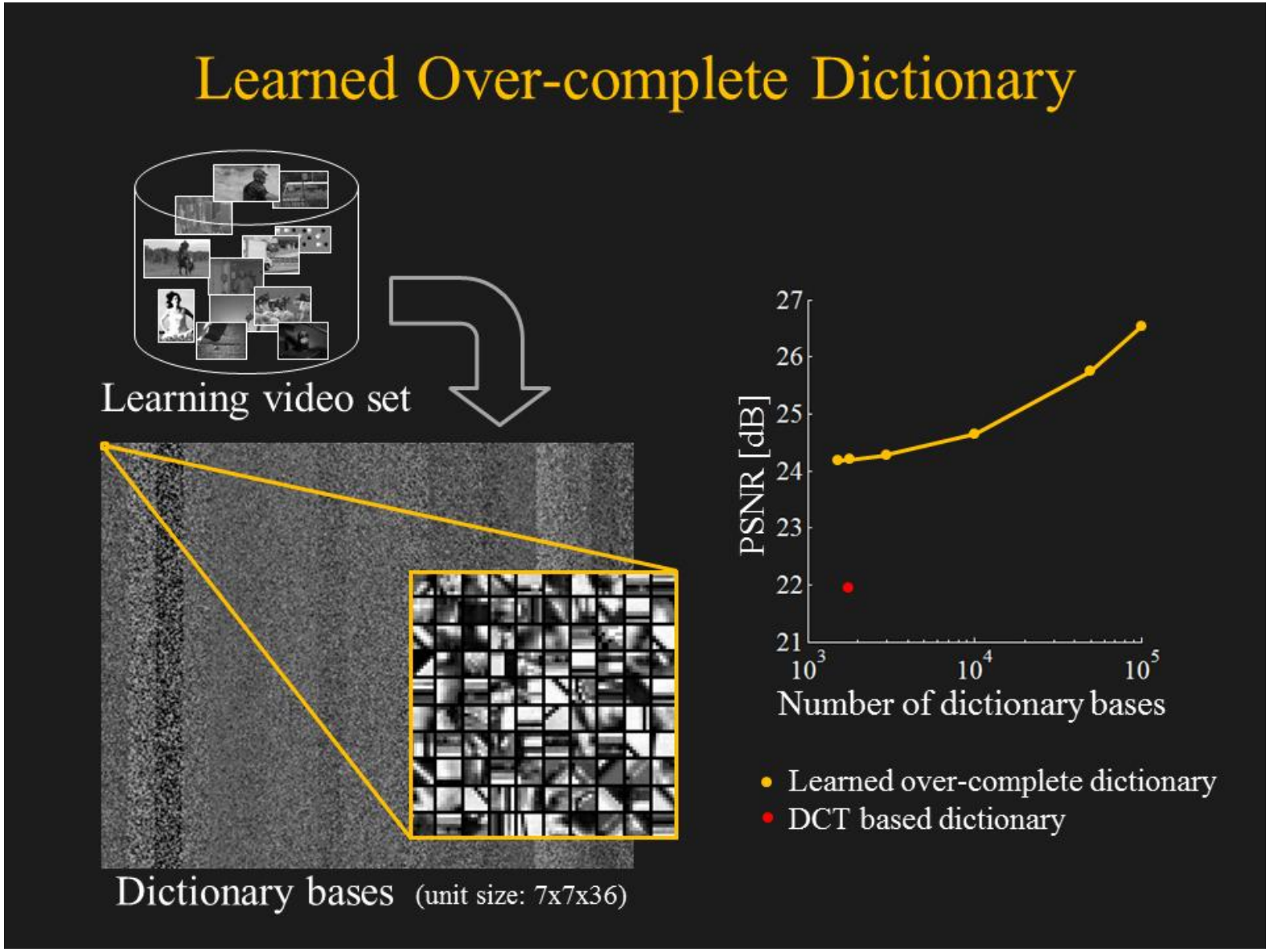
$$S_t(x,y) = S(x,y,t) \text{ for } t = 1 \text{ to } T$$

Diag(S_t) represents a *n* x *n* diagonal matrix. The diagonal contains elements from the mask S_t.

Dictionary Learning

- Done offline – training set was 20 video sequences, each video rotated in 8 directions and played forward + backward = 320 videos.
- All videos had **target frame rate** (2000 fps, as we work with a 60 fps camera and want 36 fold gain).
- Video-patch size was 7 x 7 x 36 = 1764 x 1
- Offline learning: KSVD (*), *K* = 100,000 atoms
- Sparse coding done online (using OMP)

KSVD is a dictionary learning technique. Given a set of input patches in n X N, it learns a set of vectors (n X K), such that a sparse linear combination of these vectors approximates each patch as closely as possible. There are K coefficients per patch, out of which most are encouraged to be 0 (sparse).



http://www.cs.columbia.edu/CAVE/projects/single_shot_video/

Video reconstruction results on real data available below:
https://www.youtube.com/watch?v=JAYC0C3NIdY

• Ideally, the videos whose patches are used for training must have the TARGET frame rate.
• Suppose we have a 60 fps video camera and want to increase its temporal resolution to say 60T fps. This would require coded versions of T consecutive sub-frames in the Hitomi camera (cf: slides on CS systems) to be superposed to produce a snapshot.
• Thus from a single snapshot with n x n pixels, we would like to reconstruct a video of size n x n x T. The dictionary needs to be trained on patches of size m x m x T (m much less than n).
• Thus ideally, we would require the availability of videos acquired at a frame rate of 60T (or close to it).
• If you wish to design a system with a different value of T (say T'), you may ideally need to re-train the dictionary on patches taken from videos with appropriate frame rate.

Blind compressed sensing

- In standard compressed sensing, we assume knowledge of the basis (may be orthonormal or even overcomplete) in which the signal is sparse.
- However this knowledge may not be available to us always.
- For example, consider a remote sensing application with compressive acquisitions of water bodies followed by forests over mountain ranges.
- Images of the former are likely to be highly sparse in DCT whereas the latter may not be – and another basis may serve a better purpose.
- Blind compressed sensing is the task of inferring the signal basis as well as the signal coefficients directly from the compressed data.

Blind compressed sensing

- Consider *N* signals and their compressive measurements in the following form:

$$y_i = \Phi_i x_i + \eta_i, \Phi_i \in R^{m \times n}, x_i \in R^n, y_i \in R^m, m << n$$

- Our sparsity model for each signal is as follows:

$$x_i = D\theta_i, D \in R^{n \times K}, \theta_i \in R^K$$

- Combined model is:

$$y_i = \Phi_i D\theta_i + \eta_i$$

Blind compressed sensing

- The objective function to be minimized is:

$$E(D,\{\theta_i\}) = \sum_{i=1}^{N} \left\| y_i - \Phi_i D\theta_i \right\|^2 \text{ s.t.} \left\| \theta_i \right\|_0 \le T, d_k^t d_k = 1$$

- The algorithm for this is alternating in nature: sparse coding assuming a fixed dictionary, and dictionary updates assuming sparse codes.

- Sparse coding: OMP

$$E_1(\theta_i) = \left\| y_i - \Phi_i D\theta_i \right\|^2 \text{ s.t.} \left\| \theta_i \right\|_0 \le T$$

Blind compressed sensing

- The dictionary update is as follows:

$$E(d_k) = \sum_{i=1}^{N} \left\| y_i - \Phi_i \sum_{l=1}^{K} d_l \theta_{il} \right\|^2$$

$$= \sum_{i=1}^{N} \left\| y_i - \Phi_i \sum_{l=1, l \neq k}^{K} d_l \theta_{il} - \Phi_i d_k \theta_{ik} \right\|^2$$

$$= \sum_{i=1}^{N} \left\| y_i^{(-k)} - \Phi_i d_k \theta_{ik} \right\|^2$$

Blind compressed sensing

- Taking derivative w.r.t. d_k and setting to 0 gives:

$$-2\sum_{i=1}^{N}\Phi_i^t(y_i^{(-k)}-\Phi_i d_k\theta_{ik})\theta_{ik}=0$$

$$\rightarrow d_k=\left(\sum_{i=1}^{N}\Phi_i^t\Phi_i\theta_{ik}^2\right)^{-1}\sum_{i=1}^{N}\Phi_i^t y_i^{(-k)}\theta_{ik}$$

$$d_k\leftarrow d_k/\|d_k\|_2$$

$$\theta_{ik}=\frac{d_k^t\Phi_i^t y_i^{(-k)}}{d_k^t\Phi_i^t\Phi_i d_k}$$

This matrix will be invertible with probability 1 only if N is such that Nm >= n. Notice that it is important for every point to have a different sensing matrix, otherwise the dictionary cannot be inferred.

- Repeat for other columns iteratively.

Blind CS: Pseudocode

- Start with a random dictionary, infer the sparse codes for each data point.
- Given the sparse codes, infer each column of the dictionary keeping the others fixed. Infer the corresponding dictionary coefficients as well.
- Repeat the previous two steps till a convergence criterion is reached.

Blind compressed sensing: experiments

- Synthetic dictionary of size 50 by 10 ($K = 10$) generated iid from N(0,1).
- Training signals generated using a linear combination of these dictionary columns, with coefficients generated from N(0,80).
- Corresponding compressive measurements are generated using a different random projection for each training signal.
- The dictionary and coefficients are inferred from the compressed data, starting with random initial conditions.

Blind compressed sensing: experiments

- The inferred dictionary columns are compared to the true dictionary columns using dot products.
- The number of correctly identified (i.e. dot product of more than 0.95) dictionary columns is computed.
- Plots seen on next slide.

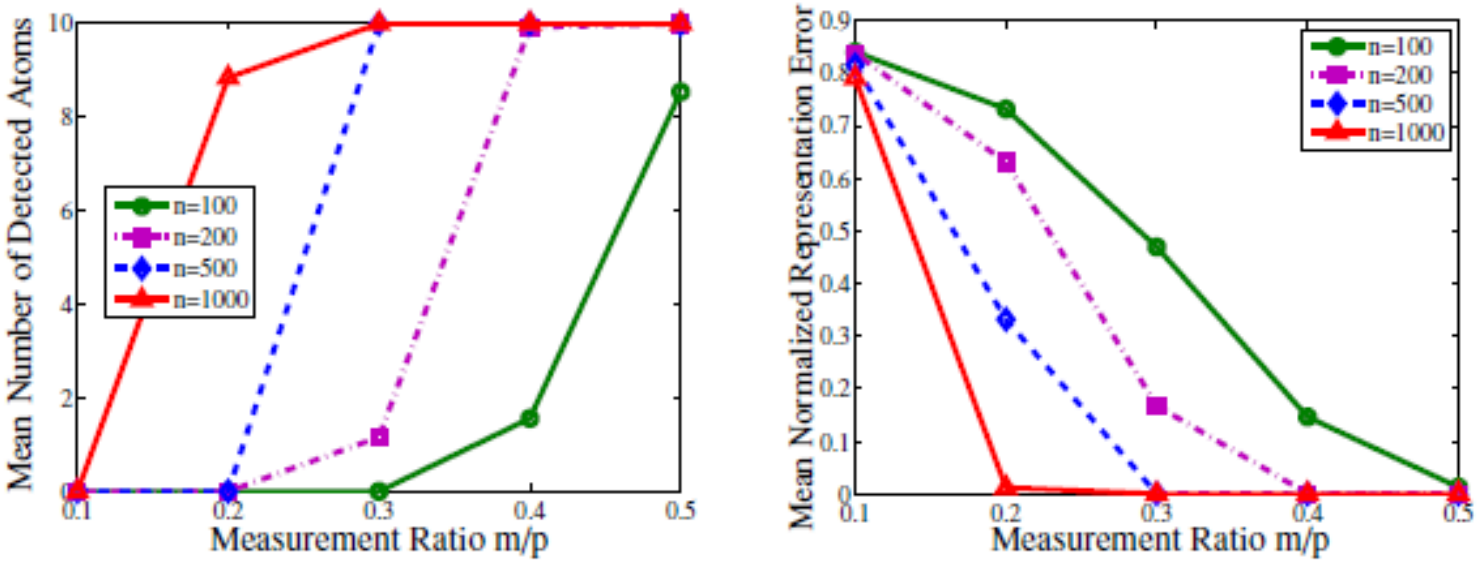
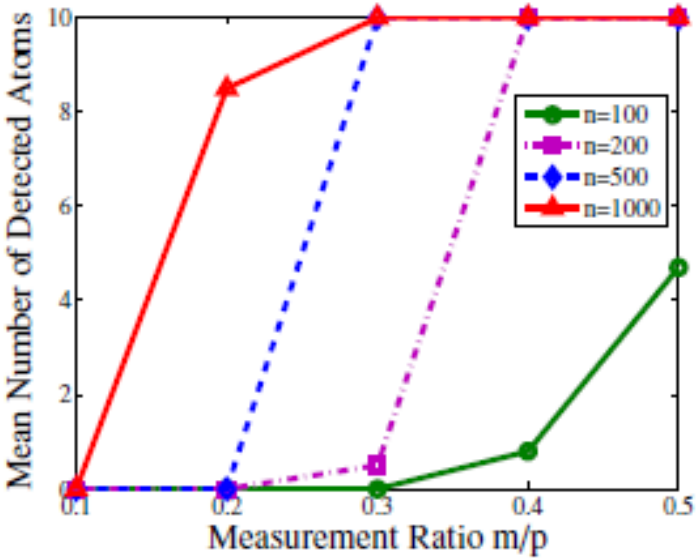
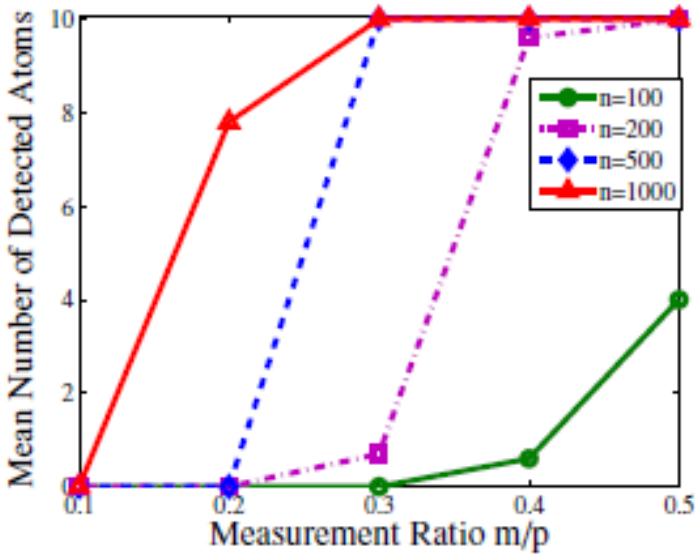


Fig. 2: Results for synthetic data. (a) Plot of the mean number of detected atoms in 50 trials for varying n and m/p. (b) Plot of mean normalized representation error of the training signals in 50 trials for varying n and m/p.

https://arxiv.org/pdf/1504.01169.pdf



(a)

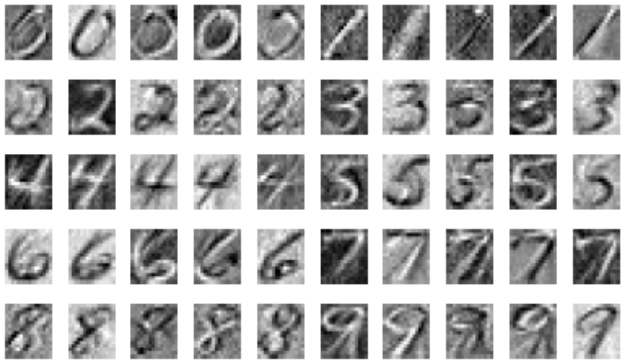


(b)

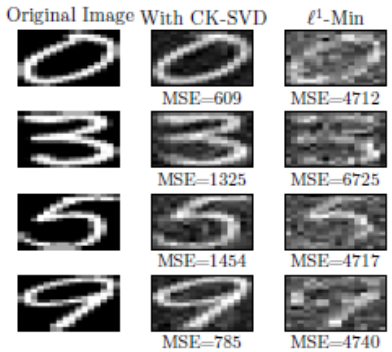
Fig. 3: Results for noisy CS measurements. Plot of mean number of detected atoms in 10 trials for the case that (a) SNR $= 50$ dB, and (b) SNR $= 30$ dB.

https://arxiv.org/pdf/1504.01169.pdf

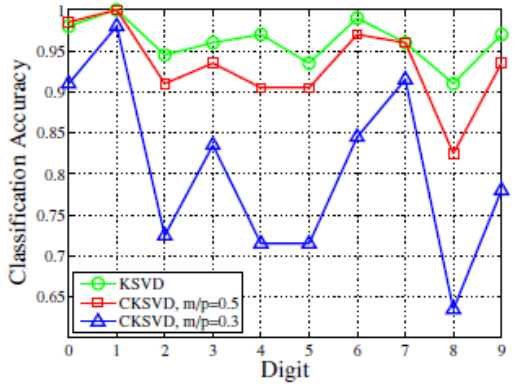
(a) Samples



(b) The Learned Dictionary



(c) CS Recovery



(d) Classification

Fig. 4: Results for the USPS dataset. (a) Samples from the USPS dataset. (b) Samples from the learned dictionary atoms using CK-SVD on ten classes of the USPS dataset for the measurement ratio $m/p = 0.5$. (c) Reconstruction accuracy comparison of OMP using the CK-SVD learned dictionary and ℓ^1-minimization using Daubechies 8 wavelets. (d) Plot of classification accuracy for K-SVD on the data versus CK-SVD on the CS measurements.

Transform Learning for Classification

Transform learning

- Techniques like PCA, NMF, KSVD, MOD, etc. are design for (compact) image representation.
- They sometimes yield great results on image classification.
- But they are not designed for optimal image classification.

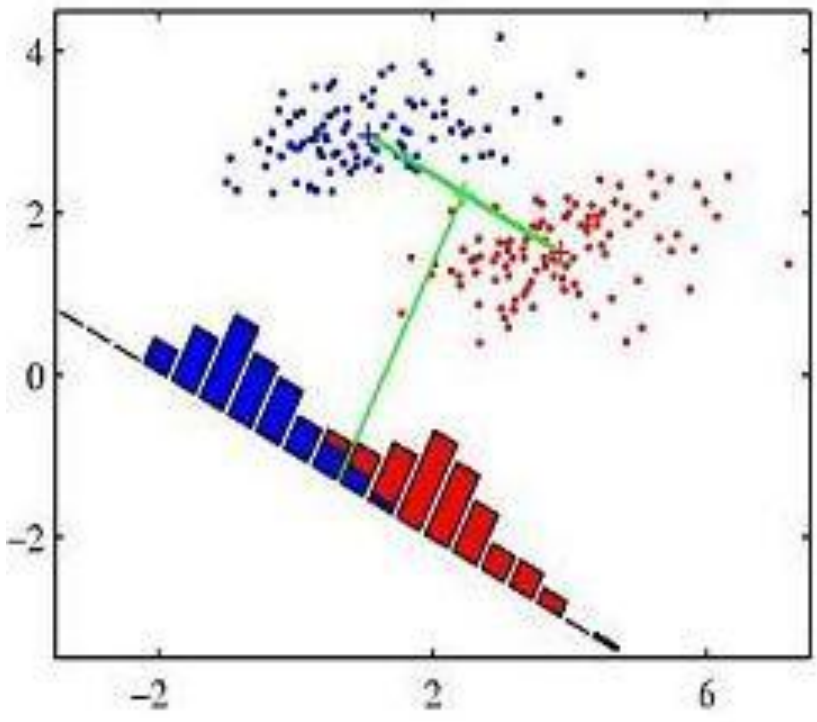
Transform learning

- Given input points $\{\mathbf{x}_i\}$, i = 1 to *N*, we will learn a transformation vector **w** with the aim of optimal classification of the transformed points $y_i = \mathbf{w}^t\mathbf{x}_i$.

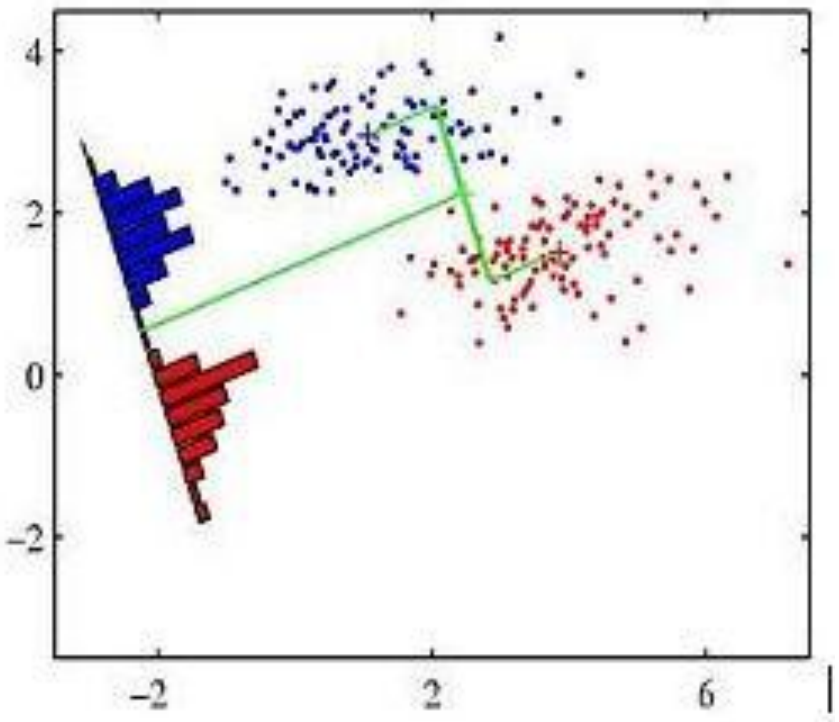
- Two techniques for this:

❑ Fisher's linear discrimant

❑ Mutual information based discriminant



Not so good **w**



Good **w**

https://stackoverflow.com/questions/33844198/reproduce-fisher-linear-discriminant-figure

Fisher's linear discriminant (FLD)

- Also called linear discriminant analysis (LDA).
- We start off considering c = 2 classes C1 and C2 and generalize later.
- We have:

$$\mathbf{m_1} = \frac{\sum_{\mathbf{x_i} \in C_1}^{N} \mathbf{x_i}}{n_1}, \mathbf{m_2} = \frac{\sum_{\mathbf{x_i} \in C_2}^{N} \mathbf{x_i}}{n_2}$$

Fisher's linear discriminat (FLD)

- Consider:

$$y_i = \mathbf{w}^t \boldsymbol{x_i};$$

$$\tilde{m}_1 = \frac{\sum_{\boldsymbol{x_i} \in C_1}^{N} \mathbf{w}^t \boldsymbol{x_i}}{n_1} = \mathbf{w}^t \boldsymbol{m_1},$$

$$\tilde{m}_2 = \frac{\sum_{\boldsymbol{x_i} \in C_2}^{N} \mathbf{w}^t \boldsymbol{x_i}}{n_2} = \mathbf{w}^t \boldsymbol{m_2}$$

Criterion for a good **w**:

The transformed mean vectors should be as far apart as possible.

But that is not enough. We want the difference between the transformed mean vectors to be large RELATIVE to the variances of the transformed points of each class. **Otherwise, there may be conflation between the points of the two classes due to a large spread even if their means are far apart.**

Fisher's linear discriminat (FLD)

- Consider:

$$\tilde{s}_1^2 = \sum_{y_i \in C_1} (y_i - \tilde{m}_1)^2 , \tilde{s}_2^2 = \sum_{y_i \in C_2} (y_i - \tilde{m}_2)^2$$

- The vector **w** should be designed based on maximizing the following criterion:

$$J(\boldsymbol{w}) = \frac{\left\| \tilde{m}_1 - \tilde{m}_2 \right\|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

$$= \left\| \boldsymbol{w}^t (\boldsymbol{m}_1 - \boldsymbol{m}_2) \right\|^2 = \boldsymbol{w}^t (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^t \boldsymbol{w}$$

$$= \boldsymbol{w}^t \boldsymbol{S}_B \boldsymbol{w}$$

Between-class scatter matrix

$$J(\boldsymbol{w}) = \frac{\left\|\tilde{m}_1 - \tilde{m}_2\right\|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

$$= \left\|\boldsymbol{w}^t(\boldsymbol{m}_1 - \boldsymbol{m}_2)\right\|^2 = \boldsymbol{w}^t(\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^t\boldsymbol{w}$$

$$= \boldsymbol{w}^t\boldsymbol{S}_B\boldsymbol{w}$$

Between-class scatter matrix

$$\tilde{s}_1^2 = \sum_{y_i \in C_1}\left\|y_i - \tilde{m}_1\right\|^2 = \sum_{y_i \in C_1}\boldsymbol{w}^t(\boldsymbol{x}_i - \boldsymbol{m}_1)(\boldsymbol{x}_i - \boldsymbol{m}_1)^t\boldsymbol{w} = \boldsymbol{w}^t\boldsymbol{S}_1\boldsymbol{w}$$

$$\tilde{s}_2^2 = \boldsymbol{w}^t\boldsymbol{S}_2\boldsymbol{w}$$

$$\tilde{s}_1^2 + \tilde{s}_2^2 = \boldsymbol{w}^t(\boldsymbol{S}_1 + \boldsymbol{S}_2)\boldsymbol{w} = \boldsymbol{w}^t\boldsymbol{S}_W\boldsymbol{w}$$

Within-class scatter matrix

$$\therefore J(\boldsymbol{w}) = \frac{\boldsymbol{w}^t\boldsymbol{S}_B\boldsymbol{w}}{\boldsymbol{w}^t\boldsymbol{S}_W\boldsymbol{w}}$$

Find $\boldsymbol{w}$ which maximizes this criterion – often called the Fisher criterion

Fisher's linear discriminant

- Taking derivatives and setting to 0, we have:

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^t \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^t \boldsymbol{S}_W \boldsymbol{w}}, \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0 \rightarrow$$

$$\boldsymbol{w}^t \boldsymbol{S}_W \boldsymbol{w}(2\boldsymbol{S}_B \boldsymbol{w}) = \boldsymbol{w}^t \boldsymbol{S}_B \boldsymbol{w}(2\boldsymbol{S}_W \boldsymbol{w}) \rightarrow$$

$$\boldsymbol{S}_B \boldsymbol{w} = \lambda \boldsymbol{S}_W \boldsymbol{w} \rightarrow \boldsymbol{S}_W^{-1} \boldsymbol{S}_B \boldsymbol{w} = \lambda \boldsymbol{w}$$

$$\lambda = \boldsymbol{w}^t \boldsymbol{S}_B \boldsymbol{w} / \boldsymbol{w}^t \boldsymbol{S}_W \boldsymbol{w}$$

Called a Generalized eigenvalue problem.
Assumes that $\mathbf{S_W}$ is not singular.

Fisher's linear discriminant

- In the $c = 2$ case:

$$\boldsymbol{S_B} = (\boldsymbol{m_1} - \boldsymbol{m_2})(\boldsymbol{m_1} - \boldsymbol{m_2})^t$$

$$\rightarrow \boldsymbol{S_B w} = \beta(\boldsymbol{m_1} - \boldsymbol{m_2}) \text{ for some scalar } \beta = (\boldsymbol{m_1} - \boldsymbol{m_2})^t \boldsymbol{w}$$

$$\text{Recall: } S_W^{-1} \boldsymbol{S_B w} = \lambda \boldsymbol{w} \rightarrow S_W^{-1} \beta(\boldsymbol{m_1} - \boldsymbol{m_2}) = \lambda \boldsymbol{w}$$

$$\rightarrow \boldsymbol{w} \text{ points in the same direction as } S_W^{-1}(\boldsymbol{m_1} - \boldsymbol{m_2})$$

- Thus FLD determines a direction that optimizes the criterion mentioned earlier.

- Since $\mathbf{S_B}$ is rank 1, only a single such direction is possible.

Fisher's linear discriminant: classification

- Thus FLD determines a direction that optimizes the criterion mentioned earlier.

- It reduces dimensionality from *d* to 1.

- The actual classification however proceeds by nearest neighbor method in the transformed space.

Fisher's linear discriminant: case of *c* classes

- In this case, we project the *d*-dimensional points $\boldsymbol{x}_i$ onto a *c*-1 dimensional space (we assume $d \geq c$).

- Generalizing the within-class scatter matrix:

$$\boldsymbol{S}_W = \sum_{i=1}^{c} \boldsymbol{S}_i, \boldsymbol{S}_i = \sum_{\boldsymbol{x}_j \in C_i} (\boldsymbol{x}_j - \boldsymbol{m}_i)(\boldsymbol{x}_j - \boldsymbol{m}_i)^t,$$

$$\boldsymbol{m}_i = \sum_{x_j \in C_i} \boldsymbol{x}_j / n_i$$

Fisher's linear discriminant: case of *c* classes

- Define the total scatter matrix:

$$\boldsymbol{m} = \sum_{i=1}^{c} \sum_{x_j \in C_i} \boldsymbol{x_j} / N = \sum_{i=1}^{c} n_i \boldsymbol{m_i} / N,$$

$$\boldsymbol{S_T} = \sum_{j=1}^{N} (\boldsymbol{x_j} - \boldsymbol{m})(\boldsymbol{x_j} - \boldsymbol{m})^t$$

Fisher's linear discriminant: case of *c* classes

$$\boldsymbol{m} = \sum_{i=1}^{c} \sum_{\boldsymbol{x_j} \in C_i} \boldsymbol{x_j} / N = \sum_{i=1}^{c} n_i \boldsymbol{m_i} / N,$$

$$\boldsymbol{S_T} = \sum_{j=1}^{N} (\boldsymbol{x_j} - \boldsymbol{m})(\boldsymbol{x_j} - \boldsymbol{m})^t = \sum_{i=1}^{c} \sum_{x_j \in C_i} (\boldsymbol{x_j} - \boldsymbol{m})(\boldsymbol{x_j} - \boldsymbol{m})^t$$

$$= \sum_{i=1}^{c} \sum_{\boldsymbol{x_j} \in C_i} (\boldsymbol{x_j} - \boldsymbol{m_i} + \boldsymbol{m_i} - \boldsymbol{m})(\boldsymbol{x_j} - \boldsymbol{m_i} + \boldsymbol{m_i} - \boldsymbol{m})^t$$

$$= \sum_{i=1}^{c} \sum_{\boldsymbol{x_j} \in C_i} (\boldsymbol{x_j} - \boldsymbol{m_i})(\boldsymbol{x_j} - \boldsymbol{m_i})^t + \sum_{i=1}^{c} \sum_{x_j \in C_i} (\boldsymbol{m_i} - \boldsymbol{m})(\boldsymbol{m_i} - \boldsymbol{m})^t$$

$$= \boldsymbol{S_W} + \sum_{i=1}^{c} n_i (\boldsymbol{m_i} - \boldsymbol{m})(\boldsymbol{m_i} - \boldsymbol{m})^t = \boldsymbol{S_W} + \boldsymbol{S_B}$$

Fisher's linear discriminant: case of *c* classes

- We perform a dimensionality reduction from *d* dimensions to *c*-1 dimensions using a transform matrix **W** of size *d* x (*c*-1) where:

$$\forall i, \boldsymbol{y_i} = \boldsymbol{W}^T \boldsymbol{x_i}$$

- We define scatter matrices in the transformed space:

$$\tilde{\boldsymbol{S}}_{\boldsymbol{W}} = \boldsymbol{W}^T \boldsymbol{S}_{\boldsymbol{W}} \boldsymbol{W}, \tilde{\boldsymbol{S}}_{\boldsymbol{B}} = \boldsymbol{W}^T \boldsymbol{S}_{\boldsymbol{B}} \boldsymbol{W}$$

Fisher's criterion: case of *c* classes

- The Fisher criterion which we maximize w.r.t. **W** is:

$$J(\boldsymbol{W}) = \frac{\left|\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W}\right|}{\left|\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W}\right|}$$

$$\tilde{\boldsymbol{S}}_W = \boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W}, \tilde{\boldsymbol{S}}_B = \boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W}$$

The determinant of the scatter matrix is a scalar measure of the scatter. As the determinant is the product of eigenvalues, it is a product of variances in the principal directions and thus the square of the scattering volume of the hyperellipsoid.

- This is a generalized eigenvalue problem again.

Fisher's criterion: case of *c* classes

- The solution is a set of *c*-1 generalized eigenvectors of the following form:

$$\boldsymbol{S}_B \boldsymbol{w}_i = \lambda_i \boldsymbol{S}_W \boldsymbol{w}_i$$

- There are no more than *c*-1 such eigenvectors with non-zero eigenvalues as the rank of $\mathbf{S_B}$ is at the most *c*-1. (As $\mathbf{S_B}$ is the sum of *c* rank-one matrices.)

Fisher's linear discriminant: uniqueness

- Is the solution **W** unique?

- Answer is no – it is unique only up to an arbitrary non-singular linear transformation because

$$J(\boldsymbol{W}) = \frac{\left|\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W}\right|}{\left|\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W}\right|} = \frac{\left|(\boldsymbol{WA})^T \boldsymbol{S}_B \boldsymbol{WA}\right|}{\left|(\boldsymbol{WA})^T \boldsymbol{S}_W (\boldsymbol{WA})\right|} = \frac{|\boldsymbol{A}|\left|\boldsymbol{W}^T \boldsymbol{S}_B \boldsymbol{W}\right||\boldsymbol{A}|}{|\boldsymbol{A}||\boldsymbol{W}^T \boldsymbol{S}_W \boldsymbol{W}||\boldsymbol{A}|}$$

Fisherfaces: applications in face recognition

- Problem: the matrix $\mathbf{S_W}$ becomes singular.

- Why? Because the number of images is usually much smaller than the image dimensions!

- PCA to the rescue!

Fisherfaces: applications in face recognition

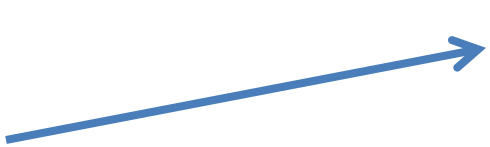
- Reduce the dimensionality of the *d*-dimensional points – of which there are *N* – using PCA.

- Project onto the largest *N*-*c* eigenvectors – the former causes no loss of information. (This technique is called "Fisherfaces"- see equation 5 of this.)

- Then perform FLD to reduce dimensionality to *c*-1. See equation on next slide.

Fisherfaces: applications in face recognition

- The criterion for this modified FLD is as follows:



$$W_{pca} = \arg\max_{W} |W^T S_T W|$$

$$W_{fld} = \arg\max_{W} \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|}.$$

Results with Fisherfaces

- Training data: Database of face images under different lighting conditions.
- Subsets 1, 2, 3, 4, 5 where the latitudinal and longitudinal angles of the light source direction are respectively within 15, 30, 45, 60, 75 degrees of the camera axis.
- Interpolation: Training on sets 1 and 5 – testing on the others.
- Extrapolation: training on set 1, testing on the others.

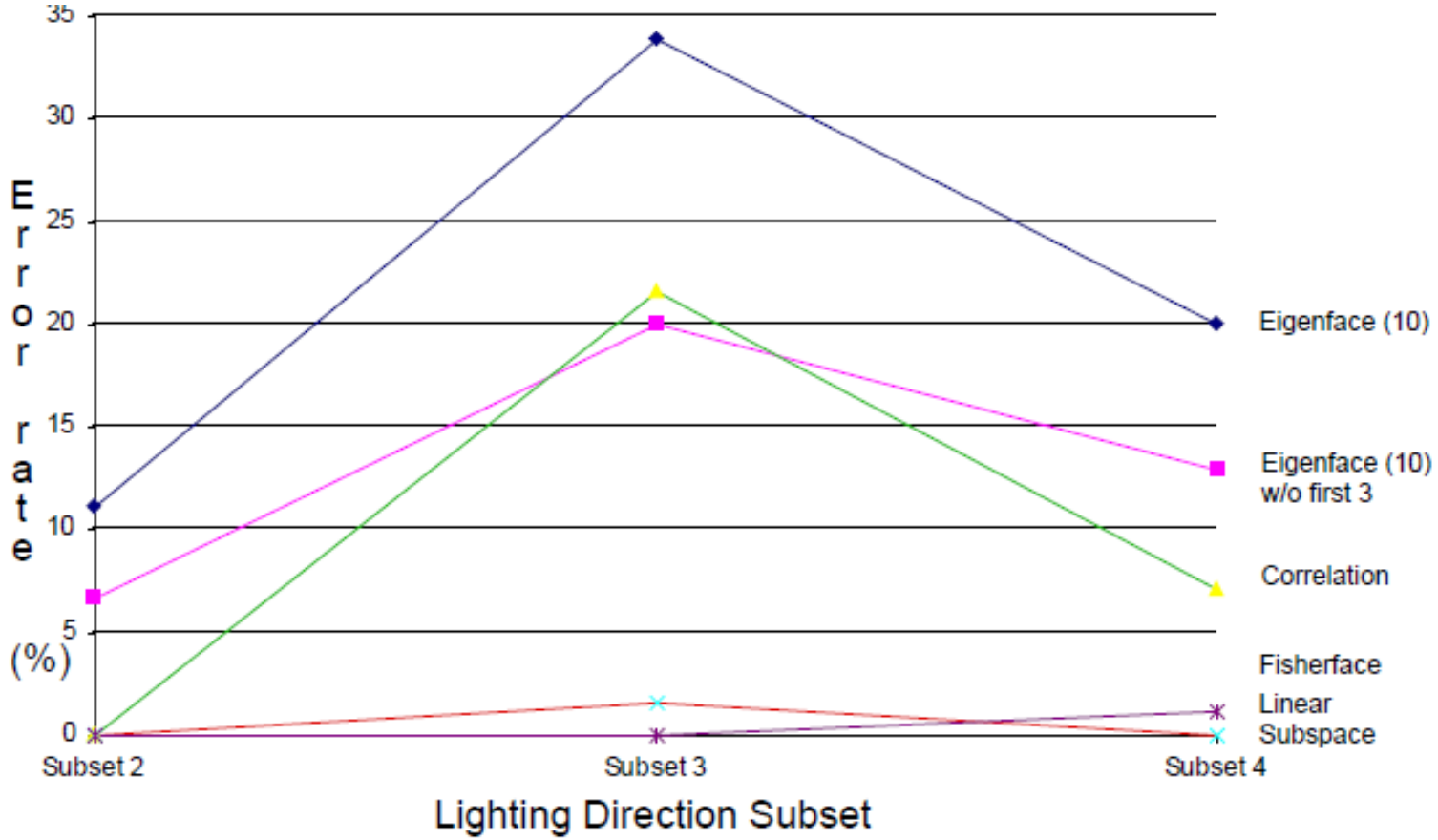


Fig. 4. Interpolation: When each of the methods is trained on images from both near frontal and extreme lighting (Subsets 1 and 5), the graph and corresponding table show the methods' relative performance under intermediate lighting conditions.

Fisherfaces produces better results!

http://www.face-rec.org/algorithms/LDA/belhumeur96eigenfaces.pdf

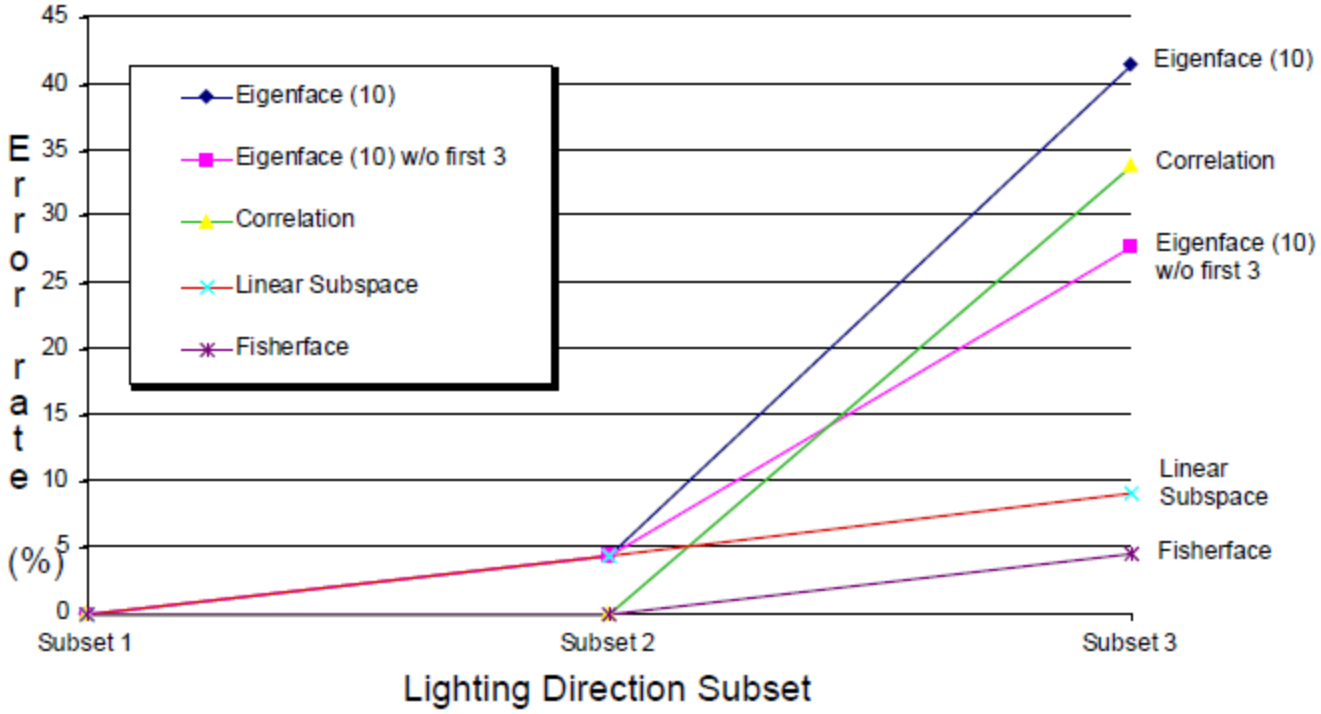


Fig. 3. Extrapolation: When each of the methods is trained on images with near frontal illumination (Subset 1), the graph and corresponding table show their relative performance under extreme light source conditions.

Fisherfaces produces better results!

http://www.face-rec.org/algorithms/LDA/belhumeur96eigenfaces.pdf

Classification using Mutual Information

- The FLD approach is constrained to reduce data dimensions to at most *c*-1 when there are *c* classes.

- It is desirable to remove this limitation and have more flexibility.

- Mutual information = quantity that characterizes the dependence between two random variables.

Mutual information: definition

- Given random variables X and Y, the mutual information between then I(X,Y) is defined as:

$$I(X,Y) = \sum_{X=x,Y=y} p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right)$$

- For continuous random variables, it is:

$$I(X,Y) = \iint p(x,y) \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right) dxdy$$

Mutual information: definition

- It thus tells you how much information *X* and *Y* contain about each other – or how close they come to being independent of each other.

- MI has the property of being non-negative and symmetric.

- Zero-valued for independent random variables.

MI for classification

- Consider data $\{\mathbf{x}_i\}$ for $i = 1$ to N where each $\mathbf{x}_i$ in R^D .
- Each $\mathbf{x}_i$ has label c_i which belongs to exactly one of N_c classes.
- Let X = R.V. for data, C = R.V. for class label.
- Infer transform vector $\mathbf{w}$ such that the mutual information between $\{c_i\}$ and $\{\mathbf{y}_i = \mathbf{w}^t\mathbf{x}_i\}$ is maximized.

MI for classification

- Infer transform vector **w** such that the mutual information between $\{c_i\}$ and $\{\mathbf{y_i} = \mathbf{w}^t\mathbf{x_i}\}$ is maximized.

- For our problem, we seek to maximize:

$$I(C,Y) = \sum_c \int p(c,\boldsymbol{y}) \log \frac{p(c,\boldsymbol{y})}{P(c)\,p(\boldsymbol{y})}\, d\boldsymbol{y}$$

- To compute MI, we need expressions for all the probabilities.

Probability expressions

- The expression for P(c) is

$$P(c_p) = J_p / N, J_p = (\#\text{ points in class } p)/N$$

- Expression for p($\mathbf{y}$):

$$p(\mathbf{y}) = \sum_{c=1}^{N_c} p(c, \mathbf{y}) = \sum_{c=1}^{N_c} p(\mathbf{y}|c)P(c)$$

$$p(\mathbf{y}|c_p) = \frac{1}{J_p} \sum_{j=1}^{J_p} G(\mathbf{y} - \mathbf{y}_{pj}, \sigma^2 I)$$

$$p(c_p, \mathbf{y}) = \frac{1}{N} \sum_{j=1}^{J_p} G(\mathbf{y} - \mathbf{y}_{pj}, \sigma^2 I), \quad p = 1, \ldots, N_c$$

Segway: Kernel density estimation

- What motivates the specific expression for $p(\mathbf{y}|c_p)$?

$$p(\mathbf{y}|c_p) = \frac{1}{J_p} \sum_{j=1}^{J_p} G(\mathbf{y} - \mathbf{y}_{pj}, \sigma^2 I)$$

- This is a common way of **non-parametric** estimation of the probability density.
- Non-parametric: because it does not assume any families of densities (eg: Gaussian, Laplacian, Cauchy, etc.)

Segway: Kernel density estimation

- G in the expression below is called the kernel, which is often chosen to be Gaussian.

$$p(\mathbf{y}|c_p) = \frac{1}{J_p} \sum_{j=1}^{J_p} G(\mathbf{y} - \mathbf{y}_{pj}, \sigma^2 I)$$

- This estimator is often called the Parzen-Rosenblatt density estimator.

- Histogramming is another form of non-parametric density estimation – but produces discontinuous estimates.

- Note: Histogramming and kernel method are both **estimates** of a true underlying (unknown) probability density.

Segway: Kernel density estimation

- The choice of σ in this estimator is critical:

$$p(\mathbf{y}|c_p) = \frac{1}{J_p} \sum_{j=1}^{J_p} G(\mathbf{y} - \mathbf{y}_{pj}, \sigma^2 I)$$

- The optimal σ depends on the second derivatives of the underlying true density – which is unknown.

- A rule of thumb is to choose σ = O($n^{-0.2}$) where n is the number of samples.

Probability expressions

- Expression for p(**y**):

$$p(\mathbf{y}) = \sum_{c=1}^{N_c} p(c, \mathbf{y}) = \sum_{c=1}^{N_c} p(\mathbf{y}|c)P(c)$$

$$p(\mathbf{y}|c_p) = \frac{1}{J_p} \sum_{j=1}^{J_p} G(\mathbf{y} - \mathbf{y}_{pj}, \sigma^2 I)$$

$$p(c_p, \mathbf{y}) = \frac{1}{N} \sum_{j=1}^{J_p} G(\mathbf{y} - \mathbf{y}_{pj}, \sigma^2 I), \quad p = 1, \ldots, N_c$$

$$p(\mathbf{y}) = \frac{1}{N} \sum_{p=1}^{N_c} \sum_{j=1}^{J_p} G(\mathbf{y} - \mathbf{y}_{pj}, \sigma^2 I) = \frac{1}{N} \sum_{i=1}^{N} G(\mathbf{y} - \mathbf{y}_i, \sigma^2 I)$$

Quadratic mutual information

- The mutual information expression involves log of summations – difficult to efficiently solve.

- Instead, we use the quadratic mutual information (QMI):

$$QMI(c, \boldsymbol{y}) = \sum_{c} \int (p(c, \boldsymbol{y}) - P(c)\, p(\boldsymbol{y}))^2 d\boldsymbol{y}$$

QMI expressions

$$I_T(C,Y) = \sum_c \int_{\mathbf{y}} p(c,\mathbf{y})^2 d\mathbf{y} + \sum_c \int_{\mathbf{y}} P(c)^2 p(\mathbf{y})^2 d\mathbf{y} - 2\sum_c \int_{\mathbf{y}} p(c,\mathbf{y}) P(c) p(\mathbf{y}) d\mathbf{y}$$

$$V_{IN} \equiv \sum_c \int_{\mathbf{y}} p(c,\mathbf{y})^2 d\mathbf{y}$$

$$V_{ALL} \equiv \sum_c \int_{\mathbf{y}} P(c)^2 p(\mathbf{y})^2 d\mathbf{y}$$

$$V_{BTW} \equiv \sum_c \int_{\mathbf{y}} p(c,\mathbf{y}) P(c) p(\mathbf{y}) d\mathbf{y}$$

$$I_T(C,Y) = V_{IN} + V_{ALL} - 2V_{BTW} \qquad \frac{\partial I_T}{\partial \mathbf{y}_i} = \frac{\partial V_{IN}}{\partial \mathbf{y}_i} + \frac{\partial V_{ALL}}{\partial \mathbf{y}_i} - 2\frac{\partial V_{BTW}}{\partial \mathbf{y}_i}$$

QMI expressions

$$V_{IN} \equiv \sum_c \int_{\mathbf{y}} p(c,\mathbf{y})^2 d\mathbf{y} \qquad I_T(C,Y) = V_{IN} + V_{ALL} - 2V_{BTW}$$

$$V_{ALL} \equiv \sum_c \int_{\mathbf{y}} P(c)^2 p(\mathbf{y})^2 d\mathbf{y}$$

$$V_{BTW} \equiv \sum_c \int_{\mathbf{y}} p(c,\mathbf{y}) P(c) p(\mathbf{y}) d\mathbf{y}$$

Within-class differences

$$V_{IN}(\{c_i,\mathbf{y}_i\}) = \sum_c \int_{\mathbf{y}} p(c,\mathbf{y})^2 d\mathbf{y} = \frac{1}{N^2} \sum_{p=1}^{N_c} \sum_{k=1}^{J_p} \sum_{l=1}^{J_p} G(\mathbf{y}_{pk} - \mathbf{y}_{pl}, 2\sigma^2 I)$$

$$V_{ALL}(\{c_i,\mathbf{y}_i\}) = \sum_c \int_{\mathbf{y}} P(c)^2 p(\mathbf{y})^2 d\mathbf{y} = \frac{1}{N^2} \left(\sum_{p=1}^{N_c} \left(\frac{J_p}{N} \right)^2 \right) \sum_{k=1}^{N} \sum_{l=1}^{N} G(\mathbf{y}_k - \mathbf{y}_l, 2\sigma^2 I)$$

$$V_{BTW}(\{c_i,\mathbf{y}_i\}) = \sum_c \int_{\mathbf{y}} p(c,\mathbf{y}) P(c) p(\mathbf{y}) d\mathbf{y} = \frac{1}{N^2} \sum_{p=1}^{N_c} \frac{J_p}{N} \sum_{j=1}^{J_p} \sum_{k=1}^{N} G(\mathbf{y}_{pj} - \mathbf{y}_k, 2\sigma^2 I) \ .$$

Differences of all pairs of samples
(regardless of class)

QMI expressions for derivatives

$$\frac{\partial I_T}{\partial \mathbf{y}_i} = \frac{\partial V_{IN}}{\partial \mathbf{y}_i} + \frac{\partial V_{ALL}}{\partial \mathbf{y}_i} - 2\frac{\partial V_{BTW}}{\partial \mathbf{y}_i}$$

$$\frac{\partial}{\partial \mathbf{y}_i} G(\mathbf{y}_i - \mathbf{y}_j, 2\sigma^2 I) = G(\mathbf{y}_i - \mathbf{y}_j, 2\sigma^2 I)\frac{(\mathbf{y}_j - \mathbf{y}_i)}{2\sigma^2}$$

$$\frac{\partial}{\partial \mathbf{y}_{ci}} V_{IN} = \frac{1}{N^2\sigma^2}\sum_{k=1}^{J_c} G(\mathbf{y}_{ck} - \mathbf{y}_{ci}, 2\sigma^2 I)(\mathbf{y}_{ck} - \mathbf{y}_{ci})$$

$$\frac{\partial}{\partial \mathbf{y}_{ci}} V_{ALL} = \frac{1}{N^2\sigma^2}\left(\sum_{p=1}^{N_c}\left(\frac{J_p}{N}\right)^2\right)\sum_{k=1}^{N} G(\mathbf{y}_k - \mathbf{y}_i, 2\sigma^2 I)(\mathbf{y}_k - \mathbf{y}_i)$$

$$\frac{\partial}{\partial \mathbf{y}_{ci}} V_{BTW} = \frac{1}{N^2\sigma^2}\sum_{p=1}^{N_c}\frac{J_p + J_c}{2N}\sum_{j=1}^{J_p} G(\mathbf{y}_{pj} - \mathbf{y}_{ci}, 2\sigma^2 I)(\mathbf{y}_{pj} - \mathbf{y}_{ci})$$

QMI optimization

- Proceeds by projected gradient ascent with adaptive step-size and can reach a local maximum of the QMI:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \frac{\partial I}{\partial \mathbf{w}} = \mathbf{w}_t + \eta \sum_{i=1}^{N} \frac{\partial I}{\partial \mathbf{y}_i} \frac{\partial \mathbf{y}_i}{\partial \mathbf{w}}$$

- This was for optimizing over a single direction.
- For multiple directions, we need to optimize over a matrix **W** (size *D* x *d, d* < *D*) such that $\mathbf{y_i} = \mathbf{W}^T\mathbf{x_i}$.
- We need to impose orthonormality of **W** – hence the need for a projection step.

QMI optimization: orthogonalization

- Let $\mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_d}$ be the row vectors of $\mathbf{W}$ – which is not at this point orthogonal.

- Define the following operator:

$$proj_{\boldsymbol{u}}(\boldsymbol{v}) = \frac{\boldsymbol{v} \bullet \boldsymbol{u}}{\boldsymbol{u} \bullet \boldsymbol{u}} \boldsymbol{u}$$

- The next slide gives the orthogonalization procedure – called Gram-Schmidt O.P.

Gram Schmidt Orthogonalization Procedure

$$\boldsymbol{u_1 = w_1, e_1 = u_1 / \|u_1\|_2}$$

This forces $\mathbf{u_2}$ to be orthogonal to $\mathbf{u_1}$.

$$\boldsymbol{u_2 = w_2 - proj_{u_1}(w_2), e_2 = u_2 / \|u_2\|_2,}$$

…

$$\boldsymbol{u_d = w_d - \sum_{k=1}^{d-1} proj_{u_k}(w_d), e_d = u_d / \|u_d\|_2}$$

$$\boldsymbol{NOTE: u_2 \bullet u_1 = (w_2 - proj_{u_1}(w_2)) \bullet u_1}$$

$$= (w_2 - \frac{w_2 \bullet u_1}{\|u_1\|^2} u_1) \bullet u_1 = w_2 \bullet u_1 - w_2 \bullet u_1 = 0$$

https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process

Results

- Three classes:

❑ C1: Bimodal Gaussians with centers at (1,0,0) and (-1,0,0)

❑C2: Bimodal Gaussians with centers at (0,1,0) and (0,-1,0)

❑C3: Single Gaussian at (0,0.7,1)

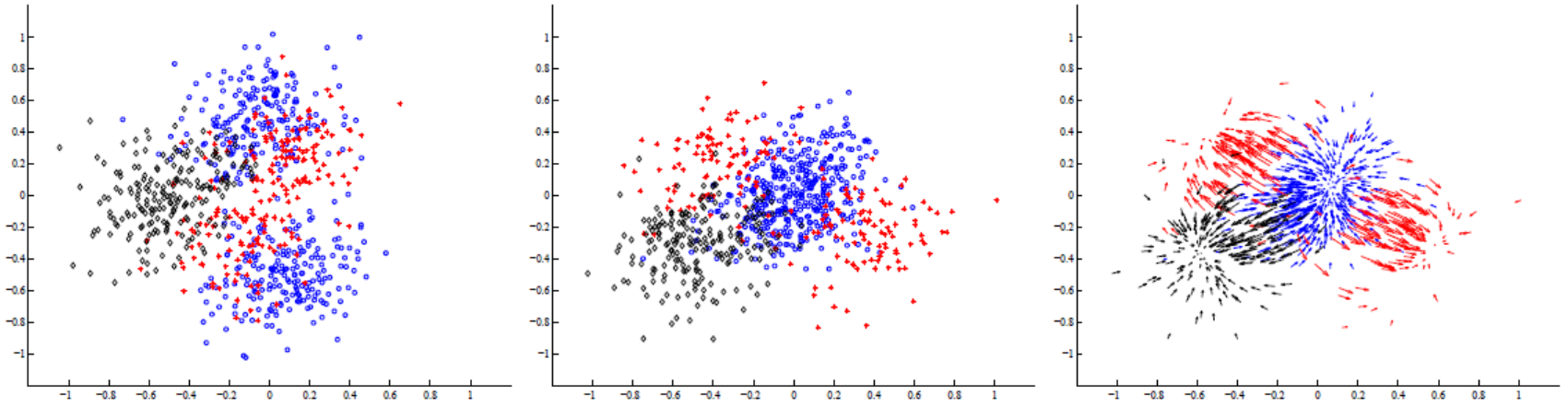


Figure 2: Three classes in three dimensions projected onto a two-dimensional subspace using both LDA (left) and MMI-based projection (center). Final information forces, with a narrow kernel, are shown on the right.

http://www.jmlr.org/papers/volume3/torkkola03a/torkkola03a.pdf

Compressed Sensing with Overcomplete Dictionaries

Towards CS with overcomplete dictionaries

- A **basis** in a vector space contains vectors that are all linearly independent.
- A **frame** generalizes this concept to vectors that may be linearly dependent.
- A set of vectors $\mathbf{e_k}$, 1 <= *k* <= *n*, form a **frame** if there exist positive values A and B, A <= B, such that for every vector **v**, we have:

$$A\|\mathbf{v}\|^2 \leq \sum_k |\mathbf{v}^t\mathbf{e_k}|^2 \leq B\|\mathbf{v}\|^2$$

Towards CS with overcomplete dictionaries

- A **frame** is overcomplete or redundant if it not a basis (of the vector space).
- A frame is called a **tight frame** if A = B. For a tight frame, we have for any **v**:

$$\mathbf{v} = \frac{1}{A}\sum_{k}(\mathbf{v}^{t}\mathbf{e_k})\mathbf{e_k}$$

Towards CS with overcomplete dictionaries

- We mention a generalization of the restricted isometry property called the D-RIP.

Definition 1.3 (D-RIP) *Let Σ_s be the union of all subspaces spanned by all subsets of s columns of D. We say that the measurement matrix A obeys the restricted isometry property adapted to D (abbreviated D-RIP) with constant δ_s if*

$$(1-\delta_s)\|v\|_2^2 \leq \|Av\|_2^2 \leq (1+\delta_s)\|v\|_2^2$$

holds for all $v \in \Sigma_s$.

We point out that Σ_s is just the image under D of all s-sparse vectors.

Note that Σ_s is the image under **D** of all *s*-sparse vectors. The only difference is that **D** here has more columns than rows unlike the case of orthonormal matrices before. Also random Gaussian and Bernoulli matrices obey D-RIP with high probability given O(s log n) rows, just like they obey RIP.

Candes et al, *Compressed sensing with coherent and redundant dictionaries*, Applied and Computational Harmonic Analysis, 2010

CS with tight frames

- Let Ψ be a tight frame of size n x d, and Φ be a m x n measurement matrix satisfying D-RIP with $\delta_{2s} < 0.08$ or $\delta_{7s} < 0.6$. Then the solution $\mathbf{x}^*$ to the optimization problem P1* below satisfies:

$$\left\|\mathbf{x}^* - \mathbf{x}\right\|_2 \le C_0 \varepsilon + C_1 \frac{\left\|\Psi^* \mathbf{x} - (\Psi^* \mathbf{x})_s\right\|_1}{\sqrt{s}}$$

$$P1^*: \min \left\|\boldsymbol{\Psi}^* \boldsymbol{x}\right\|_1 \text{ such that } \left\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\right\|_2 \le \varepsilon;$$

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\eta};\ \ \boldsymbol{x} \text{ is the signal, } \boldsymbol{y} \text{ is the measurement vector}$$

$$\boldsymbol{\Psi}^* \boldsymbol{x} \text{ is the signal - coefficient vector}$$

References

- Section 3.8 of “Pattern Classification” by Duda and Hart
- http://en.wikipedia.org/wiki/Eigenface
- M. Turk and A. Pentland (1991). "Eigenfaces for recognition". *Journal of Cognitive Neuroscience*

Appendix: Covariance matrix

- In probability and statistics, the expected value of a scalar random variable x is called its mean:

$$\mu = E(x) = \int x p(x) dx \approx \frac{1}{N} \sum_{i=1}^{N} x_i$$

Sample values of the random variable

Probability density function of x

- The variance of a scalar random variable is the expected value of its squared deviation around the mean:

$$\sigma^2 = E((x-\mu)^2) = \int (x-\mu)^2 \, p(x) dx \approx \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2$$

Appendix: Covariance matrix

- The expected value of a vector random variable is called its mean vector:

$$\vec{\boldsymbol{\mu}} = E(\vec{\mathbf{x}}), \vec{\mathbf{x}} = (x(1), x(2), ..., x(d)), \vec{\boldsymbol{\mu}} = (\mu(1), \mu(2), ..., \mu(d))$$

$$\mu_k \approx \frac{1}{N}\sum_{i=1}^{N} x_{ki}, 1 \le k \le d$$

k-th sample value of $\mathbf{x}_i$

Appendix: Covariance matrix

- The variance is now replaced by a covariance matrix. Each entry contains the covariance between two elements of the vector:

$$\mathbf{C} = \begin{pmatrix} E((x(1)-\mu(1))^2) & E((x(1)-\mu(1))(x(2)-\mu(2))) & . & . & E((x(d)-\mu(d))(x(1)-\mu(1))) \\ E((x(2)-\mu(2))(x(1)-\mu(1))) & E((x(2)-\mu(2))^2) & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ E((x(1)-\mu(1))(x(d)-\mu(d))) & . & . & . & E((x(d)-\mu(d))^2) \end{pmatrix} \in R^{d \times d}$$

$$C_{kl} = E((x(k)-\mu(k))(x(l)-\mu(l))) \approx \frac{1}{N-1}\sum_{i=1}^{N}(x_i(k)-\mu(k))(x_i(l)-\mu(l)), 1 \le k \le d, 1 \le l \le d$$

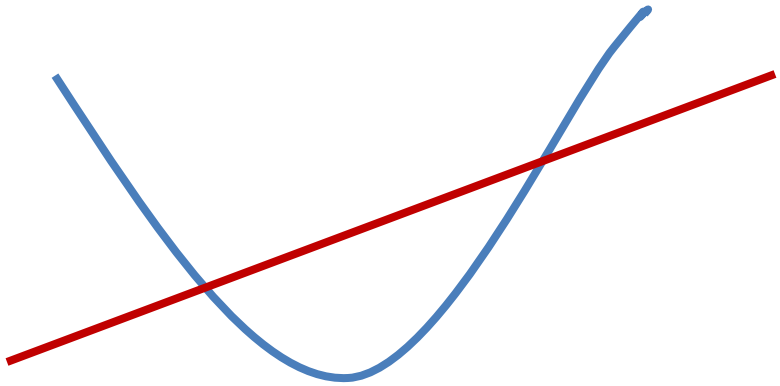
$$Note: \mu(k) = k-th \text{ element of vector } \mu$$

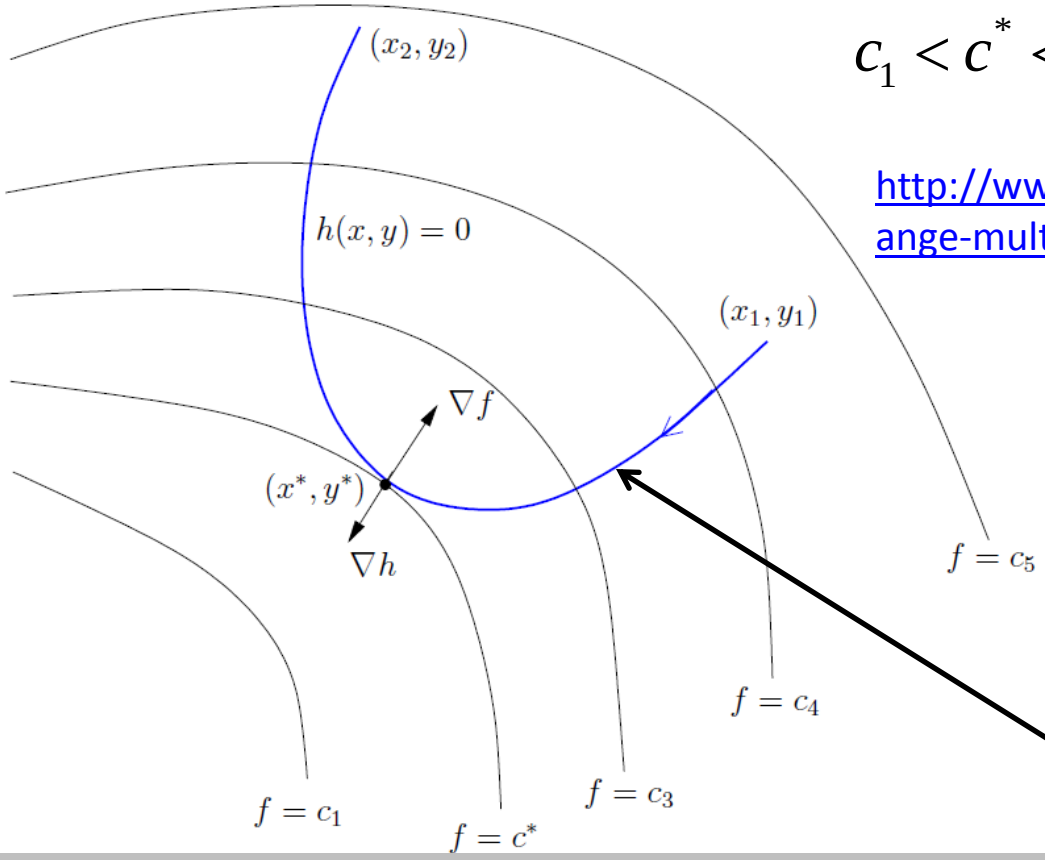
Appendix: A word about Lagrange Multipliers

- Consider you want to find the minimum or maximum of f(x,y).
- Normally, you take the derivative and set it to 0 and check the sign of the second derivative.
- Now you want to find the optimum subject to a constraint that h(x,y) = 0.

Appendix: A word about Lagrange Multipliers

- Physical analogy: if you drop a pebble into a parabola-shaped bowl, the pebble will fall to the bottom-most point.
- But if you placed a wooden plank into the bowl, the pebble will settle somewhere on the plank!





$$c_1 < c^* < c_2 < c_3 < c_4 < c_5$$

http://www.cs.iastate.edu/~cs577/handouts/lagrange-multiplier.pdf

Suppose we are walking from (x_1,y_1) to (x_2,y_2) along the constraint curve h(x,y) = 0. Initially, the tangent vector along h(x,y) = 0 has a component along –grad(f), and hence there is a decrease in the value of f(x,y) as we move along h(x,y) = 0. If the point (x*,y*) is a local minimum of f(x,y), a small motion along h(x,y) = 0 from (x*,y*) will have no component along grad(f), else it would cause an increase in the value of f. Hence the tangent to h(x,y) = 0 at (x*,y*) will be perpendicular to grad(f). As we move further, motion along h(x,y) = 0 will have a component along +grad(f) leading to an increase in the value of f(x,y). At the minimum point, we have grad(f) perpendicular to the tangent , i.e. grad(f) and grad(h) are collinear. Hence, there exists some value λ (the **Lagrange multiplier)** such that we have:

$$\nabla f(x^*, y^*) = \lambda \nabla h(x^*, y^*) \rightarrow \nabla(f(x^*, y^*) - \lambda h(x^*, y^*)) = 0$$

If (x',y') is the extremum of function $f(x,y)$, then a small perturbation to (x',y') will lead to no change in the value of f - in the limit when the magnitude of the perturbation is 0.

Hence we have $|\nabla \mathbf{f}(x', y') = 0|$.

But here we are constrained to move only along the curve $h(x,y) = 0$.

Let $\mathbf{s}$ be the tangent to this curve at a point (x', y').

The magnitude of the change in f due to infinitesimal movement along the curve is given by $\nabla \mathbf{f}(x', y') \bullet \mathbf{s}$.

If (x', y') is an extremum of f along this curve, then we must have

$\nabla \mathbf{f}(x', y') \bullet \mathbf{s} = 0$.

But the normal to the curve is $\nabla \mathbf{h}(x', y')$ and it is perpendicular to $\mathbf{s}$.

Hence we have $\nabla \mathbf{f}(x', y')$ is collinear with $\nabla \mathbf{h}(x', y')$.

$\therefore \nabla \mathbf{f}(x', y') = \lambda \nabla \mathbf{h}(x', y')$ for some value λ.

In our derivation for PCA, instead of $f(x,y)$ we have $\mathbf{e}^t\mathbf{S}\mathbf{e}$, and instead of $h(x, y) = 0$, we have $\mathbf{e}^t\mathbf{e} - 1 = 0$. Note that $\mathbf{e}$ is a vector in d dimensional space.