Structured Matrix Factorization

Many problems in machine learning and signal processing can be formulated as follows:
Given a $d \times n$ matrix $\boldsymbol{X}$, find a $d \times r$ matrix $\boldsymbol{B}$ and a $r \times n$ matrix $\boldsymbol{C}$ such that

$$\boldsymbol{X} \approx \boldsymbol{B}\boldsymbol{C},$$

under the constraints that $\boldsymbol{B}$ and $\boldsymbol{C}$ have some kind of *structure*.

To begin, let's review a fundamental result in modern linear algebra:

Eckart-Young Theorem: Let $\boldsymbol{X}$ be a $d \times n$ matrix with SVD $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$. Then the best rank-$r$ approximation to $\boldsymbol{X}$ is

$$\underset{\boldsymbol{Y} \in \mathbb{R}^{d \times n}}{\text{minimize}} \ \|\boldsymbol{X} - \boldsymbol{Y}\|_F^2 \quad \text{subject to} \quad \operatorname{rank}(\boldsymbol{Y}) = r,$$

is

$$\hat{\boldsymbol{Y}} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r^{\mathrm{T}},$$

where $\boldsymbol{\Sigma}_r$ is the $r \times r$ diagonal matrix containing the r largest singular values of $\boldsymbol{X}$, $\boldsymbol{U}_r$ is a $d \times r$ matrix whose columns are the corresponding left singular vectors, and $\boldsymbol{V}_r$ contains the corresponding right singular vectors.

Proof of the above is more involved than you might think, but nevertheless, it is a classical result. We can also replace the Frobenius norm $\|\cdot\|_F^2$ (sum of the squares of the singular values) in the optimization above with the spectral norm $\|\cdot\|$ (largest singular value). We could also re-write the program as

$$\underset{\substack{\boldsymbol{B} \in \mathbb{R}^{d \times r} \\ \boldsymbol{C} \in \mathbb{R}^{r \times n}}}{\text{minimize}} \ \|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{C}\|_F^2.$$

The rank constraint is now implicit in the size of the matrices $\boldsymbol{B}$ and $\boldsymbol{C}$. A solution to the above is then

$$\hat{\boldsymbol{B}} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r^{1/2}, \quad \hat{\boldsymbol{C}} = \boldsymbol{\Sigma}_r^{1/2} \boldsymbol{V}_r^{\mathrm{T}}.$$

Of course, this solution is not unique, as we may divvy up the $\boldsymbol{\Sigma}_r$ in an arbitrary way, i.e.

$$\hat{\boldsymbol{B}} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r, \quad \hat{\boldsymbol{C}} = \boldsymbol{V}_r^{\mathrm{T}},$$

or even apply a general invertible $r \times r$ matrix to the left and right:

$$\hat{\boldsymbol{B}} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r^{1/2} \boldsymbol{A}, \quad \hat{\boldsymbol{C}} = \boldsymbol{A}^{-1} \boldsymbol{\Sigma}_r^{1/2} \boldsymbol{V}_r^{\mathrm{T}},$$

etc.

Although we did not explicitly enforce this for the optimization, it is easy (and most natural) to make the factors $\hat{\boldsymbol{B}}$ and $\hat{\boldsymbol{C}}$ have *orthogonal* columns and rows, respectively.

What are we doing when we perform an approximation of this kind? Suppose that the columns of $\boldsymbol{X}$ are data points in $\mathbb{R}^d$:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \end{bmatrix}$$

and suppose that there exists a small $r \ll \min(d, n)$, such that

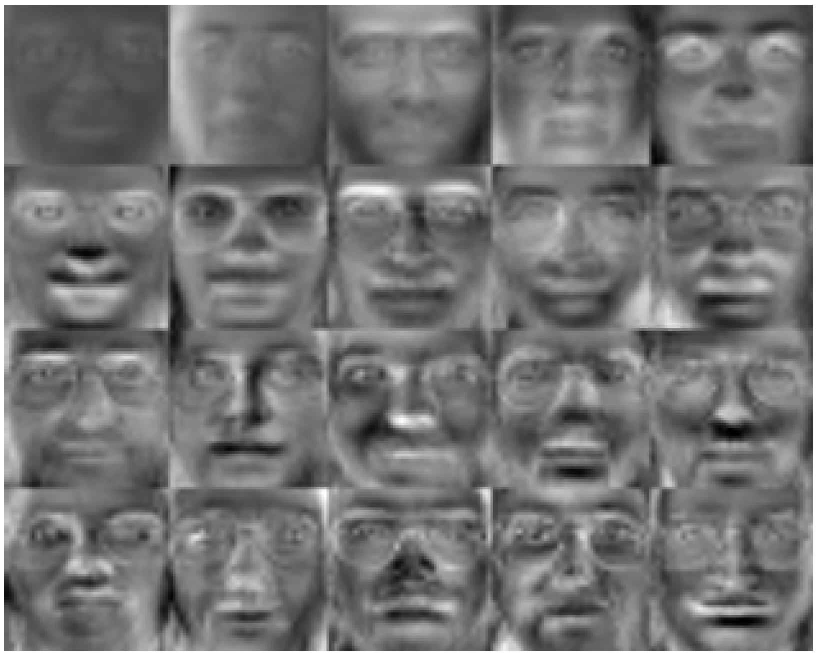
$$\begin{aligned} \boldsymbol{X} &\approx \boldsymbol{B}\boldsymbol{C} \\ &= \begin{bmatrix} \\ \boldsymbol{B} \\ \\ \end{bmatrix} \begin{bmatrix} \boldsymbol{c}_1 & \boldsymbol{c}_2 & \cdots & \boldsymbol{c}_n \end{bmatrix}, \end{aligned}$$

i.e. $\boldsymbol{x}_i \approx \boldsymbol{B}\boldsymbol{c}_i$. This means that each $\boldsymbol{x}_i$ is a different (linear) combination (specified by the entries in $\boldsymbol{c}_i$) of a small number of latent features (specified by the columns of $\boldsymbol{B}$).

Example: Eigenfaces

The data points $\boldsymbol{x}_i$ are d-pixel images[1] of faces (d is on the order of thousands):

There are many such images (n on the order of 1000s), but $\boldsymbol{X}$ can be approximated very well with rank $r = 200$. Here are the columns of $\hat{\boldsymbol{B}}$:



[1]Example from http://kixor.net/school/2008spring/comp776/assn3/.

Example: Latent Semantic Indexing

LSI is used to automatically group documents together by their semantic content. Here, the data we are given is a *document term matrix*. We parse n documents (encyclopedia/wikipedia entries, newspaper article, etc.) and count the number of times d different words appear; the tally for word k in document i is the entry $X[k,i]$:

$$X[k,i] = \# \text{ times word } k \text{ appeared in document } i.$$

The rows of $\boldsymbol{X}$ tend to be correlated, so we can approximate $\boldsymbol{X} \approx \boldsymbol{BC}$ for $r \ll \min(d,n)$. This means that word frequency counts in a document are basically a linear combination of a small number of "feature counts" given by the columns of $\boldsymbol{B}$.

But in both these cases, it is hard to interpret what the latent factors really tell us about the data, other than being able to assist us in some simple dimensionality reduction. This is because we have not enforced the proper structure on the factors $\boldsymbol{B}, \boldsymbol{C}$.

Non-negative matrix factorization

If we constrain the factors to have positive entries, we get different (and perhaps more interesting) results:

$$\underset{\substack{\boldsymbol{B}\in\mathbb{R}^{d\times r}\\ \boldsymbol{C}\in\mathbb{R}^{r\times n}}}{\text{minimize}} \ \|\boldsymbol{X}-\boldsymbol{B}\boldsymbol{C}\|_F^2, \quad \boldsymbol{B}\geq\boldsymbol{0},\ \boldsymbol{C}\geq\boldsymbol{0}. \tag{1}$$

The idea behind this is that in many cases (including the face and LSI examples above), what we are really looking for is a mixture of *positive* factors. This problem is called *non-negative matrix factorization* (NNMF).

Solving (1) is much more involved than in the unconstrained case. Not only is there no closed-form solution, but the optimization is non-convex, and comes with all the complications there-of.

But notice that if we fix one the factors, say $\boldsymbol{B}=\boldsymbol{B}^{(0)}$, then the program

$$\underset{\boldsymbol{C}\in\mathbb{R}^{r\times n}}{\text{minimize}} \ \|\boldsymbol{X}-\boldsymbol{B}^{(0)}\boldsymbol{C}\|_F^2, \quad \boldsymbol{C}\geq\boldsymbol{0}.$$

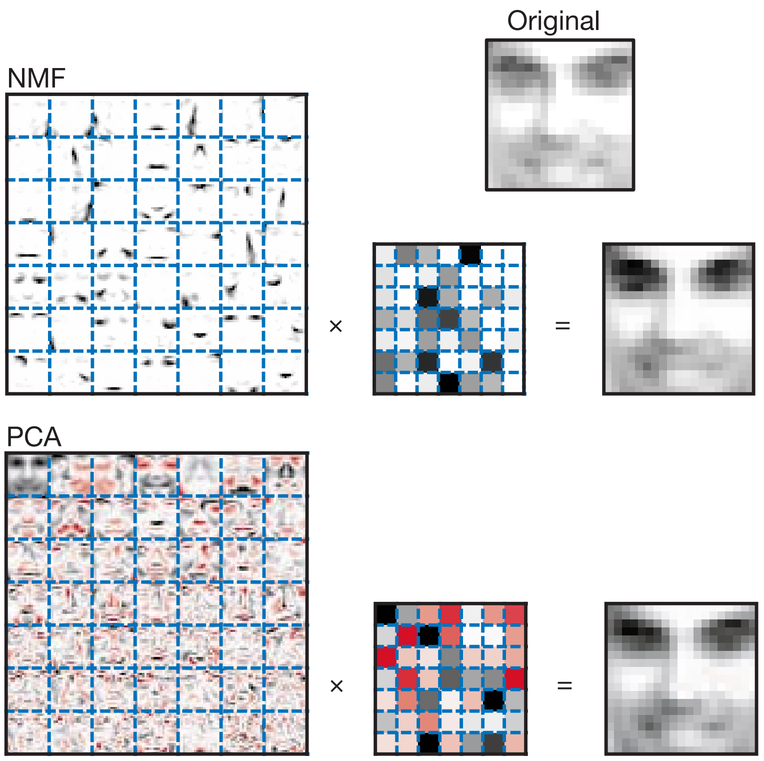
is separable by column; we can solve

$$\underset{\boldsymbol{c}_i\in\mathbb{R}^{r}}{\text{minimize}} \|\boldsymbol{x}_i-\boldsymbol{B}^{(0)}\boldsymbol{c}_i\|_2^2 \quad \text{subject to} \quad \boldsymbol{c}_i\geq 0,$$

for $i=1,\ldots,n$ independently. Each of these subproblems is a quadratics program, and so it solvable with off-the-shelf software. The same thing is true if we pin down $\boldsymbol{C}=\boldsymbol{C}^{(0)}$ and optimize over $\boldsymbol{B}$. So NNMF algorithms typically alternate back and forth between solving for $\boldsymbol{B}$ and solving for $\boldsymbol{C}$.

Here are some interesting results from the original paper on NNMF[2]

Faces:



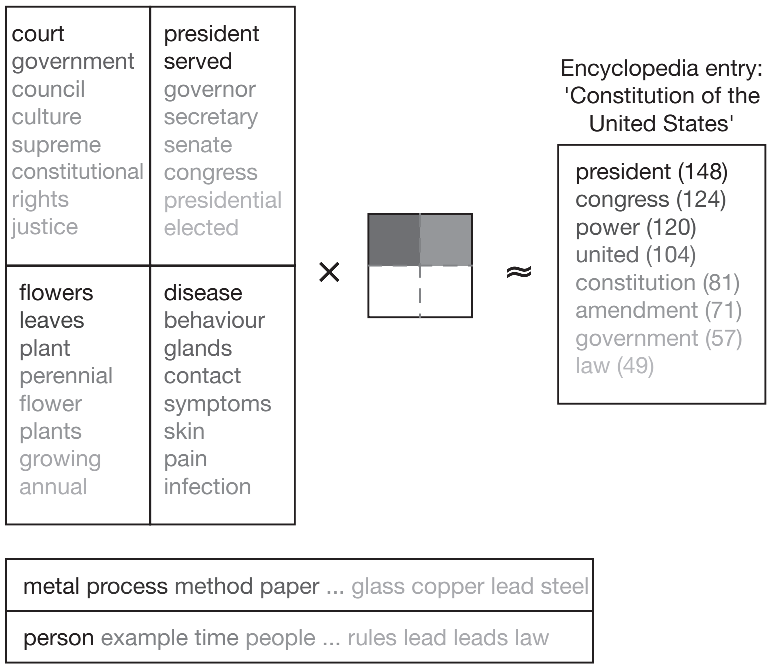
The top is essentially the eigenfaces experiment, but where we restrict the factors ($\boldsymbol{B}$) and mixture coefficients ($\boldsymbol{C}$) to be positive. Notice that, as compared to the standard PCA experiment below it, that the factors actually look like facial features.

[2]Lee and Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, October 1999.

Latent semantic indexing:

Here is an example from the same article for LSI.

$n = 30,991$ articles from an encyclopedia were parsed,
$d = 15,276$ words counted


court government council culture supreme constitutional rights justice
president served governor secretary senate congress presidential elected
flowers leaves plant perennial flower plants growing annual
disease behaviour glands contact symptoms skin pain infection
×
≈
Encyclopedia entry: 'Constitution of the United States'
president (148) congress (124) power (120) united (104) constitution (81) amendment (71) government (57) law (49)
metal process method paper ... glass copper lead steel
person example time people ... rules lead leads law

Now the columns of $\hat{\boldsymbol{B}}$ can be interpreted as *groupings* of words that appear together frequently.

On the top right, we see example weights for four different factors that decomposed the (word count of) the US Constitution ... the two factors (columns of $\boldsymbol{B}$) on top, learned while training on a separate data set, were given significant weights (entries in $\boldsymbol{c}$), while the two factors on bottom had very small weights.

The example on bottom shows the words associated with the two factors whose entry in the row corresponding to the word 'lead' was the greatest.

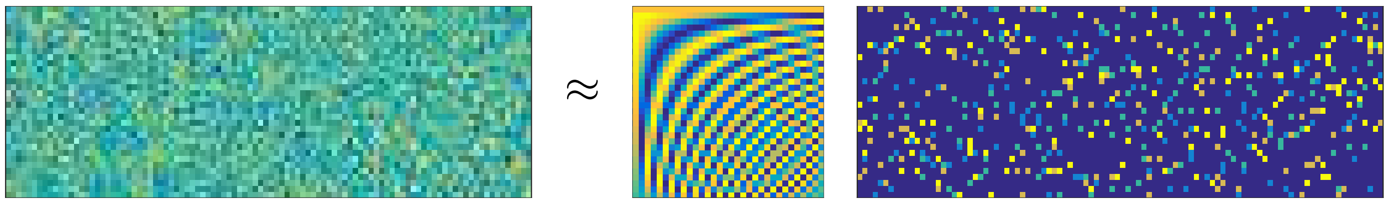
Dictionary Learning

The dictionary learning problem is to (approximately) factor a $d \times n$ matrix $\boldsymbol{X}$ as

$$\boldsymbol{X} \approx \boldsymbol{B}\boldsymbol{C},$$

where $\boldsymbol{B}$ is $d \times d$ and orthonormal (or at least invertible), and $\boldsymbol{C}$ is $d \times n$ and **sparse**. That is, the vast majority of the entries in $\boldsymbol{C}$ are non-zero.

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \end{bmatrix} \approx \begin{bmatrix} & \boldsymbol{B} & \end{bmatrix} \begin{bmatrix} \boldsymbol{c}_1 & \boldsymbol{c}_2 & \cdots & \boldsymbol{c}_n \end{bmatrix}$$



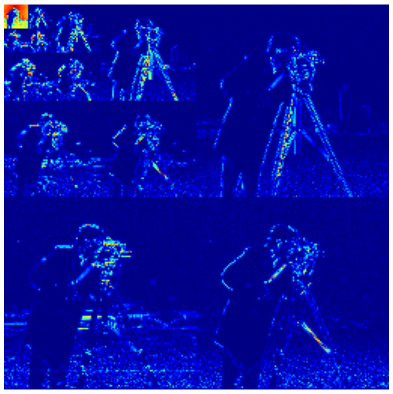
The idea is that we are looking for a basis (or *dictionary*) $\boldsymbol{B}$ such that every data point $\boldsymbol{x}_i$ can be written as a superposition of a small number of columns of $\boldsymbol{B}$. If we restrict the columns of $\boldsymbol{B}$ to be orthonormal, $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B} = \mathbf{I}$, then we can interpret this problem as a search for a *sparsifying transform* — we want a $\boldsymbol{B}$ so that the transform coefficients $\boldsymbol{c}_i = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{x}_i$ have as much of their energy compacted onto a small set as possible.

Sparsity

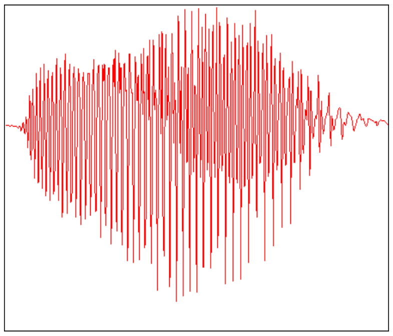
Fixed (i.e. unlearned) sparsifying transforms play a role in many places in signal and image processing.

image $\boldsymbol{x}_i$ with n pixels

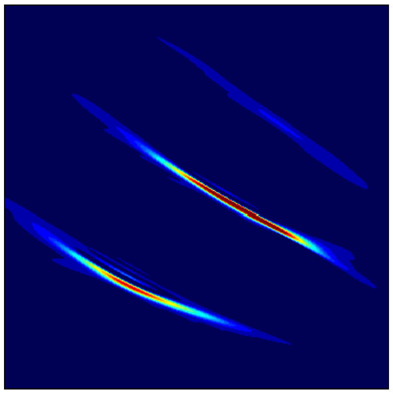
wavelet coefficients



chirp signal $\boldsymbol{x}_i$ in time domain



Gabor domain



Sparse transforms gives us a natural way to compress signals (fewer numbers to code) and to remove noise (important coefficients "stick up" past the noise floor).

Let's make some of this more precise. If $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B} = \mathbf{I}$ then then columns $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ for an orthobasis for $\mathbb{R}^d$, meaning $\boldsymbol{b}_i^{\mathrm{T}}\boldsymbol{b}_j = 0$ for $i \neq j$ and for any vector $\boldsymbol{x}$, we can write

$$\boldsymbol{x} = \sum_{k=1}^{d} c[k]\boldsymbol{b}_k, \quad \text{for} \quad c[k] = \boldsymbol{b}_k^{\mathrm{T}}\boldsymbol{x}.$$

When we say a vector $\boldsymbol{x}$ is sparse in the $\boldsymbol{B}$-domain, we mean that

$$\boldsymbol{x} \approx \sum_{k\in\mathcal{I}} c[k]\boldsymbol{b}_k,$$

for some small index set $\mathcal{I}$, $|\mathcal{I}| \ll d$[3]. If we set $\hat{\boldsymbol{x}} = \sum_{k\in\mathcal{I}} c[k]\boldsymbol{b}_k$, then by the generalized Parseval theorem, we know

$$\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2 = \sum_{k\notin\mathcal{I}} |c[k]|^2,$$

and so if the coefficients outside of $\mathcal{I}$ are all very small, this approximation error will be very small.

Note that this type of model is inherently nonlinear, as the $\mathcal{I}$ above *adapts* to the vector $\boldsymbol{x}$ (i.e. which coefficients are important is different from signal to signal). Suppose that $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are exactly s-sparse in that

$$\boldsymbol{x}_1 = \sum_{k\in\mathcal{I}_1} c_1[k]\boldsymbol{b}_k, \quad \boldsymbol{x}_2 = \sum_{k\in\mathcal{I}_2} c_2[k]\boldsymbol{b}_k,$$

for some $|\mathcal{I}_1| = |\mathcal{I}_2| = s$. Then the number of non-zero terms in $a\boldsymbol{x}_1 + b\boldsymbol{x}_2$ for $a, b \in \mathbb{R}$ are located on $\mathcal{I}_1 \cup \mathcal{I}_2$, which can be as large as $2s$ ... this means that $a\boldsymbol{x}_1 + b\boldsymbol{x}_2$ will not in general be s-sparse.

[3]We are using the notation $|\mathcal{I}|$ to mean the number of elements in the set $\mathcal{I}$.

Sparse approximation

How can we find the best sparse approximation for a vector $\boldsymbol{x}$ in a basis $\boldsymbol{B}$? The following optimization program is one way to pose this problem. Using the notation $\|\boldsymbol{c}\|_0$ = number of non-zero terms in the vector $\boldsymbol{c}$, we have

$$\underset{\boldsymbol{c}\in\mathbb{R}^d}{\text{minimize}} \ \|\boldsymbol{x}-\boldsymbol{B}\boldsymbol{c}\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{c}\|_0 \le s. \tag{2}$$

When $\boldsymbol{B}$ is orthonormal, then

$$\|\boldsymbol{x}-\boldsymbol{B}\boldsymbol{c}\|_2 = \|\boldsymbol{B}^{\mathrm{T}}(\boldsymbol{x}-\boldsymbol{B}\boldsymbol{c})\|_2 = \|\boldsymbol{y}-\boldsymbol{c}\|_2$$

where $\boldsymbol{y}$ is the vector of transform coefficients of $\boldsymbol{x}$, $\boldsymbol{y} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{x}$. The resulting program

$$\underset{\boldsymbol{c}\in\mathbb{R}^d}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{c}\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{c}\|_0 \le s.$$

is easy to solve — simply take

$$\hat{\mathcal{I}} = \text{locations of } s \text{ largest magnitude terms in } \boldsymbol{y},$$

then

$$\hat{c}[k] = \begin{cases} y[k], & k \in \hat{\mathcal{I}}, \\ 0, & k \notin \hat{\mathcal{I}}. \end{cases}$$

When $\boldsymbol{B}$ is not orthonormal but is invertible (so the columns of $\boldsymbol{B}$ form a basis for $\mathbb{R}^d$ but not an orthobasis), then

$$\|\boldsymbol{x}-\boldsymbol{B}\boldsymbol{c}\|_2 \ne \|\boldsymbol{B}^{-1}(\boldsymbol{x}-\boldsymbol{B}\boldsymbol{c})\|_2,$$

and so translating the optimization program into the $\boldsymbol{B}$ domain is not possible. In fact, in this case solving (2) exactly is NP-hard. But there are many heuristics for solving (2) in the general case — one approach is to replace the $\|\cdot\|_0$ constraint with the convex $\|\boldsymbol{c}\|_1 \le \tau$ for some appropriate τ. There are also simple greedy algorithms (most notable of which is *orthogonal matching pursuit (OMP)* approximate solutions.

Dictionary Learning Formulation

Back to the dictionary learning problem; given examples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, we want to find a $d \times d$ basis (dictionary) matrix $\boldsymbol{B}$ and a sets of sparse expansion coefficients $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n \in \mathbb{R}^d$ so that

$$\begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_n \end{bmatrix} \approx \begin{bmatrix} & \boldsymbol{B} & \end{bmatrix} \begin{bmatrix} \boldsymbol{c}_1 & \boldsymbol{c}_2 & \cdots & \boldsymbol{c}_n \end{bmatrix}.$$

Let's start by codifying this as an optimization program. We want to solve

$$\underset{\boldsymbol{B},\boldsymbol{C}}{\text{minimize}} \ \|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{C}\|_F^2 \quad \text{subject to} \quad \|\boldsymbol{c}_i\|_0 \leq s, \ i = 1, \ldots, n. \quad (3)$$

The functional[4] is simply the sum of all the approximation errors for each of the data points:

$$\|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{C}\|_F^2 = \sum_{i=1}^{n} \left\| \boldsymbol{x}_i - \sum_{k=1}^{d} c_i[k] \boldsymbol{b}_k \right\|_2^2,$$

while the constraints ensure that only s terms are non-zero in each of the $\boldsymbol{c}_i$.

The optimization program (3) is nonconvex, and impossible to solve exactly under anything but ideal conditions. One class of techniques for solving it alternate between fixing $\boldsymbol{B}$ and optimizing over $\boldsymbol{C}$, then fixing $\boldsymbol{C}$ and optimizing over $\boldsymbol{B}$. Each of these problems is more tractable.

[4]Recall that the Frobenius norm of a matrix squared $\|\boldsymbol{M}\|_F^2$ is simply the sum of all the squares of the entries in the matrix.

In fact, let's consider the case[5] where we restrict $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B} = \mathbf{I}$. With orthonormal $\boldsymbol{b}_k$, both of the subproblems have known solutions. With $\boldsymbol{B}$ fixed, we are solving

$$\underset{\boldsymbol{c}_1,\ldots,\boldsymbol{c}_n}{\text{minimize}} \sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{B}\boldsymbol{c}_i\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{c}_i\|_0 \leq s, \ i = 1, \ldots, n.$$

With $\boldsymbol{B}$ fixed, the above is really n decoupled problems of the form

$$\underset{\boldsymbol{c}_i}{\text{minimize}} \|\boldsymbol{x}_i - \boldsymbol{B}\boldsymbol{c}_i\|_2^2 \quad \text{subject to} \quad \|\boldsymbol{c}_i\|_0 \leq s.$$

We have already seen that the solution to this problem is to take $\hat{\boldsymbol{c}}_i$ to contain the s largest magnitude coefficients in $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{x}_i$, and be zero elsewhere.

Now suppose that the $\boldsymbol{c}_i$ are fixed, and we want to solve

$$\underset{\boldsymbol{B}}{\text{minimize}} \|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{C}\|_F^2 \quad \text{subject to} \quad \boldsymbol{B}^{\mathrm{T}}\boldsymbol{B} = \mathbf{I}.$$

This is known as the *orthogonal Procrustes problem*, and as you might guess by this point, its solution involves solving another eigenvalue/singular value problem. Note that

$$\begin{aligned}
\|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{C}\|_F^2 &= \|\boldsymbol{X}\|_F^2 + \|\boldsymbol{B}\boldsymbol{C}\|_F^2 - 2\langle \boldsymbol{X}, \boldsymbol{B}\boldsymbol{C}\rangle_F \\
&= \|\boldsymbol{X}\|_F^2 + \|\boldsymbol{C}\|_F^2 - 2\langle \boldsymbol{X}, \boldsymbol{B}\boldsymbol{C}\rangle_F \\
&= \|\boldsymbol{X}\|_F^2 + \|\boldsymbol{C}\|_F^2 - 2\langle \boldsymbol{X}\boldsymbol{C}^{\mathrm{T}}, \boldsymbol{B}\rangle_F
\end{aligned}$$

and so since the first two terms do not involve $\boldsymbol{B}$, the problem is equivalent to

$$\underset{\boldsymbol{B}}{\text{maximize}} \langle \boldsymbol{X}\boldsymbol{C}^{\mathrm{T}}, \boldsymbol{B}\rangle_F \quad \text{subject to} \quad \boldsymbol{B}^{\mathrm{T}}\boldsymbol{B} = \mathbf{I}.$$

[5]For more details on this case, see O. Sezer, *Data Driven Transform Optimizations...*, Georgia Tech ECE PhD Thesis, 2011.

Taking the SVD of $\boldsymbol{X}\boldsymbol{C}^{\mathrm{T}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$, we know

$$\langle \boldsymbol{X}\boldsymbol{C}^{\mathrm{T}}, \boldsymbol{B}\rangle_F = \langle \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}, \boldsymbol{B}\rangle_F = \langle \boldsymbol{\Sigma}, \boldsymbol{U}^{\mathrm{T}}\boldsymbol{B}\boldsymbol{V}\rangle_F.$$

Since $\boldsymbol{U}, \boldsymbol{B}$ and $\boldsymbol{V}$ are all $d \times d$ and orthonormal, so is $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{B}\boldsymbol{V}$. Thus, the program is equivalent to

$$\underset{\boldsymbol{Z}}{\text{maximize}} \langle \boldsymbol{\Sigma}, \boldsymbol{Z}\rangle_F, \quad \text{subject to} \quad \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z} = \mathbf{I},$$

where $\boldsymbol{\Sigma}$ is diagonal with $\Sigma[i,i] \geq 0$. Also, since

$$\langle \boldsymbol{\Sigma}, \boldsymbol{Z}\rangle_F = \sum_{i=1}^{d} \Sigma[i,i]\, Z[i,i],$$

and $Z[i,i] \leq 1$ (since the sum of the squares in each column/row must be equal to 1), we know that $\langle \boldsymbol{\Sigma}, \boldsymbol{Z}\rangle_F$ is maximized when $Z[i,i] = 1$ for all $i = 1, \ldots, d$, i.e. for $\hat{\boldsymbol{Z}} = \mathbf{I}$. This means

$$\hat{\boldsymbol{B}} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}.$$

Simple Dictionary Learning, Orthogonal Case:

- initialize orthonormal dictionary $\boldsymbol{B}^{(0)}$ (e.g. $\boldsymbol{B}^{(0)} = \mathbf{I}$)
- for $j = 1, 2, \ldots$
  - solve for $\boldsymbol{C}^{(j)}$ by keeping the s largest magnitude terms in each column of $\boldsymbol{B}^{(j-1)\mathrm{T}}\boldsymbol{X}$, and setting the rest to zero;
  - take SVD of $\boldsymbol{X}\boldsymbol{C}^{(j)^{\mathrm{T}}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$, and set $\boldsymbol{B}^{(j)} = \boldsymbol{U}\boldsymbol{V}^{\mathrm{T}}$.
- repeat until $\|\boldsymbol{B}^{(j)} - \boldsymbol{B}^{(j-1)}\|_F$ is appropriately small

The case of learning non-orthogonal $\boldsymbol{B}$ is also of interest, and many times results in a *much* sparser representation. Removing the constraint definitely changes the iterations, as the fact that $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B} = \mathbf{I}$ played a role in the derivation of both steps above. In fact, significant gains can be had by making $\boldsymbol{B}$ *overcomplete*, meaning that it has more columns than rows.

The starting point for the non-orthogonal problem is the k-SVD algorithm[6]. We will not present the details here, the interested student can see the reference below, but the algorithm has two basic steps:

- With $\boldsymbol{B}^{(j-1)}$ fixed, solve a series of n sparse approximation problems to get $\boldsymbol{C}^{(j)}$. These programs have form like (2) with non-orthogonal $\boldsymbol{B}$, so heuristics (e.g. OMP or ℓ_1 minimization) are used to solve them.
- The dictionary $\boldsymbol{B}$ is updated one column at a time — to do the update, we look at the error for all the data points that use this column, then choose the new column in such a way that this error is reduced as much as possible. Like everything in unsupervised learning, this is done by finding a leading singular vector. The coefficients are also updated in place as the corresponding columns change.

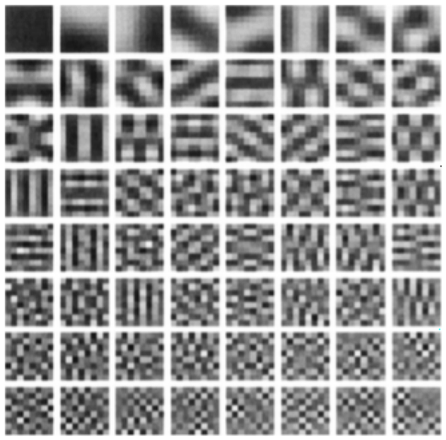
Again, details can be found in the footnoted reference.

[6]Aharon et al. "K-SVD: An algorithm for designing overcopmlete dictionaries for sparse representation," *IEEE Trans. Sig. Proc.*, November 2006.

Examples

The first people to think of the dictionary learning problem in these terms were Olshausen and Field in their seminal 1996 paper[7] They were studying human vision, and naturally wondered if most things we see can be broken-down into component parts, with only a small percentage of those parts active for any given image.

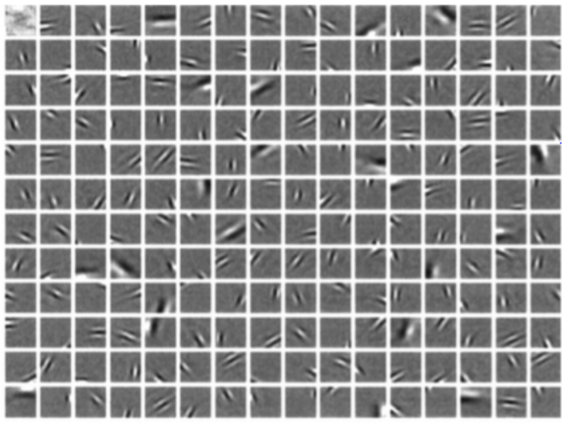
They took many 8×8 pixel image patches from natural scenes, and used them to train their sparse representation — the methods they used were similar to those discussed in the previous section. Here is the 64 dimensional orthobasis they got when they ran PCA on the image patches:



Olshausen and Field, PCA experiment

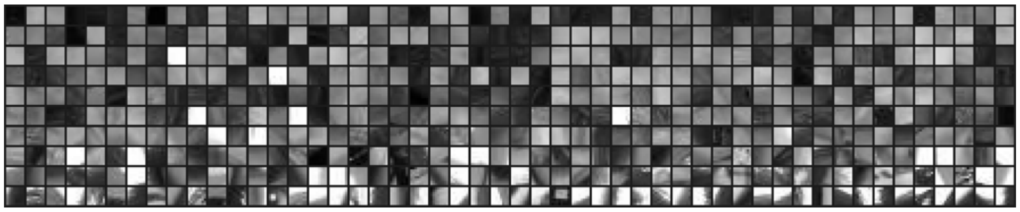
Notice that this looks pretty similar to a discrete cosine transform. Here is what they got using sparse dictionary learning on 16×16 patches:

[7]Olshausen and Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature Letters*, June 1996.



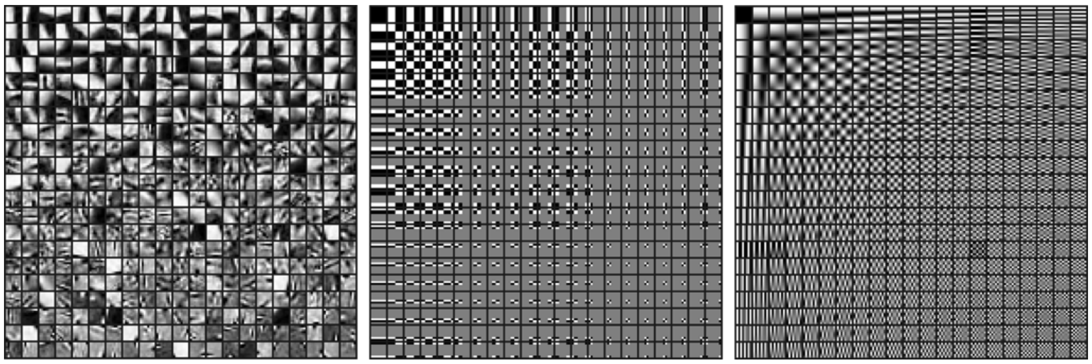
Olshausen and Field, dictionary learning experiment

Here are some results from the K-SVD paper, which used $\sim 11,000$ 8×8 blocks like this:



Aharon et al, example image patches

They trained a 64×441 overcomplete dictionary — the elements are on the left below:



Aharon et al, learned overcomplete dictionary (left)

In the middle are the Haar wavelet and discrete cosine basis functions for reference.