Sparse Regression and Dictionary Learning

Let's turn our attentions for a moment back to our oldest friend: linear regression using least-squares.

Here we are given pairs $(\boldsymbol{x}_n, y_n)$ and are trying to find $\boldsymbol{w} \in \mathbb{R}^{D+1}$ such that

$$\sum_{d=1}^{D} w[d]x_n[d] + w[D+1] \approx y_n.$$

If we stack up the $\boldsymbol{x}_n$ and y_n,

$$\boldsymbol{A} = \begin{bmatrix} - & \boldsymbol{x}_1^{\mathrm{T}} & - & 1 \\ - & \boldsymbol{x}_2^{\mathrm{T}} & - & 1 \\ & \vdots & & \vdots \\ - & \boldsymbol{x}_N^{\mathrm{T}} & - & 1 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix},$$

then this is equivalent to

$$\boldsymbol{A}\boldsymbol{w} \approx \boldsymbol{y}.$$

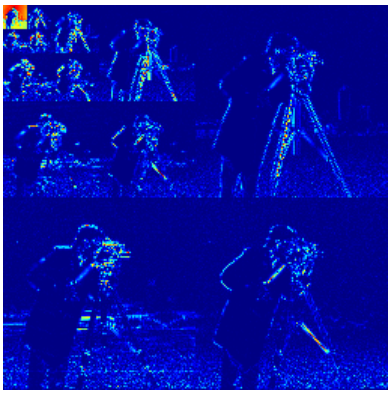
Now, it is often the case that we want to use only a small number of features (entries of the $\boldsymbol{x}_n$) to explain the responses y_n, we just don't know what these features will be a priori. The general idea is that while we have measured a great many features to populate the $\boldsymbol{x}_n$, only a small number of these will end up being important.

Sparsity

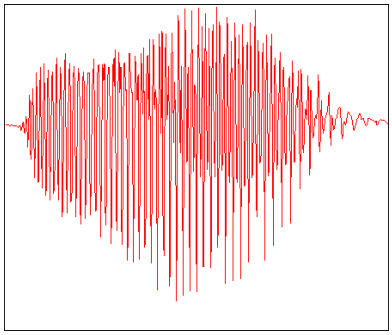
In signal and image processing, it is often possible to take a piece of data and transform it (i.e. represent it in a basis) in such a way that it (meaning the vector of basis coefficients) is sparse.

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

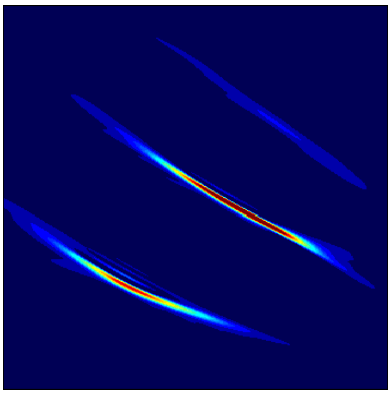
wavelet coefficients



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 $\mathcal{I}$, $|\mathcal{I}| \ll d$[1]. 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 K-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 $|\mathcal{I}_1| = |\mathcal{I}_2| = K$. 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 $2K$ ... this means that $a\boldsymbol{x}_1 + b\boldsymbol{x}_2$ will not in general be K-sparse.

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

Sparse regression refers to a general class of techniques to solve systems of equations using as few variables as possible. Here is the template problem[2]:

$$\underset{\boldsymbol{w}\in\mathbb{R}^D}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{w}\|_2^2 \quad \text{subject to} \quad \text{nnz}(\boldsymbol{w}) \leq K, \tag{1}$$

where nnz(⋮) returns the number of nonzero entries in a vector. We might not know what to make K beforehand, so we might treat this constraint as a "knob" we can turn to get different solutions. We can also put the program above in "Lagrange form" as

$$\underset{\boldsymbol{w}\in\mathbb{R}^D}{\text{minimize}} \ \|\boldsymbol{y}-\boldsymbol{A}\boldsymbol{w}\|_2^2 + \tau \cdot \text{nnz}(\boldsymbol{w}),$$

where $\tau \geq 0$ is just a different version of the knob above.

Unfortunately, unlike standard least-squares, both of these problems are very hard to solve in general (in fact, they have been both been shown to be NP-hard). There are, however, plenty of heuristics for solving them. Here are three:

Orthogonal matching pursuit (OMP)

We can think of sparse regression as trying to select a small number of columns of $\boldsymbol{A}$ that we can use to represent $\boldsymbol{y}$. That is, we are trying to choose a **subspace** that contains $\boldsymbol{y}$ that has dimension much smaller than D, where this subspace must be created by taking linear spans of the columns of $\boldsymbol{A}$.

OMP works as follows: It starts by looking for the column most closely aligned with $\boldsymbol{y}$, and then projects onto that (one dimensional)

[2]We are dropping the column of all ones in $\boldsymbol{A}$ at this point. This is just for simplicity; it is straightforward to adapt this entire discussion to include it.

subspace. Then it takes the residual (error in the approximation), looks for the column most closely aligned with it, and then projects on to the two-dimensional subspace spanned by the two columns chosen so far. The third column is then chosen based on this residual, etc.

Here is a careful statement of the algorithm ... below, $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_D$ are the columns of the $N \times D$ matrix $\boldsymbol{A}$.

Orthogonal matching pursuit

Initialize: $\boldsymbol{w}_0 = \boldsymbol{0}$ some guess, $k = 0$, $\mathcal{T}_0 = \emptyset$, $\boldsymbol{r}_0 = \boldsymbol{y}$.

while not converged, $\|\boldsymbol{r}_k\|_2 \geq \delta$ **do**

$\gamma_k = \arg\max_d |\boldsymbol{a}_d^{\mathrm{T}} \boldsymbol{r}_k| / \|\boldsymbol{a}_d\|_2$

$\mathcal{T}_{k+1} = \mathcal{T}_k \cup \{\boldsymbol{a}_{\gamma_k}\}$

$\boldsymbol{w}_{k+1} = \arg\min_{\boldsymbol{v} \in \mathcal{T}_{k+1}} \|\boldsymbol{y} - \boldsymbol{v}\|_2^2$

$\boldsymbol{r}_{k+1} = \boldsymbol{y} - \boldsymbol{w}_{k+1}$

$k = k + 1$

end while

Iterative hard thresholding

This is a simple iteration that is basically a version of projected gradient descent. We take $\boldsymbol{w}_0 = \boldsymbol{0}$, and then

$$\boldsymbol{w}_i = \mathrm{Hard}_K(\boldsymbol{w}_i + \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{w}_i)),$$

where Hard_K is the "hard thresholding" operator that sets all but the K largest terms in its argument equal to zero. The iteration above can be interpreted as taking a step in the gradient direction

for $\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{w}\|_2^2$, then projecting the result onto the set of vectors that have at most K non-zero terms. Since this set is non-convex, we are not guaranteed to find a global minimum, but there are practical situations where this algorithm performs nicely.

ℓ_1 relaxation

Vectors that are sparse tend to have small ℓ_1 norm relative to their ℓ_2 norms. For example

$$\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{v}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \\ 0 \end{bmatrix} \quad \boldsymbol{v}_3 = \begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \\ 0 \end{bmatrix} \quad \boldsymbol{v}_4 = \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix},$$

all have $\|\boldsymbol{v}_i\|_2 = 1$, but

$$\|\boldsymbol{v}_1\|_1 = 1, \quad \|\boldsymbol{v}_2\|_1 \approx 1.41, \quad \|\boldsymbol{v}_3\|_1 \approx 1.73, \quad \|\boldsymbol{v}_4\|_1 = 2.$$

In light of this, another technique for sparse regression is to simply replace $\mathrm{nnz}(\cdot)$ with $\|\cdot\|_1$, and solve

$$\underset{\boldsymbol{w} \in \mathbb{R}^D}{\text{minimize}} \ \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{w}\|_2^2 + \tau \|\boldsymbol{w}\|_1. \tag{2}$$

This program is now convex, and a global minimum can be found in any one of a number of ways. The simplest is through the iteration

$$\boldsymbol{w}_{i+1} = \mathrm{S}_\tau \left(\boldsymbol{w}_i + \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{w}_i) \right),$$

where S_τ is the *soft thresholding* operator that "shrinks" the entries of its argument towards zero:

$$\mathrm{S}_\tau(\boldsymbol{v})[d] = \begin{cases} v[d] - \tau, & v[d] > \tau, \\ 0, & |v[d]| \leq \tau, \\ v[d] + \tau, & v[d] < \tau. \end{cases}$$

This algorithm is referred to as *iterative soft thresholding*; unlike iterative hard thresholding, it is guaranteed to converge to the global minimum of (2) under very mild conditions.

Dictionary Learning

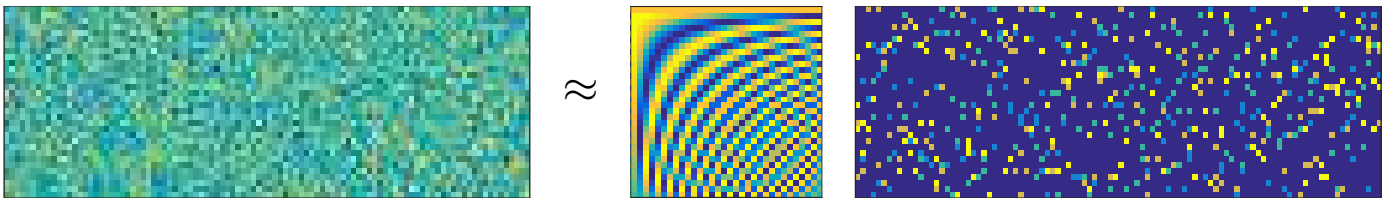
Dictionary learning is our second example of structured matrix factorization. The basic idea is that we are trying to learn, in an unsupervised way, a basis in which a given set of data points is sparse. While PCA learns a subspace, dictionary learning can be thought of as learning a **union of subspaces** model: if each data point is indeed K sparse in some basis, then the inherent model is that the data points live in one of the $\binom{D}{K}$ subspaces generated by extracting K columns from a basis for $\mathbb{R}^D$.

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.

Dictionary Learning Formulation

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

$$\underset{\boldsymbol{B},\boldsymbol{C}}{\text{minimize}} \ \|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{C}\|_F^2 \quad \text{subject to} \quad \text{nnz}(\boldsymbol{c}_n) \leq K, \ n = 1, \ldots, N. \tag{3}$$

The functional[3] is simply the sum of all the approximation errors for each of the data points:

$$\|\boldsymbol{X} - \boldsymbol{B}\boldsymbol{C}\|_F^2 = \sum_{n=1}^{N} \left\| \boldsymbol{x}_n - \sum_{d=1}^{D} c_n[d]\boldsymbol{b}_d \right\|_2^2,$$

[3]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.

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

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.

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

$$\underset{\boldsymbol{c}_1,\ldots,\boldsymbol{c}_N}{\text{minimize}} \sum_{n=1}^{N} \|\boldsymbol{x}_n - \boldsymbol{B}\boldsymbol{c}_n\|_2^2 \quad \text{subject to} \quad \text{nnz}(\boldsymbol{c}_n) \leq K,\ n = 1, \ldots, N.$$

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

$$\underset{\boldsymbol{c}_n}{\text{minimize}} \|\boldsymbol{x}_n - \boldsymbol{B}\boldsymbol{c}_n\|_2^2 \quad \text{subject to} \quad \text{nnz}(\boldsymbol{c}_n) \leq K.$$

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

Now suppose that the $\boldsymbol{c}_n$ 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 eigen-

value/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

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

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 $\Sigma[d,d]\geq 0$. Also, since

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

and $Z[d,d]\leq 1$ (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 $Z[d,d]=1$ for all $d=1,\ldots,D$, 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[4]. 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 (1) 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

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

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.

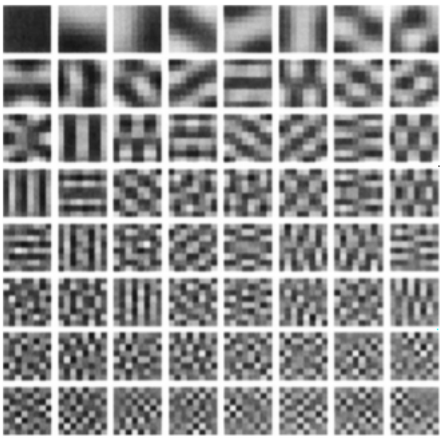
Details can be found in the footnoted reference.

Examples

The first people to think of the dictionary learning problem in these terms were Olshausen and Field in their seminal 1996 paper[5] 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.

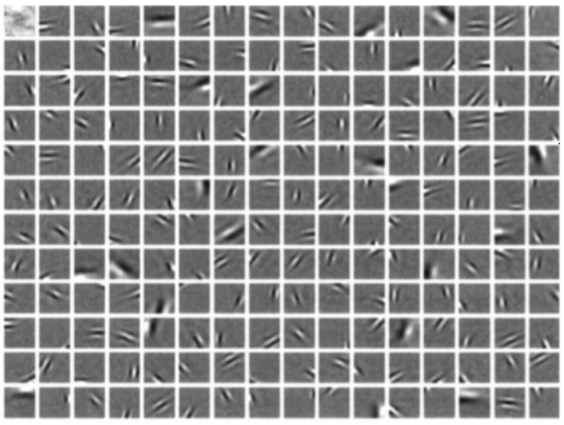
They took many 8×8 pixel image patches from natural scenes, and used them to train their sparse representation — the methods they used were similar to those discussed in the previous section. Here is the 64 dimensional orthobasis they got when they ran PCA on the image patches:

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



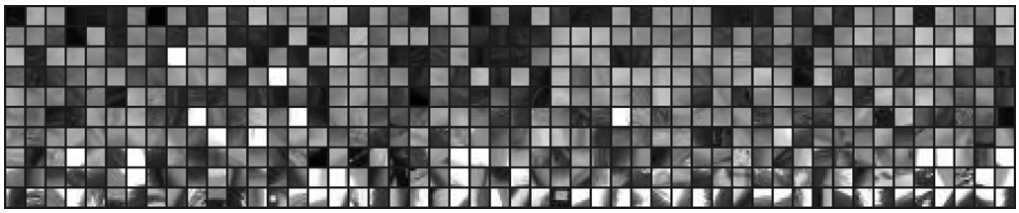
Olshausen and Field, PCA experiment

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



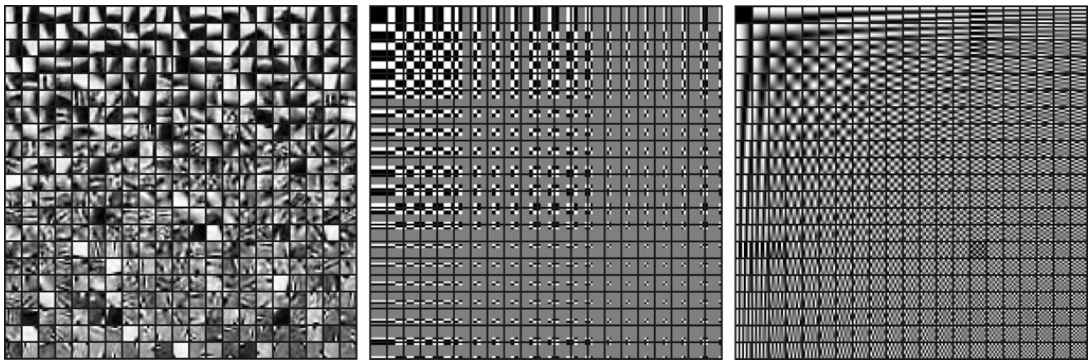
Olshausen and Field, dictionary learning experiment

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.