Deep learning

2.5. Basic clusterings and embeddings

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ
DE GENÈVE

Deep learning models combine embeddings and dimension reduction operations.

They parametrize and re-parametrize multiple times the input signal into representations that get more and more invariant and noise free.

To get an intuition of how this is possible, we consider here two standard algorithms:

- K-means, and
- Principal Component Analysis (PCA).

We will illustrate these methods on our two favorite data-sets.

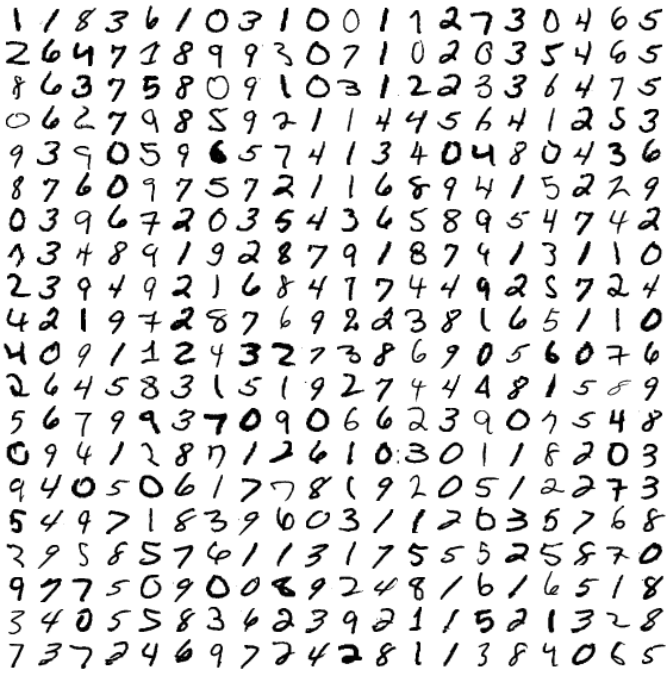
Notes

An embedding is a projection of a signal into a different space that usually aims at keeping all the information. Dimension reduction is a projection into a space of smaller dimension.
A input signal usually has a lot of nuisance in it:

- in computer vision, there can be illumination changes, geometric pose, occlusion, complex background, etc.
- in sound processing, there can be reverberation, ambient noise, features of the microphone, etc.

Prediction requires to go from the raw input signal to a refined representation which no longer depends on these nuisances.

MNIST data-set

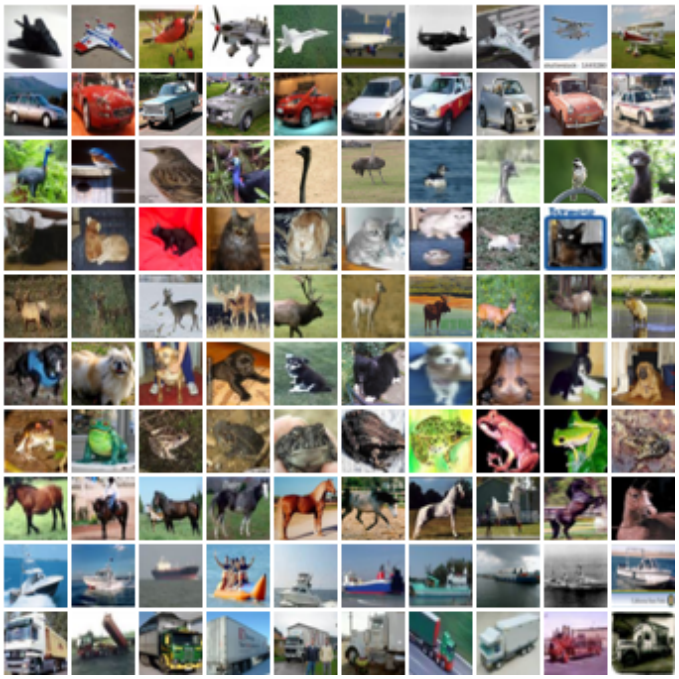


28×28 grayscale images, 60k train samples, 10k test samples.

Notes

MNIST is a data set of grayscale images of digits.
The images are of size 28×28. There are $60,000$ train samples and $10,000$k test samples.

CIFAR10 data-set



32×32 color images, 50k train samples, 10k test samples.

(Krizhevsky, 2009, chap. 3)

Notes

CIFAR10 is a data set of RGB images of size 32×32. There are $50,000$ train samples and $10,000$ test samples. The classes are airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.
This data set is more challenging than MNIST.

Given

$$x_n \in \mathbb{R}^D, \ n = 1, \ldots, N,$$

and a fixed number of clusters $K > 0$, K-means tries to find K "centroids" that span uniformly the training population.

Given a point, the index of its closest centroid is a good coding.

Formally, [Lloyd's algorithm for] K-means (approximately) solves

$$\underset{c_1,\ldots,c_K \in \mathbb{R}^D}{\operatorname{argmin}} \sum_n \min_k \|x_n - c_k\|^2.$$

This is achieved with a random initialization of $c_1^0, \ldots c_K^0$ followed by repeating until convergence:

$$\forall n, k_n^t = \underset{k}{\operatorname{argmin}} \|x_n - c_k^t\| \quad (1)$$

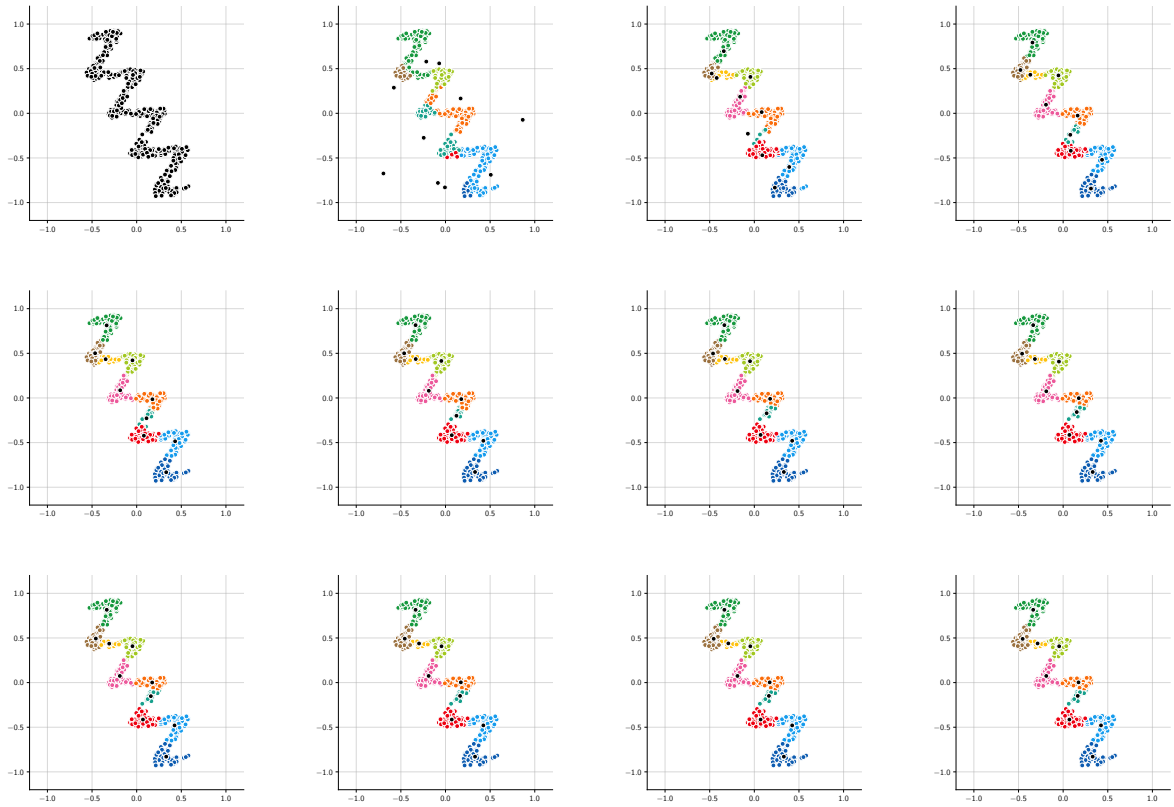
$$\forall k, c_k^{t+1} = \frac{1}{|n : k_n^t = k|} \sum_{n:k_n^t=k} x_n \quad (2)$$

At every iteration, (1) each sample is associated to its closest centroid's cluster, and (2) each centroid is updated to the average of its cluster.

Notes

In the K-means problem, we want to find K centroids $c_1, \ldots, c_K$ such that the sum over all the samples of their distance to the closest centroid is minimal.
The complexity of the K-means problem is NP-hard, so we only have heuristic algorithms to approximate it.
The most used is Lloyd's algorithm, an iterative process which at each step:

- associates each sample to its closest centroid,
- updates every centroid by taking the average of all the samples which have been associated to it.



Notes

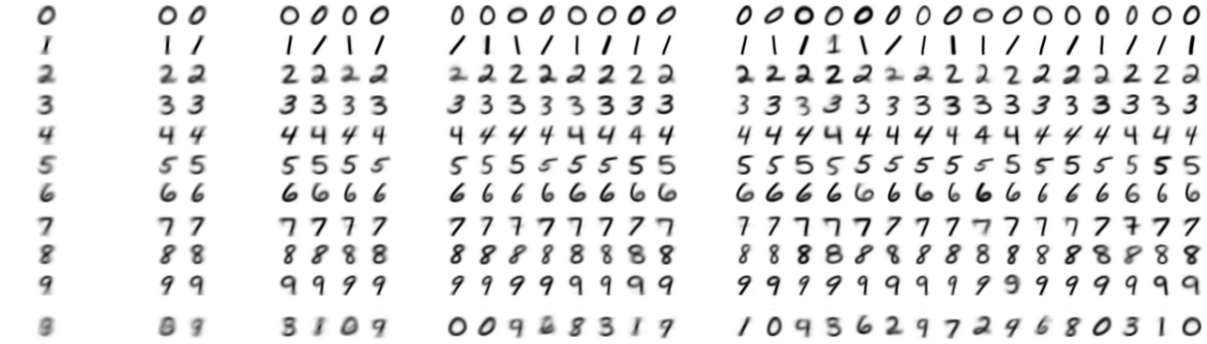
We illustrate the behavior of Lloyd's algorithm on a 2d synthetic toy example.
The top-left scatter plot show all the training points. On the other scatter plots, the black dots are the centroids, and all the samples belonging to a given cluster are depicted with the same color.
Here the centroids are initialized uniformly in $[-1, 1]^2$ which explain why after initialization (second scatter plot of the first row) they are "outside" the training data. Other strategies consists in initializing the centroids with training samples at random, or "carefully" chosen to start with a small distance to minimize.
Over the iterations, the centroids get more and more centered around the training points, and in the end are very well dispatched among the training population.

We can apply that algorithm to images from MNIST ($1 \times 28 \times 28$) or CIFAR10 ($3 \times 32 \times 32$) by considering them as vectors from $\mathbb{R}^{784}$ and $\mathbb{R}^{3072}$ respectively.

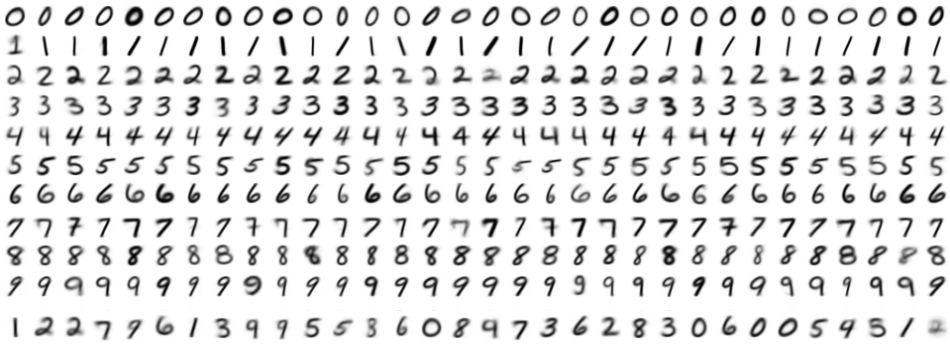
Centroids can similarly be visualized as images, and clustering can be done per-class, or for all the classes mixed.

Notes

To apply the K-means algorithm on images, we represent each image as a vector by reshaping the 28×28 tensor for MNIST or $3 \times 32 \times 32$ for CIFAR10 into a 1d tensor of dimension 784 for MNIST and 3072 for CIFAR10.
Once the centroids are learned, we can visualize them as images by reconstructing the image by reshaping the vector into the image of the right dimension.



$K = 1$ $K = 2$ $K = 4$ $K = 8$ $K = 16$



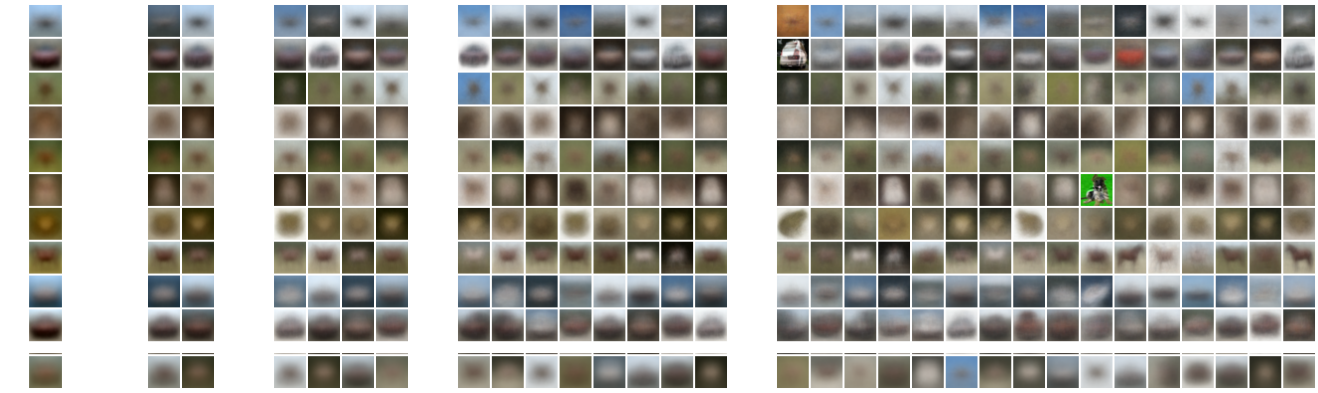
$K = 32$

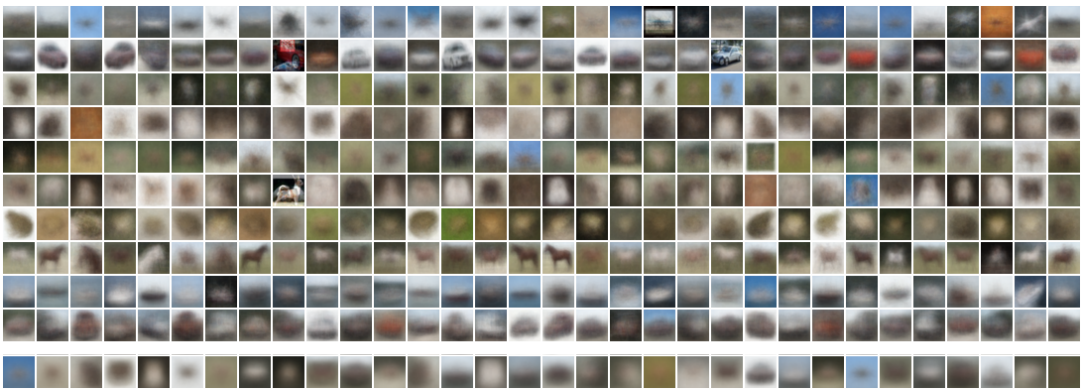
Notes

The images show, for different values of K, the centroids obtained on MNIST for each class separately, and for classes all together (bottom row in each group).
These results are really pleasing:

- for separate classes, the centroids capture different orientations, thickness and morphology of the digit;
- when the algorithm is applied on all the classes together, the classes themselves are dispatched among several centroids. We see that is capture some "pure" classes, like 0, 6, or 2; and similar classes which share common shapes, like 3 and 8, 9 and 1, 9 and 4.

These experiments also show that MNIST is a simple data set, with an nice and easy structure.





Notes

These images are the result of the exact same experiment as before now on CIFAR10. We apply the K-means algorithms on each class separately, as well as on the full data set (last row). The images are the centroids.
The very good behavior observed on MNIST does not hold anymore. Even with 32 centroids, we get meaningless blobs, although for some classes like car and horse, we see a shape of the object. The centroids which depict a clear image are degenerated cases, where there is only one sample in the cluster.

The Principal Component Analysis (PCA) aims also at extracting an information in a L^2 sense. Instead of clusters, it looks for an "affine subspace", i.e. a point and a basis, that spans the data.

Given data-points

$$x_n \in \mathbb{R}^D, \ n = 1, \ldots, N$$

(A) compute the average and center the data

$$\begin{aligned} \bar{x} &= \frac{1}{N} \sum_n x_n \\ \forall n, \ x_n^{(0)} &= x_n - \bar{x} \end{aligned}$$

and then for $t = 1, \ldots, D$,

(B) pick the direction and project the data

$$\begin{aligned} v_t &= \underset{\|v\|=1}{\operatorname{argmax}} \sum_n \left(v \cdot x_n^{(t-1)} \right)^2 \\ \forall n, \ x_n^{(t)} &= x_n^{(t-1)} - \left(v_t \cdot x_n^{(t-1)} \right) v_t. \end{aligned}$$

Although this is a simple way to envision PCA, standard implementations rely on an eigen decomposition. With

$$X = \begin{pmatrix} — & x_1 & — \\ & \vdots & \\ — & x_N & — \end{pmatrix}$$

the centered data points, we have

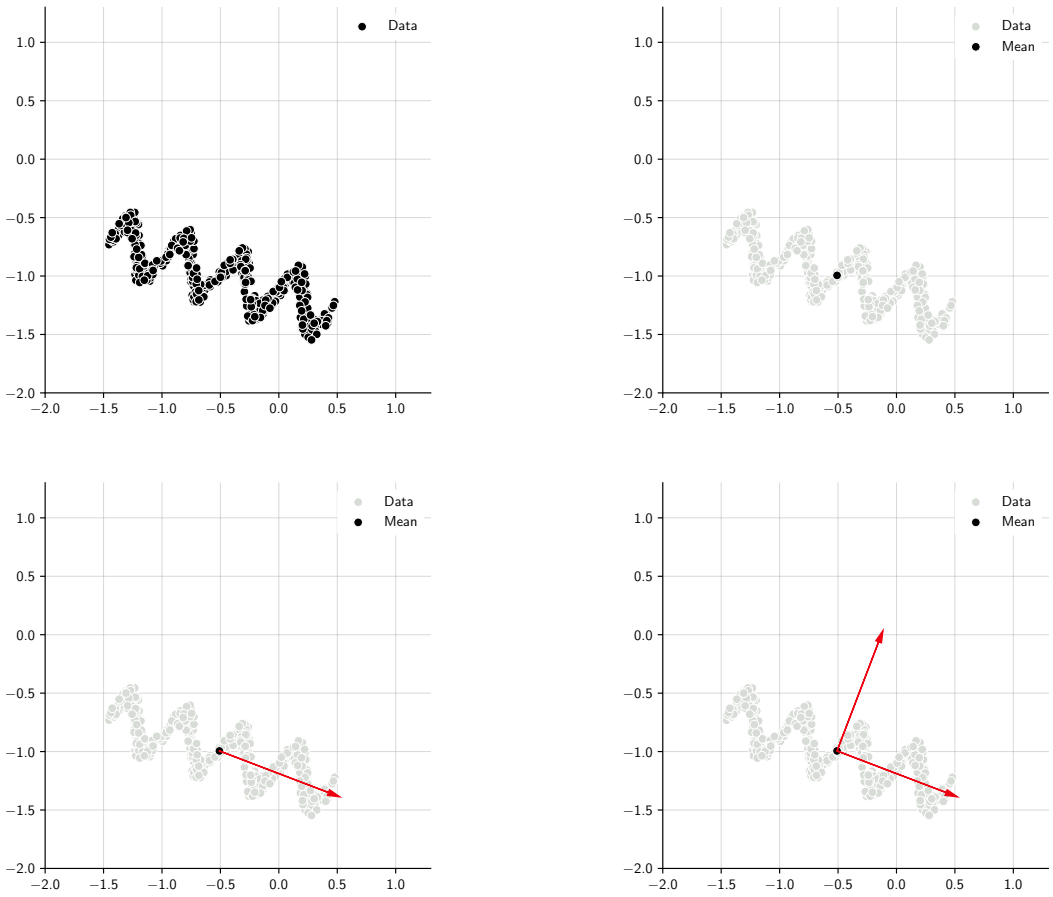
$$\begin{aligned} \sum_n (v \cdot x_n)^2 &= \left\| \begin{pmatrix} v \cdot x_1 \\ \vdots \\ v \cdot x_N \end{pmatrix} \right\|_2^2 \\ &= \left\| v X^\top \right\|_2^2 \\ &= (vX^\top)(vX^\top)^\top \\ &= v(X^\top X)v^\top. \end{aligned}$$

From this we can derive that $v_1, v_2, \ldots, v_D$ are the eigenvectors of $X^\top X$ ranked according to [the absolute values of] their eigenvalues.

Notes

In practice, to compute the PCA basis:

- we center the data by subtracting their mean,
- we compute the eigen decomposition of $X^\top X$ where X with the matrix of the row samples,
- we rank the eigenvectors according to the absolute value of their corresponding eigenvalue,
- v_1 is the first vector of the PCA basis, v_2 the second, etc.



Notes

We illustrate the behavior of PCA on a 2d synthetic toy example. The first basis vector goes in the direction of maximum dispersion, and the second one is orthogonal to the first one.

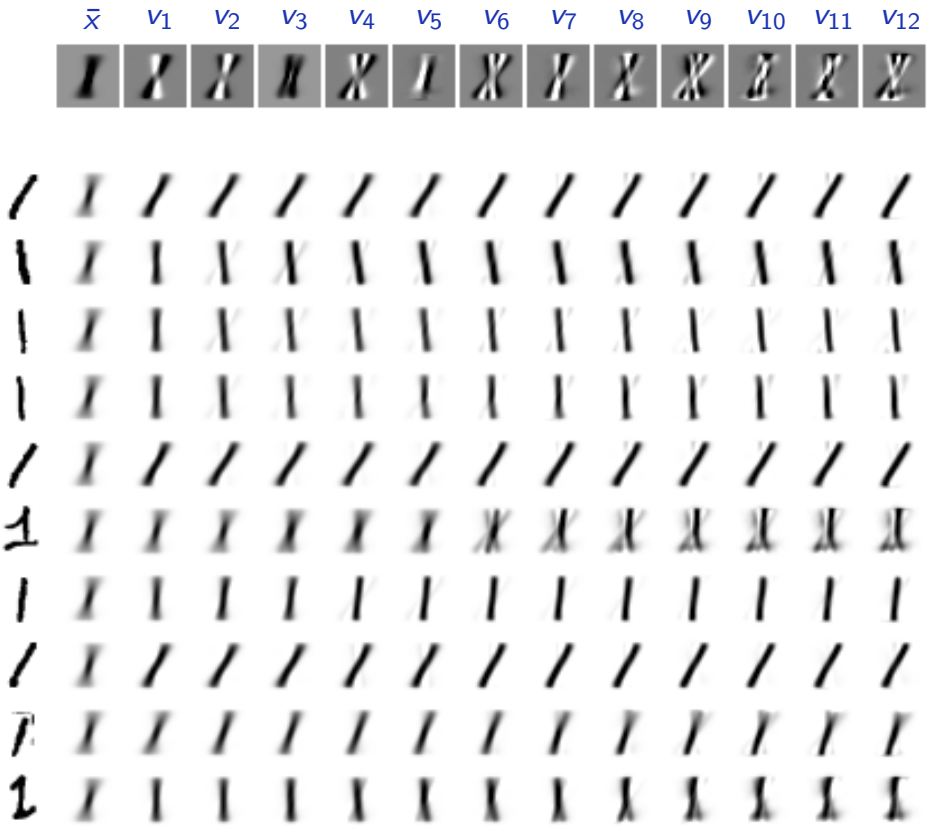
As for K-means, we can apply that algorithm to images from MNIST or CIFAR10 by considering them as vectors.

For any sample x and any T, we can compute a reconstruction using T vectors from the PCA basis, i.e.

$$\bar{x} + \sum_{t=1}^{T} (v_t \cdot (x - \bar{x}))v_t.$$

Notes

T is the number of vectors from the PCA basis that will be used to reconstruct the input sample. The vectors in the PCA basis are ranked in decreasing order of the absolute value of their eigenvalue.
Then, we can compare how close the reconstructed sample is to the original one.

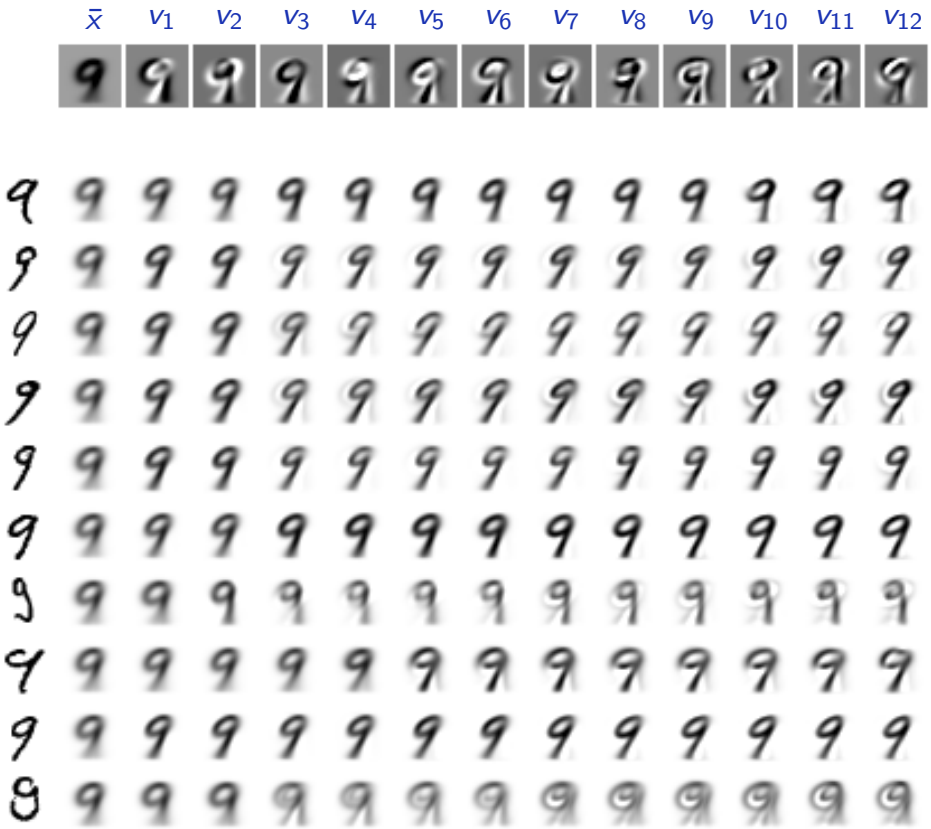


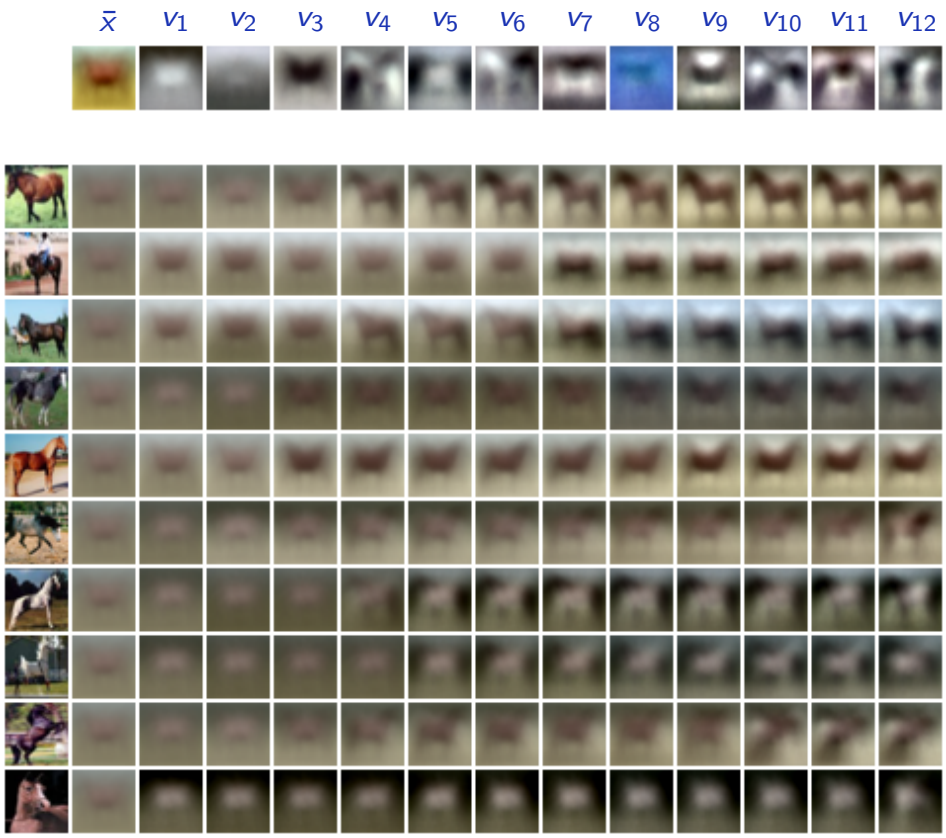
Notes

We show the result of PCA applied to MNIST. On the top image, $\bar{x}$ is the mean of all samples of class 1, the v_t represent the eigenvectors. Gray is for values around 0, white is positive, and black negative.
It appears that the eigenvectors encode small geometric transformations such as rotations, translations, thickening.
The image below shows, for each input sample (first column), its projection in the affine subspace:

- column 1: the original sample,
- column 2: the reconstruction with no basis vector, so only the mean,
- column 3: the reconstruction with 1 basis vector,
- column 4: the reconstruction with 2 basis vectors,
- etc.

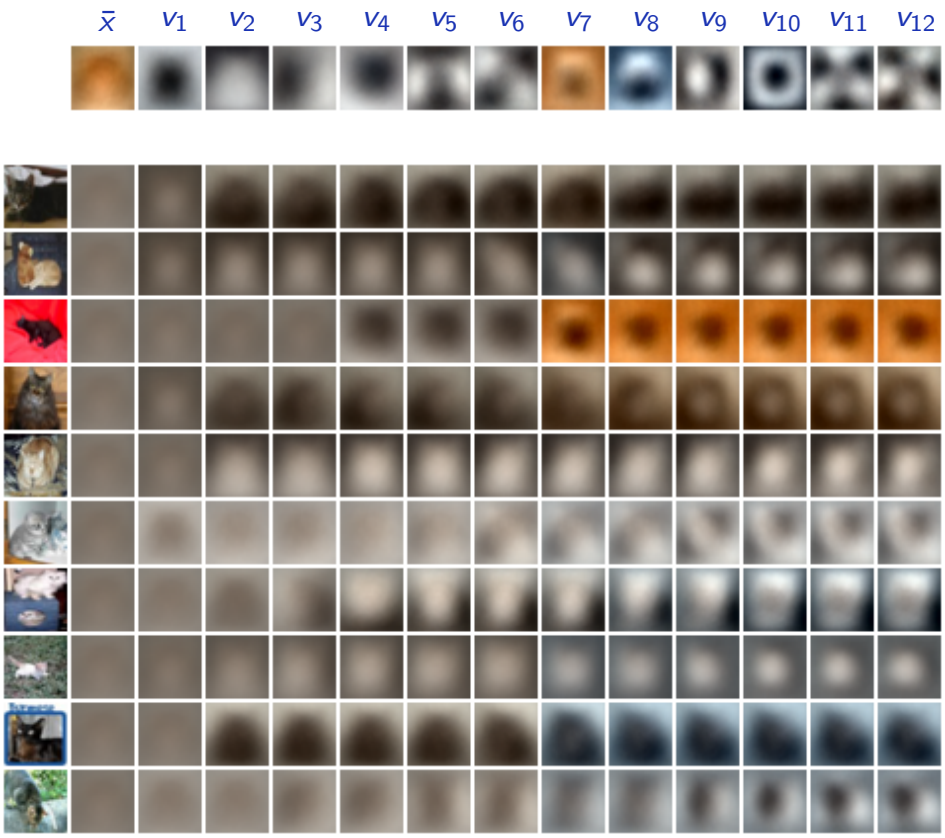
The more we increase the dimension of the space (i.e. with more basis vector), the better we reconstruct the original signal. With twelve vectors, we are able to reconstruct the digit pretty well.


$\bar{x}$
v_1
v_2
v_3
v_4
v_5
v_6
v_7
v_8
v_9
v_{10}
v_{11}
v_{12}



Notes

We now illustrate the behavior of PCA on the CIFAR10 data-set. As before, the top row display the mean of the current class (here "horse") followed by the eigenvectors ranked in decreasing order of absolute value of the eigenvalues.
The eigenvectors also encode some geometric deformations which is visible with the black and white areas, a bit like for Haar filters.
Although we can see that the reconstruction is not as good as in the case of digits, and that it is clear that "strong AI" is not going to be achieved with PCA, the reconstruction does a reasonable job, and in some cases, twelve basis vectors are enough to reconstruct an image with enough information for a human to recognize the animal.



Notes

The class "cat" is a difficult class in the CIFAR10 data set because it exhibits a lot of intra-class variations: cats appears in a a wide range of poses and view points, as opposed to others classes such as "horse" or "frog".
This is reflected here in both the eigenvectors and the reconstructed patterns.

These results show that even crude embeddings capture something meaningful. Changes in pixel intensity as expected, but also deformations in the "indexing" space (i.e. the image plan).

However, translations and deformations damage the representation badly, and "composition" (e.g. object on background) is not handled at all.

These strengths and shortcomings provide an intuitive motivation for "deep neural networks", and the rest of this course.

We would like

- to use many encoding "of these sorts" for small local structures with limited variability,
- have different "channels" for different components,
- process at multiple scales.

Computationally, we would like to deal with large signals and large training sets, so we need to avoid super-linear cost in one or the other.

References

A. Krizhevsky. **Learning multiple layers of features from tiny images**. Master's thesis, Department of Computer Science, University of Toronto, 2009.