Unsupervised Learning - MVA 2017/2018
Homework 1

Victor Busa, Rémi Lespinet

1 Low rank Matrix Completion

The low rank matrix completion (LRMC) has been implemented in python3 using the numpy library.

2 Face completion

We know that the intensity of light reflected on a lambertian surface under a varying light source position (with one source) leaves in a space of dimension 3 :

$$I_D(x, y, z) = L(x, y, z) \cdot N(x, y, z) C I_L$$

where $L(x, y, z)$ is the light direction vector pointing from the point on the surface to the light source, N is the normal of the surface in that point, C is the color of the light source and I_L is the intensity of the light source.

In our case, we have faces under a varying illumination source. These are of course not lambertian surfaces (even if we forget about self shadowing, skin is a very complicated material to model). Still it provides the intuitition that the matrix obtained by stacking the images seen as vectors (vectors of size $192 \times 168 = 32256$, thus forming a matrix of size 32256×10) is low rank.

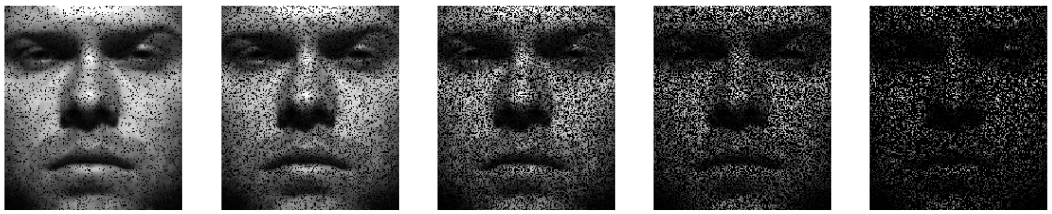
For each pixel we remove the pixel with probability ρ (Bernoulli random variable), and we use this matrix as input in the LRMC algorithm. The figure 1 present the results (showing only the first image of the dataset), for different values of τ and different values of ρ. (The full reconstruction for the 10 images figures for the same values of τ are shown on the appendix, figures 6, 7, 8, 9)

As τ increases, we see that we are able to recover better the image (the MSE decreases and we can see it visually). This is not suprising since the relaxed problem we want to solve look like

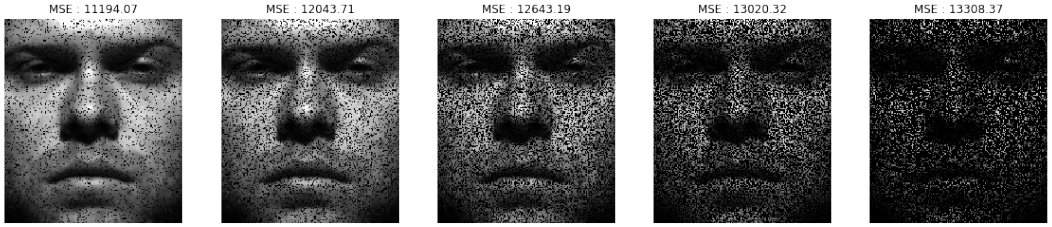
$$\begin{aligned} \underset{A}{\text{minimize}} \quad & \|A\|_* \\ \text{subject to} \quad & P_\Omega(A) = P_\Omega(X) \end{aligned}$$

And the problem we are actually solving with LRMC is

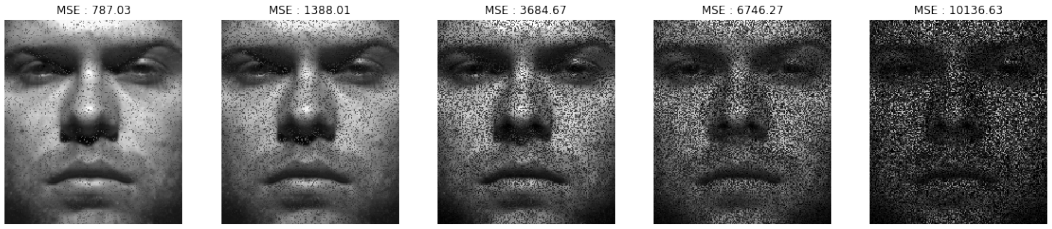
$$\begin{aligned} \underset{A}{\text{minimize}} \quad & \tau\|A\|_* + \frac{1}{2}\|A\|_F^2 \\ \text{subject to} \quad & P_\Omega(A) = P_\Omega(X) \end{aligned} \tag{$\star$}$$



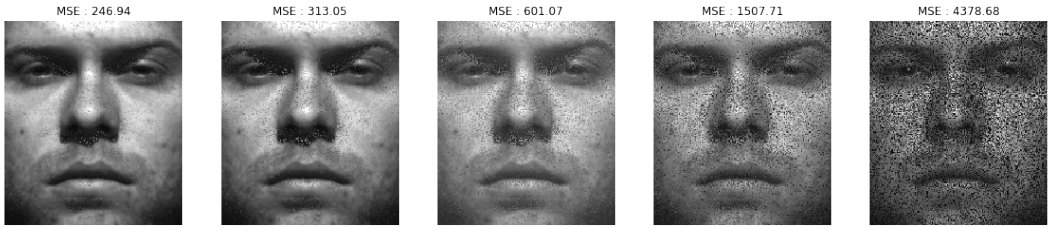
(a) Faces images with (10%, 20%, 40% 60% and 80%) of missing entries



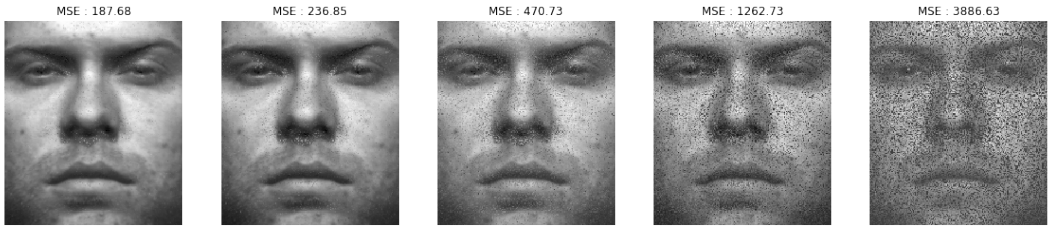
(b) Faces images reconstructed by convex optimization with $\tau = 10^3$



(c) Faces images reconstructed by convex optimization with $\tau = 2 \cdot 10^4$



(d) Faces images reconstructed by convex optimization with $\tau = 4 \cdot 10^5$



(e) Faces images reconstructed by convex optimization with $\tau = 8 \cdot 10^6$

Figure 1: Reconstruction of the first face of the Yale dataset using the LRMC algorithm on 10 images of the set (192×168) for different values of ρ and τ

Hence, the larger τ, the more the solution of LRMC approximates well our original problem. If τ is small, the problem becomes equivalent to

$$\begin{aligned} \underset{A}{\text{minimize}} \quad & \|A\|_F^2 \\ \text{subject to} \quad & P_\Omega(A) = P_\Omega(X) \end{aligned}$$

Which admits the trivial solution

$$A(i,j) = \begin{cases} X(i,j) & \text{if } (i,j) \in \Omega \\ 0 & \text{otherwise} \end{cases}$$

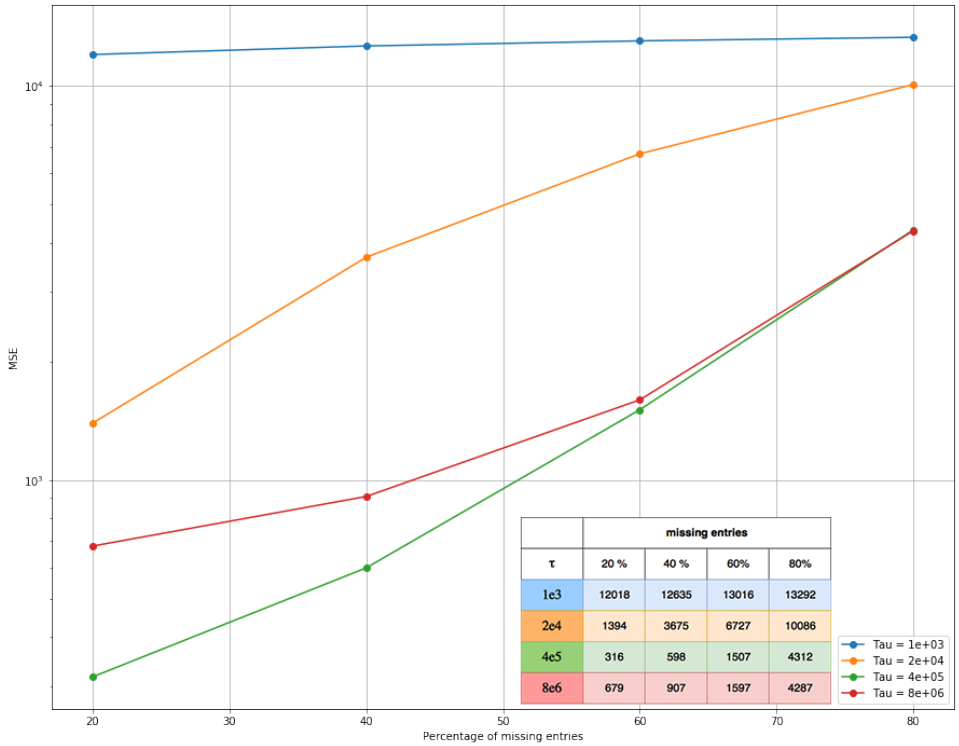


Figure 2: Mean-squared-error for different value of τ function of the percentage of missing entries (for the 10 images (192×168) suggested of the first person in the yale dataset)

The figure 2 represent the observed MSE as a function of the number of missing entries ρ for different values of τ.

The results are good, but looking at the book [1] (figure 3), we see that it has better results for the same values of τ and ρ. To obtain similar results, we took the whole *yaleB20* of the *Extended Yale Face Database B* [2] (64 images), resized all the images by 50%, and ran the exact same procedure with these 64 images of size 96×84. The results obtained are represented in figure 4

The results are better both visually and in term of MSE. This shows that the higher the number of image, the better the algorithm reconstructs (if we forget for the fact that we resized the image for speed reasons). This is intuitive, and theoritically, Here, the expected number of observed entries M is

$$M = (1-\rho)ND$$

Hence the theorem presented in the class can be expressed in the following form : there exist a constant c such that if

$$(1-\rho) \geq c\nu^4 \frac{d}{D} (\log(N))^2$$

then the images are recovered with porbability at least $1 - N^{-3}$

Which means that if we increase D (we add new images), we can take a higher ρ and still verify the equality

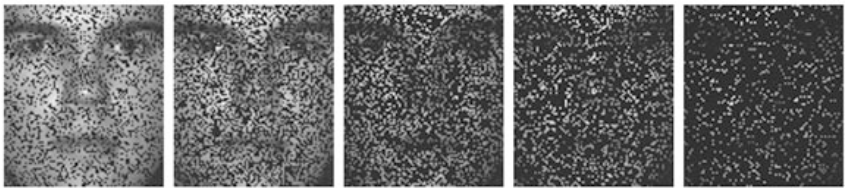
Choice of β

As suggested in the subject, we chose β as

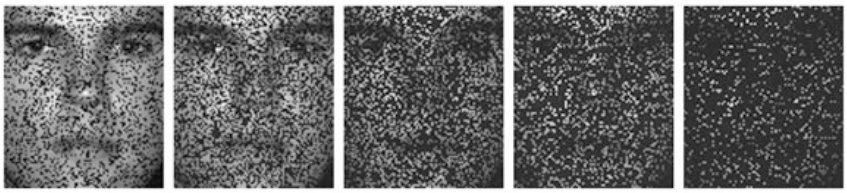
$$\beta = \min\left(2, \frac{ND}{M}\right) = \min\left(2, \frac{1}{1-\rho}\right)$$

This makes sense because the algorithm provably converges [3] to the solution of the minimization problem $(\star)$ if

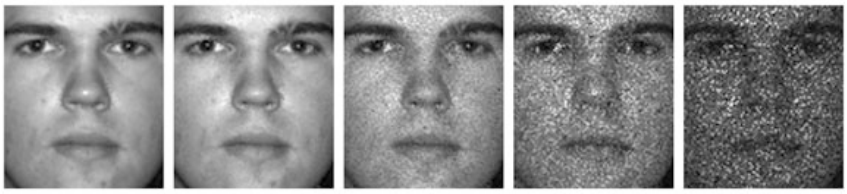
$$0 < \beta < 2$$



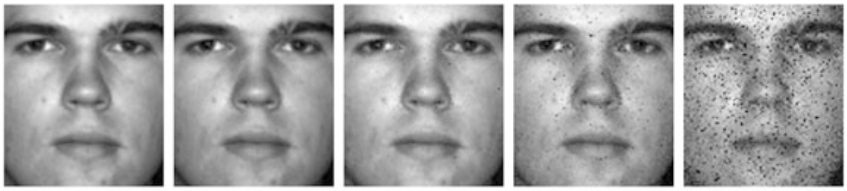
(a) Face images with (30, 50, 70, 80, 90)% percentage of missing entries



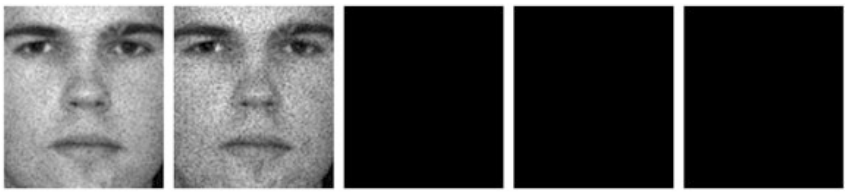
(b) Face images reconstructed by convex optimization with $\tau = 10^3$



(c) Face images reconstructed by convex optimization with $\tau = 2 \times 10^4$

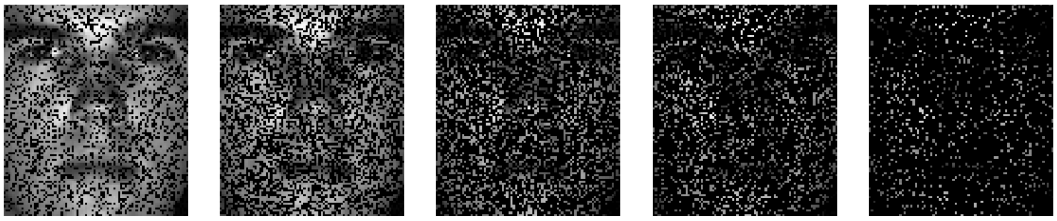


(d) Face images reconstructed by convex optimization with $\tau = 4 \times 10^5$

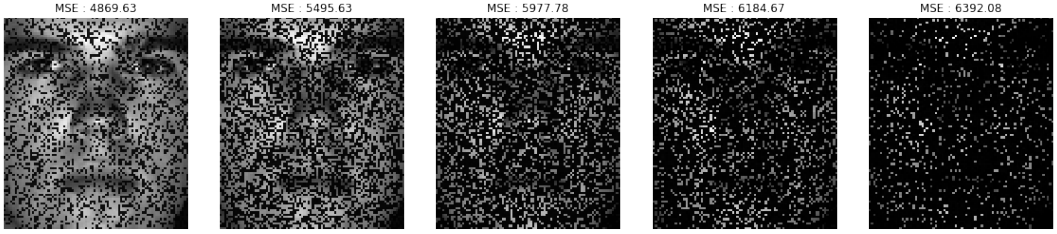


(e) Face images reconstructed by convex optimization with $\tau = 8 \times 10^6$

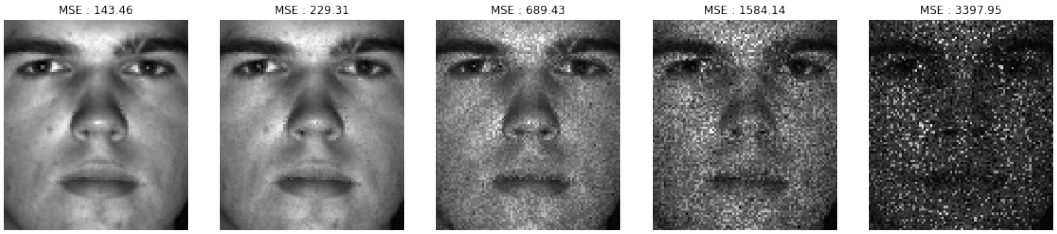
Figure 3: First row: Face images with (30, 50, 70, 80, 90)% of missing entries, second to last rows: Face images reconstructed for τ being respectively (1e3, 2e4, 4e5, 8e6)



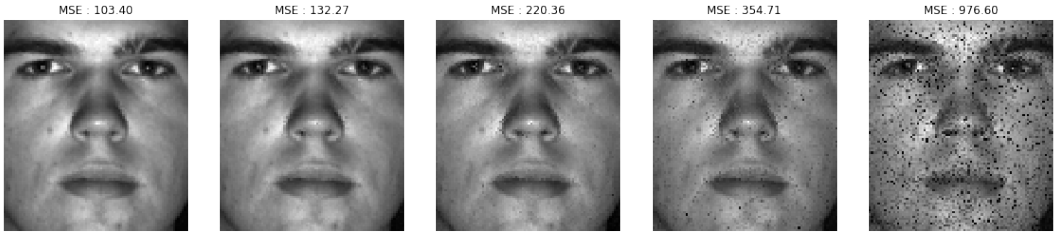
(a) Faces images with (30%, 50%, 70% 80% and 90%) of missing entries



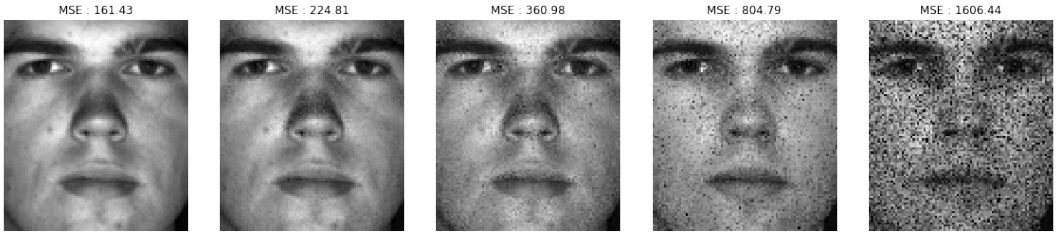
(b) Faces images reconstructed by convex optimization with $\tau = 10^3$



(c) Faces images reconstructed by convex optimization with $\tau = 2 \cdot 10^4$



(d) Faces images reconstructed by convex optimization with $\tau = 4 \cdot 10^5$



(e) Faces images reconstructed by convex optimization with $\tau = 8 \cdot 10^6$

Figure 4: Reconstruction of the same face of the book 3 (on the Yale dataset) using the LRMC algorithm on 64 images (96×84) of this face under different illuminations for different values of ρ and τ

3 Movie Recommendation Grand Challenge

For the Movie recommendation challenge, we want to apply the same algorithm. We build the matrix X such that $X(i, j)$ corresponds to the rating that the user i has given to the movie j.

Intuitively, the fact that the matrix is low rank can be explained by the fact that every user is a combination of a small number of "eigen-users", that is there's a small there's a small number of "behaviors". This assumption is not totally absurd, and it makes sense to try to recover the full matrix of ranking using this method.

This situation is different than the previous one, in which we artifially removed entries in the image, and hence we did know the real values of the missing entries. In this case we do not have access to the real matrice (in that sense, this is a more realistic situation). To compute the MSE, we remove 10% of the known entries, give that as input to LRMC, and look at these 10% entries to compute the MSE. To have more accurate results, we use cross-validation : we randomly create a partition of size 10 of the missing entries and calculate the mean MSE over the elements of the partition.

The results are reported in the table 1

τ	100	500	2500	12500
Horror	5.33	2.72	1.81	1.59
Romance	5.28	2.94	2.09	1.91
Both	5.51	2.83	1.88	1.63

Table 1: Mean square error obtained for the horror and romance categories for different values of τ (obtained with cross-validation by the procedure described above)

As seen in the previous case, increasing the value of τ leads to a lower MSE, but also requires a higher number of iterations. In our experiment, the number of iterations is capped to 1000. For our choice of the norm, we always reach the maximum number of iteration, and never terminate due to the stopping criterion. This might explain the results

For this reason, we have tried to improve the algorithm (see 4).

The figure 5 shows the evolution of the MSE with τ (computed with cross-validation by the method described above).

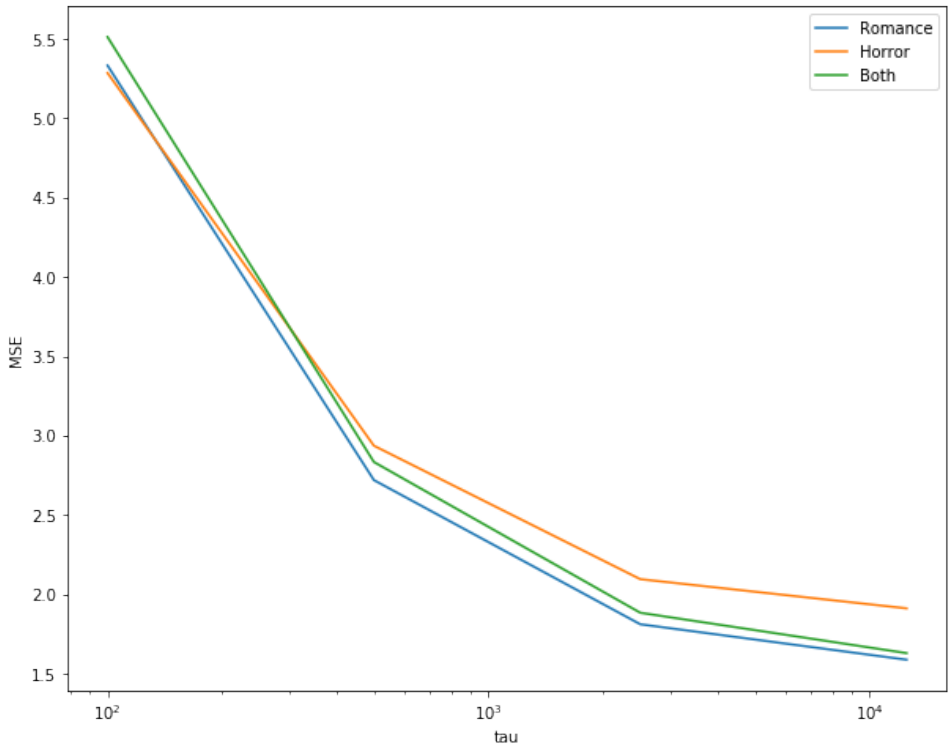


Figure 5: Ouptut of the LRMC algorithm for different value of τ and ρ

4 Ameliorations

4.1 Algorithm speed

In this algorithm, the time performance entirely relies only on the SVD computation. Fortunately, high number of missing entries corresponds to sparse matrices and hence the SVD can be computed more easily. We did the implementation in python, and we could not use this potentential at maximum. Moreover, with the tested stopping criterion, the algorithm was always stopped by the maximum number of iterations, and taking a very high number of iterations lead to some numerical problems. We were very interested in knowing if we could improve the result, so we decided first to use Scikit-CUDA or even TensorFlow to do calculations, but it appeared that not only there is no efficient parallel implementation of the SVD on GPU, but also that the CPU implementation is higly optimized (aka MKL Intel), and hence hard to outperform. We also did a $C++$ implementation of the algorithm, hoping that the SVD would be much faster in this language. We use the *Eigen3* library (not included in the archive) for linear algebra calculations, and the *REDSVD* library to compute the SVD. nfortunately, it appears that our implementation is not really faster, and we did not have the time to improve it before the deadline... We have included this code in the archive anyway.

In this algorithm, we only care about the singular values that are above τ in the singular value decomposition, and since it appears that some SVD algorithms can compute very efficiently the d largest singular values. A proposed method[3] to increase efficiency would be to compute the d largest singular values, and then iteratively increase d up to the point where the lowest singular value returned is less than τ. At this point, it is not necessary to compute the other singular values, since they will all be zero after the thresholding, hence it is only necessary to compute a smaller part of the U and V matrices of the singular value decomposition.

We might also get better results by starting the algorithm with the value of A computed for a lower value of τ.

4.2 Using a decay learning rate

Decreasing the learning rate over time can both lead to better or worse performance. Actually it is very hard to determine how to decrease the learning rate over time.

Here we choose to decrease the learning rate if the mean-squared-error computed after updating A is higher than the mean-squared-error before updating A. In python it corresponds to:

```
fro_prev = mse(X, A, W)
...
fro = mse(X, A, W)
if (fro > fro_prev):
  beta *= 0.9
```

learning rate	run 1	run 2	run3	run4
no decay	45.6	49.6	40.5	39.6
decay by 0.9	54.6	71.1	49	51.7

Table 2: For τ = 4e5 and 40 % of missing entries

learning rate	run 1	run 2	run3	run4
no decay	283	296	330	282
decay by 0.9	235	252	245	268

Table 3: For τ = 2e4 and 80 % of missing entries

As we can see on the tables above, sometimes using a decay learning can lead to better final mean-squared-errors (Table 3), while it can also leads to worst performance as shown in Table 2. On average it appears that tuning the rate by which we want to decay the parameter β is a really difficult task as β both depends on the number of missing entries and on the disposition of these entries in W. Here, decreasing the learning rate doesn't lead to better performance **on average**. The only interest is that it allows the algorithm to stop earlier in its process and thus take less time to terminate. This method is not applicable to the third exercise as the matrix X is already sparse.

5 Conclusion

We both implemented the algorithm entirely, this allowed us to verify our results (the two codes are availables in the `src` directory). When everything was working properly, Victor Busa tried to use TensorFlow for computing the SVD more efficiently, experimented decay on the learning rate. Rémi Lespinet adapted the code in *C++*. Since we obtain the same results for both implementations, figures come from either one or the other. For the redaction of this report, we both wrote a draft version on our side, that we later merged to produce this document.

Figure 6: images reconstruction for **20% of missing entries**. The first row corresponds to the corrupted images, the second to last rows correspond to the reconstruction for τ being {1e3, 2e4, 4e5, 8e6} respectively

Figure 7: images reconstruction for **40% of missing entries**. The first row corresponds to the corrupted images, the second to last rows correspond to the reconstruction for τ being {1e3, 2e4, 4e5, 8e6} respectively

Figure 8: images reconstruction for **60% of missing entries**. The first row corresponds to the corrupted images, the second to last rows correspond to the reconstruction for τ being {1e3, 2e4, 4e5, 8e6} respectively

Figure 9: images reconstruction for **80% of missing entries**. The first row corresponds to the corrupted images, the second to last rows correspond to the reconstruction for τ being {1e3, 2e4, 4e5, 8e6} respectively

References

[1] René Vidal, Yi Ma, S.S Sastry *Generalized Principal Component Analysis*. Springer, 2016.

[2] Athinodoros Georghiades, Peter Belhumeur, and David Kriegman *From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose*. PAMI, 2001.

[3] Jian-Feng Cai, Emmanuel J. Candès, Zuowei Shen *A Singular Value Thresholding Algorithm for Matrix Completion*. IAM Journal on Optimization, vol. 20, no. 4, pp. 1956–1982, 2010.