University of Pisa

Computational Mathematics for Learning and Data Analysis

Irene Pisani[1] and **Sergio Latrofa**[2]

[1] *M.Sc. Computer Science, Artificial Intelligence* - 560104 - i.pisani1@studenti.unipi.it
[2] *M.Sc. Computer Science, Artificial Intelligence* - 640584 - s.latrofa1@studenti.unipi.it

April, 2023

1 Problem setup

(P) is a low rank approximation of a matrix $A \in \mathbb{R}^{m \times n}$, i.e.,

$$\underset{U \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{n \times k}}{\arg\min} \|A - UV^T\|_F$$

(A) is alternating optimization: first take an initial $V = V_0$, and compute $U_1 = \arg\min_U \|A - UV_0^T\|_F$, then use it to compute $V_1 = \arg\min_V \|A - U_1V^T\|_F$, then $U_2 = \arg\min_U \|A - UV_1^T\|_F$, and so on, until (hopefully) convergence. Use QR factorization to solve these sub-problems.

1.1 Introduction and Optimal Solution

Low rank approximation has its optimal solution provided by the truncated Singular Value Decomposition, as stated by the Eckart-Young-Mirsky Theorem (both for the two norm and for the Frobenious norm) [1]. We will refer to such optimal approximation using $A^* = U\Sigma_k V^T$. Given an $A \in \mathbb{R}^{m \times n}$ matrix, SVD has a computational cost of at least $O(mn^2)$ floating point operations. Basic assumption concerns the dimensions of A, having $m >> n$, hence being tall thin, or more in general "strongly-rectangular" (without loss of generality if $n >> m$ the problem is the same but transposed). The aim of this project is to study and test the proposed iterative approach: the QR-ALS (alternate least squares), in order to provide alternative solution, spending hopefully less in terms of time and/or memory, without a significant loss in terms of precision, so that it could be used for LSA, PCA or other low rank approximation based algorithms.

1.2 Internal Notation

We redefine the notation for the two sub-problems; considering a generic step s of the alternate optimization:

$$U_s = \underset{U}{\arg\min} \|A - UV_{s-1}^T\|_F \tag{1}$$

$$V_s = \arg\min_V \|A - U_s V^T\|_F \tag{2}$$

We also assume, for the very first iteration ($s = 1$), the V_0 matrix to be known (for example being composed of random values). A study on the impact of such initial values will be part of the empirical assessment of the algorithm.

We will also use the two points symbol (:) within the matrix element index notation, in order to refer to the whole i^{th} row $M_{i,:}$ or to the whole j^{th} column $M_{:,j}$ of any M matrix.

This choice also implies that, given a transpose operator, $M^T_{i,:} = M_{:,i}$ and $M^T_{:,j} = M_{j,:}$. We will rewrite the transpose operator when using this row/column notation to stress their belonging to the matrix of origin and hopefully to highlight shapes, properties and possible operations.

Finally, we will refer to the sub-problems with the term step (whenever one factor is optimized), and to subsequent execution of two of them (1) and (2) with the term iteration (the alternate optimization of all the factors).

1.3 Sub-problems as Least Squares Problems

We can express the Frobenius norm that we want to minimize as:

$$\|A - UV^T\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{i,j} - U_{i,:}V^T_{:,j})^2}$$

and being $(a_{ij} - U_{i,:}V^T_{:,j})^2 \ = (U_{i,:}V^T_{:,j} - a_{i,j})^2$, we can claim that:

$$\|A - UV^T\|_F = \|UV^T - A\|_F = \|VU^T - A^T\|_F$$

Focusing on the transposed problem, we can see that (1) can be expressed as m LLS involving the columns of U^T and A^T, :

$$VU^T_{:,i} = A^T_{:,i} \Leftrightarrow VU_{i,:} = A_{i,:} \quad \forall i \in [1...m]$$

Having $V \in \mathbb{R}^{n\times k}$, $U_{i,:} \in \mathbb{R}^k$ and $A_{i,:} \in \mathbb{R}^n$.

Let's now focus on (2), looking at the original problem formulation, we have n LLS involving columns of V^T (or just rows of V) and rows of A^T (columns of A).

$$UV^T_{:,j} = A_{:,j} \Leftrightarrow UV_{j,:} = A_{:,j} \quad \forall j \in [1...n]$$

Having $U \in \mathbb{R}^{m\times k}$, $V_{j,:} \in \mathbb{R}^k$ and $A_{:,j} \in \mathbb{R}^m$.

1.4 QR Factorization as LLSs solver

QR Factorization (or decomposition) allows to represent a generic matrix $M \in \mathbb{R}^{m \times n}$ as a product of a square orthogonal matrix $Q_M \in \mathbb{R}^{m \times m}$ and an upper triangular matrix $R_M \in \mathbb{R}^{m \times n}$. When used to solve LLS, both Q and R matrices can be used in their thin versions: $Q_{1(M)} \in \mathbb{R}^{m \times n}$ and $R_{1(M)} \in \mathbb{R}^{n \times n}$.

So given the redefined sub-problems, we can express rows of the solution of (2) as

$$R_{1(U)} V_{j,:} = Q_{1(U)}^T A_{:,j} \tag{3}$$

Where the QR factorization of U_s can be computed only once for the whole sub-problem. Expressing all the n LLS solutions as the entire V^T matrix, we would obtain:

$$R_{1(U)} V^T = Q_{1(U)}^T A \tag{4}$$

Operational cost required to solve Eq. 4 is given by the cost of solving of n LLS using QR factorization: $n(mk + k^2)$ where mk refers to the cost of the multiplication of $Q_{1(U)}^T$ with a single column of A, and k^2 refers to the cost of back-substitution involving $R_{1(U)}$.

With the same identical reasoning, for (1) we would have $VU^T = A^T$, hence:

$$R_{1(V)} U_{i,:} = Q_{1(V)}^T A_{i,:} \tag{5}$$

$$R_{1(V)} U^T = Q_{1(V)}^T A^T \tag{6}$$

Also in this case, using the QR (computed only once as well) to solve the m LLS would correspond to $m(nk+k^2)$ where nk refers to the cost of the multiplication of $Q_{1(V)}^T$ with a single row of A, and k^2 refers to the cost of back-substitution involving $R_{1(V)}$. As stated initially, m is expected to be the largest of the two dimensions of input matrix A, hence, computing m LLS solutions may be not so efficient. Alternatively, we may prefer computing separately the $R_{1(V)}^{-1}$ matrix, and then compute the entire U as a matrix product:

$$U^T = R_{1(V)}^{-1} Q_{1(V)}^T A^T$$

$$U = A(Q_{1(V)} R_{1(V)}^{-T}) \tag{7}$$

The matrix product $B = Q_{1(V)} R_{1(V)}^{-T}$ can be computed directly inverting $R_{1(V)}$ with respect to the rows of $Q_{1(V)}$, solving $BR_{1(V)}^T = Q_1(V)$ as n linear systems. Such computation would cost nk^2, with the AB product costing mnk, for a total of $n(mk + k^2)$, as well as sub-problem (2). Notice that computing the inverse explicitly would change complexity to $mk(n + k) + 1/3k^3$ flops. The inversion cost would just be added to the cost of the LLS solution, worsening it.

1.5 Regularization

To enforce matrices U and V to stay sparse and low in terms of norm, we can apply regularization to the various LLS , using ridge regression (Thikonov regularization)[2]. In particular we have to fix a λ hyper-parameter (or even a λ_V, λ_U couple) to weight a penalty term related to the norm of the fixed matrix.

Once the regularized version of (P) is considered, we have to appropriately redefine the whole problem as:

$$\underset{U\in\mathbb{R}^{m\times k},V\in\mathbb{R}^{n\times k}}{\arg\min} \|A - UV^T\|_F + \lambda_U\|U\|_F + \lambda_V\|V\|_F \tag{8}$$

it should be noted that this reformulation (8) can be seen as a more general form of the previous not regularized one, where the two penalty terms are set to 0 when $\lambda_U = \lambda_V = 0$.

We would reformulate the two sub-problems as well adding the penalty terms: $(1)_{reg}$ and $(2)_{reg}$. They can both be solved using the same QR based approach of their not-regularized formulation, obtaining a sparser solution, lower in terms of norm.

$$(1)_{reg} : \min_{U\in\mathbb{R}^{m\times k}} \|A - UV^T\|_F \ + \lambda_U\|U\|_F \tag{9}$$

$$\min_{U_{i,:}\in\mathbb{R}^k} \| \begin{bmatrix} V \\ \lambda_U I \end{bmatrix} U_{i,:} - \begin{bmatrix} A_{i,:} \\ 0 \end{bmatrix} \|_F \forall i \in [1...m]$$

Where $I \in \mathbb{R}^{k\times k}$, the expanded $\tilde{V} \in \mathbb{R}^{(n+k)\times k}$ and $\tilde{A}_{i,:} \in \mathbb{R}^{(n+k)}$. Solutions can be still obtained computing QR of $\tilde{V}$.

Note that adding Thikonov regularization increase the computational complexity of the LLS: the factor $n \leftarrow n + k$. Thus, keep using the convenient order of Eq. 7:

$$V^T = \tilde{A}(Q_{1(\tilde{U})}R^{-T}_{1(\tilde{U})})$$

the cost of the matrix construction grows to $k(n + k)(k + m)$ flops and at the same time, the cost of computing QR factorization of $\tilde{V}$ increase as well up to $2(n + k)k^2$ flops as well.

$$(2)_{reg} : \min_{V\in\mathbb{R}^{n\times k}} \|A - UV^T\|_F \ + \lambda_V\|V\|_F \tag{10}$$

$$\min_{V_{j,:}\in\mathbb{R}^k} \| \begin{bmatrix} U \\ \lambda_V I \end{bmatrix} V_{j,:} - \begin{bmatrix} A_{:,j} \\ 0 \end{bmatrix} \|_F \forall j \in [1...n]$$

Where $I \in \mathbb{R}^{k\times k}$, the expanded $\tilde{U} \in \mathbb{R}^{(m+k)\times k}$ and $\tilde{A}_{:,j} \in \mathbb{R}^{(m+k)}$. Solutions can be still obtained computing QR (of $\tilde{U}$).

Also in this case adding Thikonov regularization increase the computational complexity of the LLS: the factor $m \leftarrow m + k$. Thus, the cost grows to $nk(m + 2k)$ flops. At the same time, the cost of computing QR factorization of $\tilde{V}$ increase as well up to $2(m + k)k^2 - 2/3k^3$ flops.

1.6 Stopping criteria

The algorithm is supposed to iterate (1) and (2) for a given number of "iterations". Whenever A^* is not known, idea is to stop when the amount of "correction" that each iteration add is not significant anymore. So we fix a ϵ hyper-parameter and compare it to the variation in term of norm produced through the last step. If such threshold is reached, the iteration is stopped. We will keep looping until:

$$\epsilon > \frac{\|U_{s-1}V_{s-1}^T - U_sV_s^T\|_F}{\|U_sV_s^T\|_F} \tag{11}$$

An alternative heuristic to avoid the $O(mnk)$ matrix product, thought for large scale application, may be to only consider the gap with:

$$\xi > \frac{\|R_{1(U)_{s-1}}R_{1(V)_{s-1}}^T - R_{1(U)_s}R_{1(V)_s}^T\|_F}{\|R_{1(U)_s}R_{1(V)_s}^T\|_F} \tag{12}$$

ignoring the two orthogonal matrices $Q_{1(U)}$ and $Q_{1(V)}^T$ (the singular values of A are "contained" in the R matrices). Norm of the heuristic product tends to coincide with actual norm of UV^T (fig. 2). In particular, all the singular values of A are absorbed by the R matrices product, being the Q matrices orthogonal. Such last consideration will also allow us to use algorithm intermediate results to compute and approximation of a k-truncated SVD (6.1).

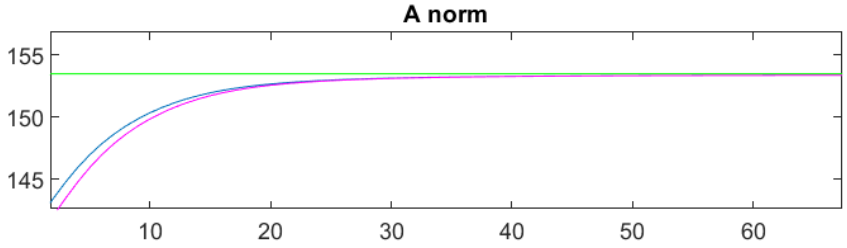


Figure 1: Behaviour of norms through iterations

Figure 2: Frobenius norm of original A matrix (green), norm of the UV^T product (blue), norm of the $R_UR_V^T$ product (red) through iterations.

1.7 Algorithm Pseudo-code

Algorithm 1 Low rank approximation: QR-Alternate Least Square (QR-ALS)

Require: $A \in \mathbb{R}^{m \times n}$
  Initialize $V_0 \in \mathbb{R}^{n \times k}$
  Initialize $converged \leftarrow False$
  $V \leftarrow V_0$
  while $\neg converged$ **do**
    $Q_V, R_V \leftarrow \text{THINQR}(V)$
    Update $U \leftarrow A(Q_V(R_V^{-1})^T)$
    $Q_U, R_U \leftarrow \text{THINQR}(U)$
    Update $V^T \leftarrow R_U^{-1}(Q_U^T A)$
    $converged \leftarrow \text{CHECKCONVERGENCE}(U, V)$
  end while

2 Related Works

Alternating optimization (AO) is generally defined as an iterative procedure for minimizing (or maximizing) the function $f(x) = f(X_1, X_2, \ldots, X_t)$ jointly over all variables by alternating restricted minimization over the individual subsets of variables $X_1, \ldots, X_t$. Under reasonable assumption on the objective function all the limit point of a sequence generated with AO approach are critical points[6]. AO is known to be locally, q-linearly convergent for the special case $t = 2$ [5], and globally convergent in some particular unconstrained cases [3].

Going into practical details, the least square based alternate optimization (LS-AO) technique is widely used in literature for solving several linear algebra problems such as matrix completion, matrix sensing and rank-constrained matrix recovery, as well as tensor low-rank approximation.

Matrix completion problem, that was described accurately by [9], was often handle with alternate optimization and a modified version of ALS by [14] and [12]. Given the low-rank target matrix written in a bi-linear form, literature convey that each alternating step in isolation is convex and tractable despite the problem being non-convex. These proposed alternating minimization algorithm have also theoretical guarantees on their convergence. [14] [12]

Also matrix sensing problem was solved on ALS method: it has been showed that ALS with random initialization converges to the true solution with ϵ-accuracy with finite iterations; this means that the trajectory of the ALS iterates only depends very mildly on certain entries of the random matrices [21].

LLS-based alternate optimization approach was exploited for proving the effectiveness of the incremental-rank Power factorization algorithm in the context of recovering low-rank matrices

satisfying a set of linear equality constraints. Despite the non-convexity introduced by the low-rank parameterization, the cost function is guaranteed to converge. In general, authors claim that iterates themselves are not guaranteed to converge, particularly in the case when there is sustained rank-deficiency in the LLS problems, but they showed this is not generally an issue during empirical experiments. [8]

At the same time, ALS and modified and improved version of ALS, are one of the most popular approach to tensor low rank approximation [11] and it is known that for almost all tensors, the iterates generated by the alternating least squares method for the rank-one approximation converge globally [13].

Since AO is also known as block non linear Gauss Seidel iteration, as stated in [6], in section 3.4 we have exploits theorem and conclusion given by Grippo and Sciandrone [4] in order to give a clear evidence that the sequence generate by our QR-ALS for matrix low rank approximation converges to a critical points of the problems.

Then based on a previous work of [17] we have to mention a still open problem in the global analysis of optimization on the low-rank matrix manifold: there is no guarantee that the limit point of an AO iterative sequence will converge to a rank-k ground truth instead of being stuck at some lower-rank spurious critical points.

3 Expectation over algorithm

3.1 Quadratic Formulation

Given the problem (P),the objective function can be rewritten in its quadratic formulation:

$$\|A - UV^T\|_F^2 = (A - UV^T)^T(A - UV^T) = A^TA - A^TUV^T - VU^TA + VU^TUV^T$$

Being $VU^TA = A^TUV^T$ we can obtain the quadratic formulation with respect to V:

$$f(V|A) = VU^TUV^T - 2A^TVU^T + A^TA = UV^TVU^T - 2UV^TA^T + A^TA \tag{13}$$

We can obtain sub-problem (2) just differentiation this formulation with respect to V:

$$\nabla_V f(V|A) = 2U^TUV^T - 2U^TA \tag{14}$$

To obtain the quadratic formulation with respect to U, we should consider the transposed norm as in 1.3:

$$\|A^T - VU^T\|_F^2 = (A^T - VU^T)^T(A^T - VU^T) = AA^T - AVU^T - UV^TA^T + UV^TVU^T$$

$$f(U|A) = UV^TVU^T - 2UV^TA^T - AA^T \tag{15}$$

Differentiating with respect to U we can obtain (1):

$$\nabla_U f(U|A) = 2V^TVU^T - 2V^TA^T \tag{16}$$

3.2 Sub-problems interpretation

We can look to our sub-problems (1) and (2) as matrix-wise linear least squares of type $\min_X \|AX = B\|_F$, where X and B are "vectors of vectors". The existance and uniqueness of solutions (or equivalently the convexity of the sub-problems) is ensured if the two Hessian matrices are positive semi-definite:

$$\nabla^2_U f(U|A) = 2U^TU \tag{17}$$

$$\nabla^2_V f(V|A) = 2V^TV \tag{18}$$

We can derive the two normal equations from 14 and 16 :

$$\nabla_U f_1(U|A) = 0 \Rightarrow U_s = (V^TV)^{-1}V^TA^T = V^+A^T$$
$$\nabla_V f_2(V|A) = 0 \Rightarrow V_s = (U^TU)^{-1}U^TA = U^+A$$

which just corresponds to what expressed (through the QR) in equations 7 and 4:

$$V^+ = (V^TV)^{-1}V^T$$
$$\Rightarrow I = (V^TV)^{-1}V^TV = (V^TV)^{-1}R_V^TQ_V^TQ_VR_V \Rightarrow$$
$$\Rightarrow R_V^{-1}Q_V^T = (V^TV)^{-1}R_V^TQ_V^T = (V^TV)^{-1}V^T \Rightarrow$$
$$\Rightarrow V^+ = R_V^{-1}Q_V^T \Leftrightarrow (V^+)^T = Q_VR_V^{-T}$$

So assuming the Hessians always being positive semi-definite, we can figure out each optimization step as a jump from one global minima to the other, and hence, given the same pre-condition, the optimization path being unique and fully deterministic.

3.3 Rank propagation

Unless an ill posed problem, we can safely assume $rank(A) > k$ and expect every V_s and U_s matrix to be full column rank given a random full column rank initial V_0.

Such claim is more empirical than theoretical, intermediate U_s and V_s can be non unique in case of exact row orthogonality (1) in the right case or column orthogonality in the left one (2), obtaining not full column rank results. Anyway, although such limit cases can be hand

constructed, they are very hard to occur from a probabilistic point of view when working with an initial random full column rank V_0.

Intuitively, it's easy to think the product of rank k matrices (k linear independent vectors) expressed in equations 7 and 4 is itself a rank k matrix, and that such "rank propagation" would never stop along the whole optimization path.

Assuming $rank(A) > k$, we can arbitrary choose a full column rank V_0 so that $R^{-1}_{1(V_0)}$ exists and is full column rank and $rank(Q_{1(V_0)}) = k$ because of orthogonality.

Whenever the full column rankness of V_0 does not hold, regularization is expected to be helpful, adding k new linearly independent rows to the rank deficient matrix and hence making the new LLS convex.

3.4 Approximation error and convergence

Algorithm is expected, at each step, to incrementally lower the residual of respectively the U given V and V given U, both with respect to the target A. Rank constraint is satisfied by construction given the k-elements dimension. Solving the n and m LLSs at each alternate optimization step ensures that the matrix U_s and V_s comes closer (or at least remains at the same distance), from (rows or columns of the) target matrix A. Given a certain step s, for the two sub-problems s_1 and s_2 and the "local" approximation $A_{s_1} = U_s V^T_{s-1}$ and $A_{s_2} = U_s V^T_s$, by LLS definition we have that:

$$... \geq \|A - A_{(s-1)_2}\|_F \geq \|A - A_{s_1}\|_F \geq \|A - A_{s_2}\|_F \geq \|A - A_{(s+1)_1}\|_F \geq ...$$

Hence being the step-wise residual monotonically decreasing, each A_s approximation would be even more precise. Such sequence denotes an upper bound for the loss of our optimization algorithm. The lower bound clearly corresponds to the optimal solution A^*, and as proved in the Eckart-Young theorem, better low rank approximation can not exist. Such upper bound progressively shrinks, with the amount of correction between each iteration becoming progressively lower, converging at a fixed point at infinite:

$$\lim_{s\to\infty}(\|A - A_{s-1}\|_F - \|A - A_s\|_F) = 0$$

$$\lim_{s\to\infty} A_s = \hat{U}\hat{V}^T \ |:$$

$$\hat{U} = \min_U \|A - U\hat{V}^T\|_F \tag{19}$$

$$\hat{V} = \min_V \|A - \hat{U}V^T\|_F \tag{20}$$

$$\nexists \hat{V}' \neq \hat{V} \ |: \|A - \hat{U}(\hat{V}')^T\|_F < \|A - \hat{U}\hat{V}^T\|_F \tag{21}$$

$$\nexists \hat{U}' \neq \hat{U} \ |: \|A - \hat{U}'\hat{V}^T\|_F < \|A - \hat{U}\hat{V}^T\|_F \tag{22}$$

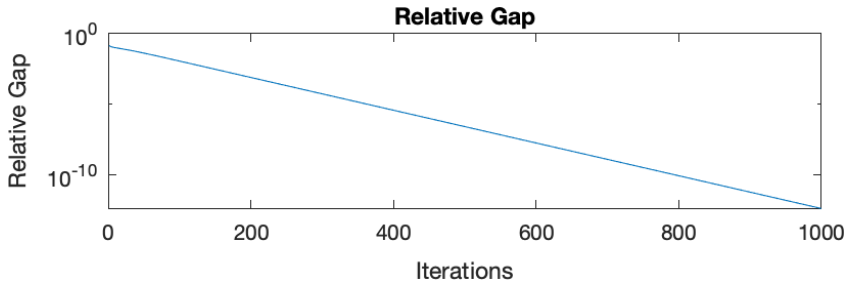


Figure 3: Logarithmic curve of relative gap among iterations for a generic low rank approximation problem (P): convergence path appears to be linear, according to the q-linear speed claimed in the various papers.

It's hard to theoretically prove such point to always be the optimal solution: local minima of the problem (P) exist and corresponds to all the versions of the truncated SVD with a number of remaining singular values lower than k: spurious saddle point of the low rank approximation manifold[17](for more evidence see section 6.7).

Anyway, several research works referring to the ALS least squares for similar problems (i.e. matrix sensing or matrix completion) claim that, though it's not well understood how, the alternate optimization is able to avoid saddle points when using random initialization V_0 [19](Th. 1, Page 5) [12](Th.2, Page 667) [14](Remark 2, Page 3372).

So, assuming limit cases not occurring, we can claim that our algorithm is a 2 Block Gauss-Seidel optimization method and hence has the same convergence properties. In fact, we can consider our loss function as

$$f(x) = f(x_1, x_2) = \|A - x_1 x_2^T\|_F$$

where

$$x_1 = [u_{1,:}, u_{2,:}..., u_{m,:}] \in \mathbb{R}^{m \times k}$$
$$x_2 = [v_{1,:}, v_{2,:}..., v_{n,:}] \in \mathbb{R}^{n \times k}$$

so that

$$x = [x_1, x_2] \in \mathbb{R}^{(m+n) \times k}$$

In (1) the x_2 block is fixed and x_1 is optimized, while in (2) it is fixed and x_1 is optimized as well. Considering intermediate solution full column rank matrices (as discussed before), we already know that (1) and (2) are convex sub-problems with exact solutions. So such alternate optimization loop generates a sequence $\{x^k\}$ where at each k we have a couple (x_1^k, x_2^k). According to [4](Corollary 2, Page 131) we can claim that such series is guaranteed to converge q-linearly (q stands for quotient, hence it is linear but scaled by a factor $0 < q < 1$) to a critical point of (P) with all "valid" (a.k.a full column rank) initialization.

3.5 Impact of regularization

The proposed regularization approach is expected to encourage sparsity within the U and V matrices, allowing to abstract on noisy entries of A. Additionally, expanding the factorized matrix with the λI block,the convexity of the sub-problem is guaranteed, hence eventual rank deficiency cases can be dealt without issues, starting from the V_0 matrix (5.4) .

When regularization is added, the algorithm objective changes as in 8 and so a new optimal solution A^*_λ must be identified. The Eckart-Young theorem does not help anymore, and the problem is still non-convex being the formulation bi-linear. For sure we know that this new optimal solution is generally "less accurate" than the un-regularized one:

$$\|A - A^*\| \leq \|A - A^*_\lambda\|$$

and we also know that the loss cannot be negative, hence it is lower bounded as well. So, a minima of the problem should be founded using a "classical" optimization algorithm.

Anyway, for what concern the convergence, the Gauss Seidel setup persists (we just added new constraints related to U and V norms), so we expect the rate of convergence to keep being q-linear, with some reasonable variation of the q factor given the different nature of $(P)_{reg}$ [4].

In general it's easy to intuitively figure out that regularization moves up the lower bound of the problem in terms of error (distance of any possible solution $U_i V_i^T$ from $A^* = U\Sigma_k V^T$), so the final approximation error is expected to be greater than the one without regularization, while the corresponding norms of U and V are expected to be lowered.

3.6 Computation cost

Computational effort of each sub-problems is related to two main steps: computing the QR and solving the involved LLS.

For what concerns the QR, both sub-problems (1) and (2) can be redirect to the tall-thin case (In (2) matrix $V \in \mathbb{R}^{k \times n}$ is taken transposed, having $n \gg k$). This settings allow us to exploit ThinQR to save operational cost. Clearly, the more k is close to n (assumed to be the smallest dimension), the worst the QR computation cost would become, going into the almost square case.

Complexity of the LLS has been already discussed in section 1.4, with the inverse optimization for sub-problem (1). So the overall costs would be as follow.

Sub-problem (1)

- $n \gg k$: ThinQR: $2nk^2 - 2/3k^3$ flops;
- $n \approx k$ ThinQR: $4/3n^3$ flops;
- LLS Solution: $nk(m + k)$ flops.

So the whole step would cost $nk(m+3k) - 2/3k^3$ flops for the rectangular case, and $mk(n+k) + 4/3n^3$ flops for the almost square one.

Sub-problem (2)

- $m \gg k$: ThinQR: $2mk^2 - 2/3k^3$ flops;
- $m \approx k$ ThinQR: $4/3m^3$ flops;
- LLS Solution: $nk(m+k)$ flops.

So the whole step would cost $nmk + nk^2 + 2mk^2 - 2/3k^3$ flops for the rectangular case, and $nmk + nk^2 + 4/3m^3$ flops for the square case (which is the worst case at all, since $m \approx n \approx k$).

Sub-problem (1) - Regularized

- $n \gg k$: ThinQR: $2(n+k)k^2 - 2/3k^3$ flops;
- LLS Solution: $k(n+k)(m+k)$flops.

Regularization provides an additional guarantee about rectangularity of the input matrix, being the case $n+k \approx k$ not possible. So the whole step would cost $k(n+k)(m+3k) - 2/3K^3$ flops.

Sub-problem (2) - Regularized

- $n \gg k$: ThinQR: $2mk^2 + 4/3k^3$ flops;
- LLS Solution: $nk(m+2k)$ flops.

So the whole step would cost $2mk^2 + 4/3k^3 + nk(m+2k)$ flops.

Cost in memory for each subprolems

In terms of memory, usage of the Thin QR factorization allows, in the tall thin case, to save costs in memory.

Storing $Q_{1(V)}$ has a cost of $O(n \times k)$, instead of $O(n^2)$, while storing $Q_{1(U)}$ costs $O(m \times k)$, instead of $O(n^2)$. In both sub-problems, storing R_1 costs $O(k^2)$ instead of respectively $0(m \times k)$ and $0(m \times k)$. Indeed, a complete (1) step would cost $O(n \times k) + O(k^2)$ while (2) costs $O(m \times k)$ + $O(k^2)$.

Regarding regularized sub-problems storing $Q_{1(V)}$ has a cost of $O((n+k) \times k)$, while storing $Q_{1(U)}$ costs $O((m+k) \times k)$. In both sub-problems, storing R_1 still costs $O(k^2)$. Indeed, a complete regularized (1) step would cost $O((n+k) \times k) + O(k^2)$ while regularized (2) costs $O((m+k) \times k) + O(k^2)$.

3.7 Backward stability

We have theoretical guarantees about backward stability of multiplication by orthogonal matrices, like in the case of the Householder vectors, used for the computation of our QR, which can be considered itself backward stable as well, involving only such kind of multiplications. Finally, backward stability also holds for the QR based LLS solver (back substitution of a triangular

matrix) allowing us to consider such solvers safe in terms of posterior residual stability, so that we do not expect accuracy issues superior to a factor of $O(\mu)$.

Possible minor stability issues may be related to the $\hat{A}_s = U_s V_s^T$ product, computed to estimate the error at each iteration 11, being it in general not backward stable, such results does not directly impact on algorithm precision, in the worst case it may only trigger the early stopping condition too late or too early. To mitigate such danger, the usage of the heuristic stopping criteria (equation 12) may be considered less unstable (less products to compute, half of them involving zero-valued elements). Anyway, being this an estimation, the algorithm itself can still be considered totally backward stable.

4 Code and implementation

During the code development phase, some methodological decisions were made that turned out be crucial for the final design of our numerical algorithm; all of these decisions are listed below.

4.1 Tools and libraries

The whole QR-ALS algorithm was implemented from scratch using Matlab environment; the code was developed without relying on any pre-built libraries except for some pre-existing numerical functions: linsolve, norms, matrix multiplication methods and utilities for plots. More in details, since solving linear systems was only an internal step in the two sub-problems explained in Section 1.3 we decided to facilitate the computation of these internal steps by exploiting Matlab default command for solving linear systems. Moreover, Matlab numerical functions for solving QR Factorization and Singular Value Decomposition were used to test the correctness and efficiency of our algorithm as well as to compare the residual of our solution with optimal one. All the results obtained in the experimental phase are saved in csv file available on Google Drive; python notebook and pandas library was preferred over MatLab to easily explore the results, finding among them visible trends and pattern via charts.

To compute the optimal solution of the regularized problem A_λ^* the Optimization Toolbox was used. An object `oprimproblem` was instantiated and equation 8 was set as the objective to be minimized. The solver relied on a further library method itself called `fminunc`, able to chose a suitable optimization method for the problem [20]. In the tested cases the chosen solver was a "BFGS-quasi-newton" method. Returned solution was guaranteed to be at least a local minima. Stopping parameters were relaxed and the number of epochs was increased, so that the iterative algorithm could continue searching "for a long time".

The linear auto-encoder was implemented using the Statistics and Machine Learning Toolbox. In particular a `patternet` object was built and adapted. T-SNE implementation was taken from such toolbox too.

4.1.1 Implementation of Thin QR Factorization algorithm

Most of our efforts focused on developing our own version of the QR Factorization which is a key point in the final algorithm. Since such an implementation may turn out to be computationally

expensive, we have tried to adopt some expedients for optimizing as much as possible this procedure. The whole factorization process of a matrix is handled throughout specific function for computing the thin QR factorization via Householder vectors.

We preferred the Thin QR factorization to the full QR Factorization because we need to perform the factorization process on V^T and U which always turn out to be tall and thin matrices.

In accordance with theory, assuming we have a matrix $A \in \mathbb{R}^{m \times k}$, if $m > n$, we can express the Thin QR factorization as follows:

$$A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \cdot \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 \cdot R_2 + Q_2 \cdot 0 = Q_1 \cdot R_1$$

where Q_1 is a tall and thin matrix with orthonormal columns and R_1 is a square and upper triangular matrix. By focusing on Q_1 and R_1 we are able to store in memory only Q_1 with $O(mn)$ which is much cheaper to store in memory than the whole matrix Q with $O(m^2)$ entries.

In practice we obtain Q_1 by only computing the first n columns (without computing the Q_2 block): we iterate over the columns of Q_1, initialized as $m \times n$ identity matrix, substituting the last i values of each column as follow:

$$Q_1[i : m, 1 : n] = H_i * Q_1[i : m, 1 : n]$$

As optimization tricks we have avoid to compute the entire Householder reflectors matrix; for computing the Q_1 and R_1 exploiting directly the *Householder vectors*. In order to save memory, we have used the zero's space in the upper triangular matrix R_1 to store the required Householder vectors. A more detailed explanation of the procedure is illustrated in Alg. 2.

Algorithm 2 Thin QR factorization

Require: $A \in \mathbb{R}^{m \times n}$

Initialize $Q \leftarrow I \in \mathbb{R}^{m \times n}$

for i = 1:n **do**

$A_{i+1:m+1,i}, A_{i,i} \leftarrow \textsc{ComputeHouseolderVector}(A_{i,m:i})$

$A_{i:m,i+1:n} \leftarrow A_{i:m,i+1:n} - 2A_{i+1:m+1,i}(A^T_{i+1:m+1,i} A_{i:m,i+1:n})$

end for

for i = n:1 **do**

$Q_{i:m,:} \leftarrow Q_{i:m,:} - 2A_{i+1:m+1,i}(A^T_{i+1:m+1,i} Q_{i:m,:})$

end for

$R \leftarrow \textsc{GetUpperTriangular}(A_{1:n,:})$

5 Experimental set-up

During the experimental part,our aim was to understand how the possible varying factors impact the problem and the algorithm. Two kinds of scenarios were configured:

- **Computational intensive scenarios (.a)**: where we focused on algorithm memory-time efficiency and scalability, playing with matrix dimensions and shapes. For this class of experiments we will first define some ranges for properties to test, combine them and gather metrics concerning execution results. This "quantitative" approach helped us better understand computational properties and limit of our algorithm, as far as clarifying several aspect of the problem and the most appropriate way to measure them.
- **Convergence related scenarios(.b)**: where we experimented with particular algebraic properties of the involved matrices (A and both V and U), in order to analyze algorithm behaviour in limit cases. Approach here will be more "qualitative", one fundamental step will be to formulate assumption, and then build a restricted experimental setup to verify them and draw consideration about the actual behaviour with respect to the hypothesized one.

5.1 Metrics

Here follow a brief reminder of all the metrics used.

- **Loss** $\mathcal{L}(U,V) = \|A - UV^T\|_F$; it is the loss that is implicitly optimized in order to solve the low-rank approximation problem.
- **Gap** $G(U,V) = \|A^* - UV^T\|_F$; it is the distance from the optimal solution.
- **Relative Gap** $R(U,V) = \|A^* - UV^T\|_F / \|A^*\|_F$; the normalized gap.

Given the optimal solution $A*$ (which corresponds to k-truncated SVD decomposition of A), the optimal loss is expected to be equal to $\mathcal{L}^* = \|A - A^*\|_F$, and hence always greater or equal than 0 (equal in the limit case of $A = A^* \implies rank(A) = k$), while the two gaps are expected to reach 0 at the optimum.

When regularization is applied, metrics need to be updated as follow:

- **Regularized Loss** $\mathcal{L}_\lambda(U,V) = \|A - UV^T\|_F + \lambda_U\|U\|_F + \lambda_V\|V\|_F$; it corresponds to the loss implicitly minimized by the regularized solver: the penalized version of $\mathcal{L}$.
- **Regularized Gap** $G_\lambda(U,V) = \|A^*_\lambda - UV^T\|_F$.
- **Regularized Relative Gap** $R_\lambda(U,V) = \|A^*_\lambda - UV^T\|_F / \|A^*_\lambda\|_F$

In order to estimate the global optimum A^*_λ and observe if the algorithm still converges there, we should rely on an external solver, reaching a min of the loss:

$$\mathcal{L}^*_\lambda = \mathcal{L}_\lambda(U^*, V^*) = \|A - U^*(V^*)^T\|_F + \lambda_U\|U^*\|_F + \lambda_V\|V^*\|_F$$

Further information about the estimation of A^*_λ can be found in section 3.

- **Last iteration** total number of iterations s performed before the early stopping take actions; it could be equal to maximum number of iterations if stop condition are never satisfied.
- **Total time** Time (seconds) spent in running all iterations s up to last iteration.
- **iteration time** Mean time (seconds) spent in running a single iterations s.

Metrics were computed at each iteration and plotted. Mean value and standard deviation were computed with respect to the last iteration results, after a number of repetition r on the same target matrix A, changing only the initialization V_0. Few of the proposed configuration are clearly inspired from the ML world.

5.2 Time efficiency

In order to study the time consumption of our implemented algorithm we have compared them to some off-the-shelf MatLab functions.

- We analyzed time consumption of our Thin QR Factorization algorithm and since we want to study how much it differs in orders of magnitude from the MatLab's Thin QR factorization algorithm we have decide to compare them as the dimension of the involved matrices increases. The full QR implementation proposed during course lectures was taken into account as well.
- In order to obtain more informative details about the time consumption of the entire resolution algorithm QR-ALS we decided to simultaneously solve the lower rank approximation problem (P) with the Truncated SVD MatLab functions and compare the time required by both algorithm.

In Appendix A are available additional information about the matrix tested in these two experiments (respectively in Table 10 and Table 11).

5.3 Algorithm behaviour with respect to A

In this section effect of varying dimension of input matrix A were studied, focusing on the following cases:

- **Tall thin** (.a): $A_{tt} \in \mathbb{R}^{m \times n}$ with $m \gg n$
- **Short fat** (.a): $A_{sf} \in \mathbb{R}^{m \times n}$ with $n \gg m$
- **Square and almost square** (.a) : $A_{sq} \in \mathbb{R}^{m \times n}$ with $m \approx n$, $A_{sq} \in \mathbb{R}^{m \times m}$
- **Decreasing Rank** (.a) We would study algorithm behaviour (convergence speed, number of iteration to achieve a certain ϵ-precision) when $rank(A)$ come progressively closer to k, landing in limit case of $rank(A) = k$ (and more generally $rank(A) \leq k$).
- **Sparsity with density** d (.b) : $V_0^S \mid : p(v \in V_0^S = 0) = d_V$ The idea is to exploit presence of 0 to study algorithm behaviour and robustness for sparser data. Presence

of zeros might cause rank deficiency in the itermediate solution of the LLS or even spot problem concerning our implementation.

Detailed description of all the configurations explored during these

5.4 Algorithm behaviour with respect to V_0

In this section, the impact of various possible characterization of the V_0 matrix will be studied, in order to understand if some particular initialization could speed up the optimization or conversely reveal to be an obstacle toward the convergence (.b).

- **Random** Matrix elements are randomly sampled from an a normal distribution, arbitrary bounded.
- **Orthonormal columns** Matrix columns are random vectors mutually orthogonal to each other, hence their product is always 0.
- **Random Sparse with density** d_V A percentage of the non-zero elements in V_0 is regulated by a density parameter d_V. $d_V \in [0, 1]$ with default value fixed to 0.5.
- **Random sum-to-one-columns (Probability method)** Each column of the matrix is a sum to one vector of random probabilities or normalized "weights".
- **Binary values** V_0 will contain only zeros and ones. Density of ones is specified through the d_V parameter. $d_V \in [0, 1]$ with default value fixed to 0.5. Idea is to induce the algorithm to "select" some particular feature among the k columns of U and compute the ones in V as a filtered combination of them.
- **Not full column rank** Rank r of V_0 will be lower than k. Default value for r parameter is fixed to $k - 1$. Such configuration would help testing robustness of (A).s

More details on this experimental set up and all the tested matrices configuration are reported in Appendix A, Table 7. experiments are reported in Appendix A, Table 8 and Table 9.

5.5 Stopping policy

Before starting the aforementioned experiments, we decided to empirically observe the algorithm behaviour under the proposed stopping conditions, in order to fix a configuration to use.

First assumption concerned expected gap and stopping conditions relationship being not dependent on matrix shapes, we have considered w.l.g. only two configurations of (P): $A_1 \in \mathbb{R}^{1000\times30}$ with $k = 10$ and $A_2 \in \mathbb{R}^{200\times100}$ with $k = 20$. For each experiment the considered A matrix and the initial V_0 were randomly populated and fixed, solving the same problem under different stopping thresholds (ϵ or ξ) for several times. Table 4 in the appendix shows all the tested values and the obtained evaluation metrics.

Considering first the un-regularized case, evidence shows, as expected, a common trend: lowering the stopping hyper-parameter corresponds to an increase in the number of iterations and conversely to a decrease in the order of magnitude of the gap: when using ϵ, the gap seems

to approximate its square root (i.e. for $\epsilon = 10^{-6}$ we expect $G_A \approx 10^{-12}$), while if using the heuristic criteria, G_A seems to follow the same order of magnitude of the given ξ.

It is evident, as well, that with lower values of ϵ or ξ algorithms need to performed a greater number of iteration before achieving the desired threshold; anyway, using the same value for the two hyper-parameters,the ϵ-criterion is always expected to require a larger number of iterations with respect to ξ-criterion.

When the problem is solved with Thikonov regularization, the correspondence of number of iterations and "stricter" thresholds seems not to change, though the error improvement is no significant anymore, this is due to a shrinkage of the lower bound of the problem: possible optimal error is increased to earn regularity of the solution. So a "sufficiently strict" stopping threshold would interrupt the algorithm at a good iterations-error trade-off (i.e. $\epsilon = 10^{-2}$ and $\xi = 10^{-4}$).

All the subsequent experiments will be carried out using the ϵ *stopping criteria* described in Section 1.6. We fixed the following early stopping criteria parameter according to specific motivations:

- Maximum number of iteration $S = 500$ was selected according to the maximum dimension of the matrices involved in the experiments;
- $\epsilon = 10^{-6}$ is the desired threshold under which each matrix update has to be judged no more significant.

6 Experimental results

6.1 Time efficiency of the QR implementation

This section aims at describing the outcomes of the experiments mentioned in Sec. 5.2

Some effort has been addressed for evaluating the time-performance achieved with our implementation of Thin QR factorization. In order to obtain more informative insights such performance was compared with the ones achieved using the Full QR implementations proposed during the course lectures and the Thin QR implementation proposed by MatLab. For this experiment, random tall-thin matrices $A \in \mathbb{R}^{m \times n}$ were considered; given all the possible shapes of the A matrix coming out from the combinations of m and n values we were able to study time efficiency by increasing matrix dimensions as well as by increasing the "rectangularity" of A.

Relevant conclusions can be drawn by looking at the graph shown in Figure 4. The y-axis shows the average time (in seconds) required to factorize a matrix while the x-axis shows all the matrices involved in the experimentation.

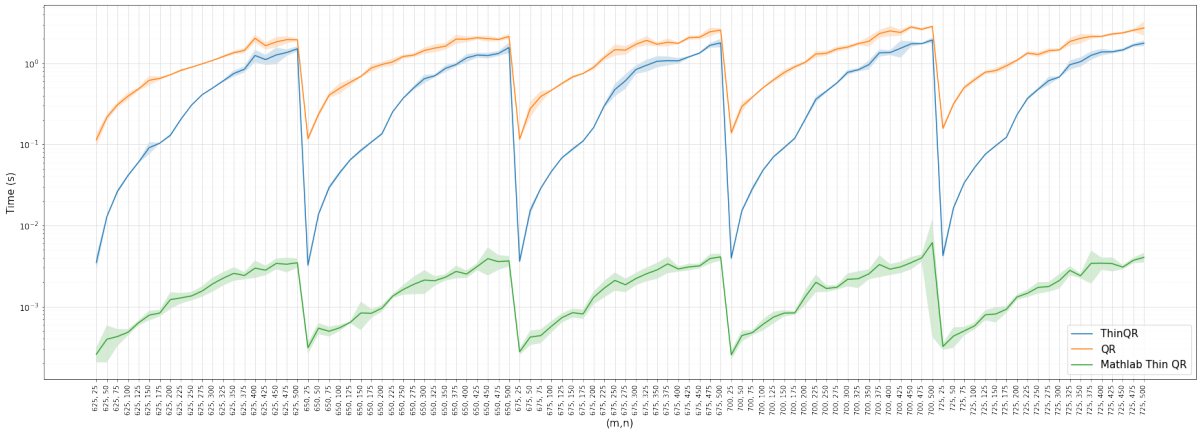


Figure 4: Time elapsed by each QR factorization implementation for different A matrices.

Comparison between Matlab ThinQR and our Full QR It's evident that MatLab Thin QR outperforms our Full QR by more than two orders of magnitude. The time required for the factorization goes from 10^{-1} for our Full QR to 10^{-4} for MatLab ThinQR in the case of strongly rectangular matrices. Similarly, the time required for the factorization goes from 10^{-3} for MatLab ThinQR to 10^{0} for Full QR in the case of almost square matrices.

This confirms our expectations by giving evidence of the high optimization of MatLab ThinQR implementation.

Thus, despite the Full QR is much more time-consuming, a similar behavior can be observed in both cases: the times required for factorization vary by one order of magnitude as the "rectangularity" of the matrix decreases.

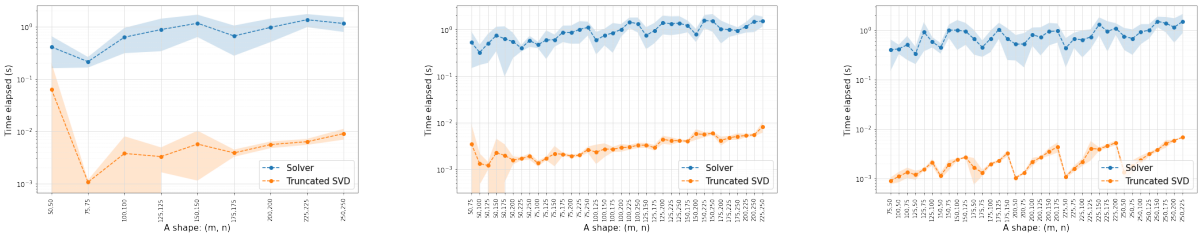
Comparison between our ThinQR and Full QR Considering Thin QR we can observe an increase of 2 orders of magnitude when passing from strongly rectangular matrices (10^{-3} seconds), to square matrices (10^{0} seconds). We should also mention that, when considering almost square matrices, we do not experience significant time savings in favor of our ThinQR w.r.t. Full QR. The advantage is evident only when looking at strongly rectangular matrices: in these cases the Thin QR outperforms the Full QR by 3 orders of magnitude, going from a mean time of 10^{0} seconds to a mean time of 10^{-3} seconds.

Comparison between our ThinQR and MatLab ThinQR With strongly rectangular matrices our Thin QR differing from the MatLab Thin QR by only 1 order of magnitude. It achieve performance on rectangular matrices comparable to the performance of MatLab ThinQR on almost square matrices.

A slight increase in the required time is observable, for all the considered QR implementations, when increasing the values of m and n; but it appears to be negligible among this experimentation. It became evident that for each QR implementation the time required for the factorization of A is mostly dependent on the "rectangularity" of A: the higher is the m/n ratio, the less time is required for the matrix factorization. Therefore, our implementation of Thin QR, is a competitive alternative when applied on strongly rectangular (tall-thin) matrices.

6.2 Time efficiency with respect to SVD

We proceeded in carrying out a comparative analysis of the time required for solving this low rank approximation problem with our algorithm and with Truncated SVD implementation proposed by MatLab. Given a fixed rank $k = 10$, several matrix $A \in \mathbb{R}^{m \times n}$ were considered with $m, n > 10$.



(a) Square A matrices. (b) Short-fat A matrices. (c) Tall-thin A matrices.

Figure 5: Time efficiency of our solving algorithm w.r.t. Truncated SVD (MatLab).

The graph above show how, given a random matrix A, significant time advantages are always gained when MatLab Truncated SVD is used. In fact, the average resolution time required by SVD is 3 orders of magnitude lower than with our algorithm, whatever the shape of A involved in the approximation process is tall-thin, short-fat or almost square. Again, considering the high level of optimization of Matlab's default functions, this result was expected.

For both our algorithm and SVD, keeping k unchanged, the time required is proportional to the size of the involved A matrix: as the values of m and n increase, a slight increase in the elapsed time is observed. This is especially more visible in the case of tall thin A matrices.

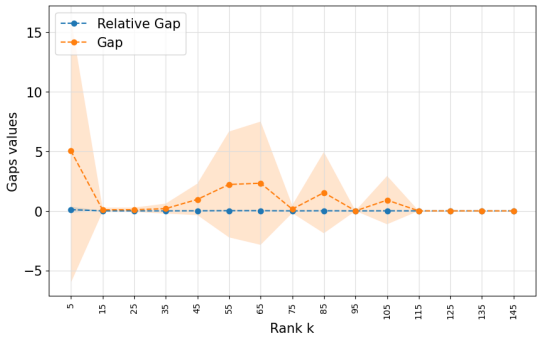
Still concerning the tall thin A matrices, it is possible to reach an interesting conclusion: by fixing m and k and varying n, there is a visible increase in the elapsed times with SVD as the value of n increases.

While on one hand this analysis highlighted how the MatLab Truncated SVD alternative outperforms (in terms of required time) our proposal, on the other hand it was useful in identifying the maximum size of the matrix A involved in the approximation process for which the resolution of the problem can be performed in reasonable time by our algorithm. Thus, it provides some insight into the maximum values of m and n for which it was possible to study additional A properties (e.g., sparsity) without conducting too time-consuming experiments.

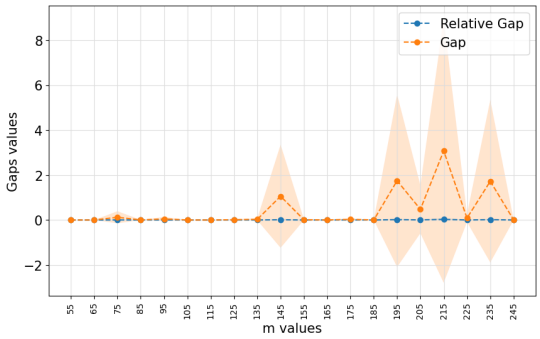
6.3 A shape: problem complexity

This section aims at describing the outcomes of the experiments mentioned in Sec. 5.3.

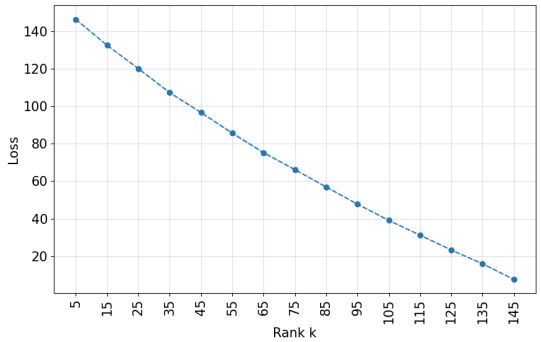
We started the proposed experiments by analyzing how loss and gaps change when two among the values of m, n and k are fixed and the other one is changing.



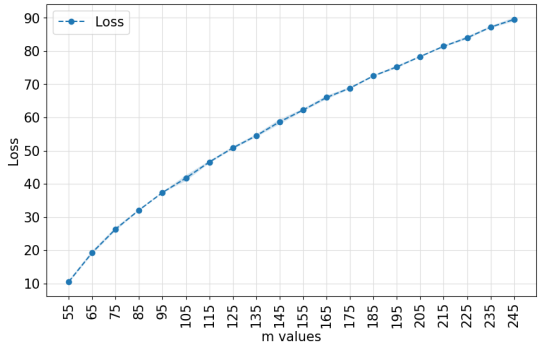
(a) Relative and absolute gap by changing target rank k; m and n are fixed resulting in $A \in \mathbb{R}^{250\times150}$ and $k \in [10, 150]$.



(b) Relative and absolute gap by changing m; $k = 50$ and n are fixed resulting in $A \in \mathbb{R}^{m\times150}$ with $m \in [50, 250]$.



(c) Loss by changing k rank; m and n are fixed resulting in $A \in \mathbb{R}^{250\times150}$ and $k \in [10, 150]$



(d) Loss by changing m; fixed $k = 50$, n is fixed resulting in $A \in \mathbb{R}^{m\times150}$ with $m \in [50, 250]$.

Figure 6: Gap behaviours.

Looking at Figure 6 we can state that, for fixed dimension of A, relative and absolute gap grow when increasing k rank, while loss decreases. Instead, when n and k are fixed, an increase in the m values lead to a decrease of relative and absolute gap and to an increase in the loss. Such behaviour can certainly be explained with the increase of representative power that an higher k gives to the low rank approximation, the approximated matrix is therefore expected to be closer (in terms of norm) to the original A. On the other hand, the more k grows, the more the optimization process becomes expensive and the optimality of our solution (under the fixed stopping condition) requires more iteration (more than the maximum number fixed for this experiment). So such trade-off should always be considered when approaching to a problem considering m, n, k and the available computational resources.

6.4 Sparsity of A

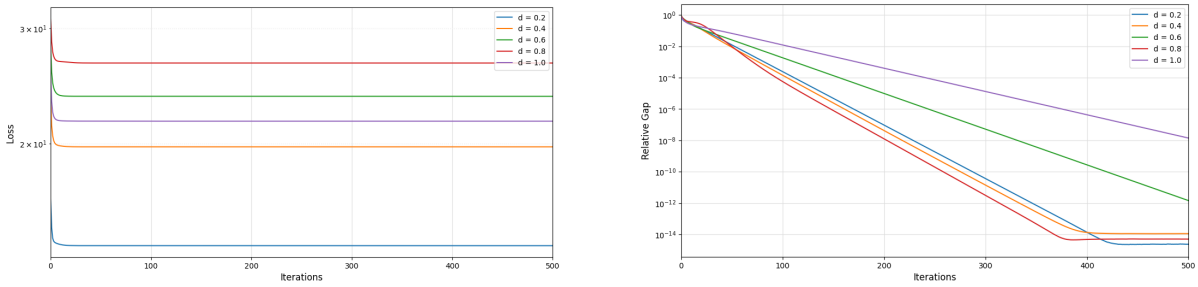
With the following experiment, we wanted to asses "how much" the QR-ALS algorithm is able to handle a sparse A matrix (without the addition of regularization). Several sparse matrices $A \in \mathbb{R}^{m\times n}$, as well as different values of k, were tested as the density d of nonzero elements in

the A decreased.

Density (m, n, k)	0.2	0.4	0.6	0.8	1.0
(150, 50, 5)	32.850	43.773	51.318	57.670	61.106
(150, 50, 20)	21.156	29.017	34.644	37.956	40.759
(150, 50, 30)	14.543	20.705	24.552	26.720	28.611
(150, 100, 5)	47.809	65.308	75.834	84.544	90.253
(150, 100, 20)	37.135	51.443	59.270	65.732	70.805
(150, 100, 30)	31.154	42.114	50.387	55.684	59.373
(200, 50, 5)	38.275	51.125	59.784	66.275	71.158
(200, 50 20)	25.778	35.336	41.821	46.196	49.370
(200, 50, 30)	18.434	25.467	30.063	33.416	36.108
(200, 100, 5)	55.992	75.238	88.525	97.557	104.450
(200, 100, 20)	44.675	60.686	71.806	78.767	84.894
(200, 100, 30)	38.230	52.030	61.551	67.769	73.242

Table 1: Mean loss achieved out of 5 run of the algorithm considering different (m, n, k) combinations and different λ values.

Table 1 shows that the sparser A is, the lower will be the loss value achieved by QR-ALS algorithm at the last iteration.



(a) Loss among iterations.

(b) Relative gap among iterations.

Figure 7: Impact of sparsity of A with $(m, n, k) = (100, 30, 10)$

Looking at relative gap among iterations in 7, we can see that generally, the gap tends to stay higher according to d. So we can state that, the whole problem (P) tends to be simpler. Such phenomena can be interpreted as an optimization focused on less elements, implicitly recognized as more salient features, skipping the zero-involving computations. Anyway such consideration should be taken "softly", not forgetting that being the values of A (and also of V_0) random

sampled, the convergence speed is not always as faster as much A is sparse. oLast but not least, the absence of numerical failures through the various experiments confirms that the QR-ALS is robust to a sparse input matrix (at least until $d \geq 0.2$).

6.5 V_0 initialization

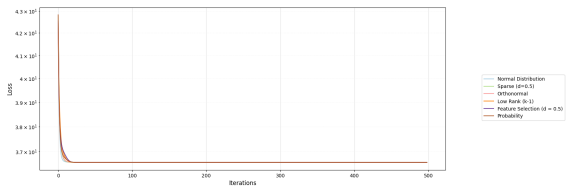
Given all the initialization methods for the V_0 matrix mentioned in section 5.4, a tailored analysis was carried out in order to verify if some performance benefits could be gained using few of these strategies, or conversely, if some initialization can worse or even interrupt convergence. Anyway theoretically known limit cases were described and studied in Section 6.7.

By fixing m, n, and k, a random (non-sparse) matrix $A \in \mathbb{R}^{m \times n}$ was generated by sampling its value from a normal distributions. Then, problem (P) w.r.t. A was repeatedly solved ($\lambda_U = \lambda_V = 0$) using each time a different initialization method for the matrix V_0.

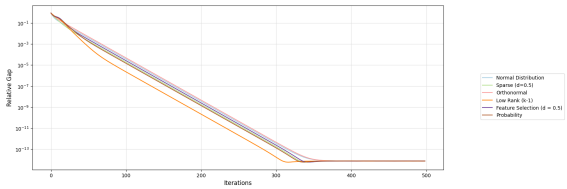
V_0 initialization	Mean Loss	Mean Relative Gap
Normal distribution	72.833667 28.528533	0.000050 0.000134
Orthonormal	72.646833 28.537270	0.001702 0.004206
Probability	72.693556 28.776920	0.011824 0.038963
Low rank (k-1)	72.671611 28.790466	0.019580 0.007934
Sparsity (d=0.5)	72.741167 29.021109	0.001273 0.004983
Binary (d=0.5)	72.837444 28.687072	0.006649 0.017249

Table 2: Mean Loss and Relative Gap achieved by QR-ALS for different (m, n, k) combination mentioned in Appendix 7 and different V_0 initialization method.

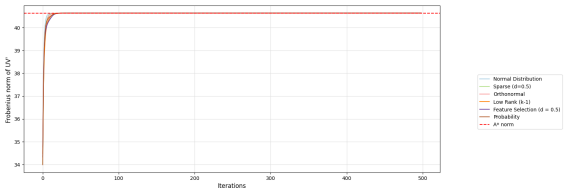
The case in figure 8 with $(m, n, k) = (100, 30, 10)$ was taken as examples to illustrate the behaviour of losses and relative gaps among the iterations. Also Figure 8 suggest us that even using different V_0 and despite some initial different oscillation, the same relative gap and loss values are achieved after a sufficient number of iterations.



(a) Loss among iteration



(b) Relative gap among iterations.



(c) Norm of UV^T among iterations w.r.t norm of A_*

Figure 8: History among iterations for different metrics with several V_0 initialization technique with $m = 100, n = 30, k = 10$.

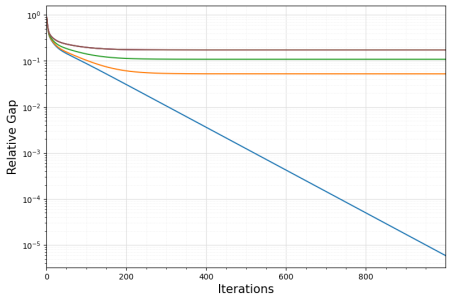
So we can state that algorithm performance is not so sensitive w.r.t. the particular initialization tested. Such "initialization invariance" seems to persist independently from the characteristics of A.

6.6 Regularization: convergence and optimality

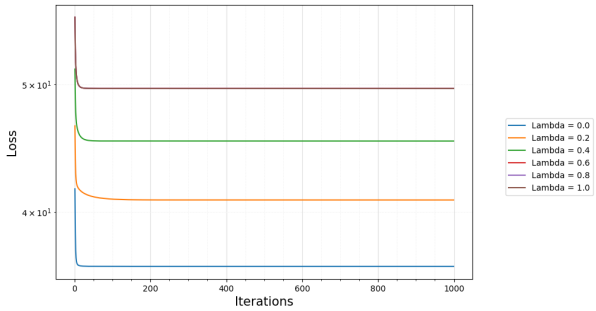
To address how Thikonov regularization impact on the algorithm behaviour we considered a random matrix $A \in \mathbb{R}^{m \times n}$ and we tried to figure out if different behaviours come out by changing the values of λ_U, λ_V. For all the case-study concerning the regularization impact we have always imposed the constraint $\lambda_U = \lambda_V$.

The outcomes for a single case study with $(m, n, k) = (100, 30, 10)$ are reported in Figure 9. [1]

[1] In Figure 9 the curves for $\lambda = 0.6$ and $\lambda = 0.8$ are not visible since they follow the same path of the curves for $\lambda = 1.0$.

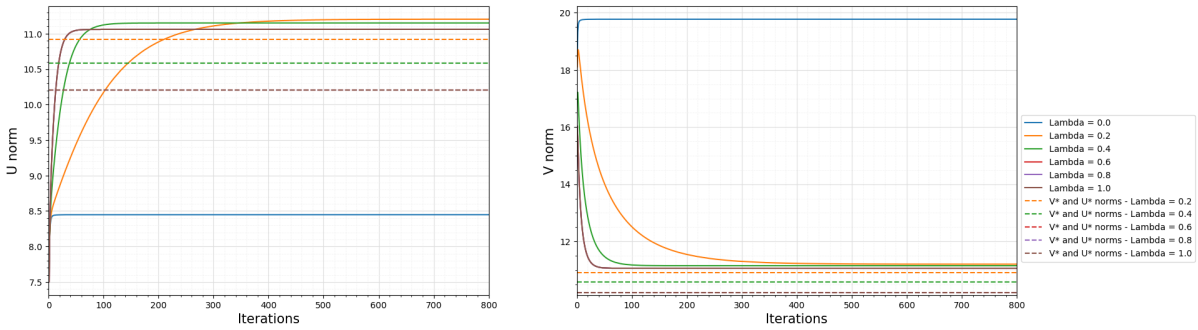


(a) Regularized relative gap among iterations.

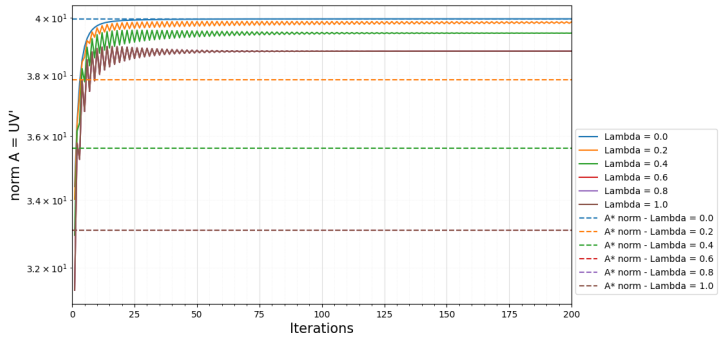


(b) Regularized loss among iterations.

Figure 9: Metrics values for differentλ values among iterations $(m, n, k) = (100, 30, 10)$.



(a) Regularization impact on U-norm and V-norm among iterations).



(b) Regularization impact on UV^T-norm among each iteration step s_1 and s_2.

Figure 10: Impact of different λ values among iterations for $(m, n, k) = (100, 30, 10)$.

As expected, high regularization impacts the norm of U and V. In the mentioned example regularization helps to balance out the norms of the 2 matrices by raising the norm of U and decreasing the norm of V iteration by iteration, thus forcing both to stay low. The higher is the λ value the more the two norms are both forced to stay low in fewer number of iterations.

The interesting fact is that for $\lambda > 0$ and s equal to the last iteration of the algorithm (s could be equal to the maximum number of iterations or to the iteration where stopping conditions take action), algorithm use a finite number of epochs $t \leq s$ to balance out the norm of U and V according to the Thikonov constraint.

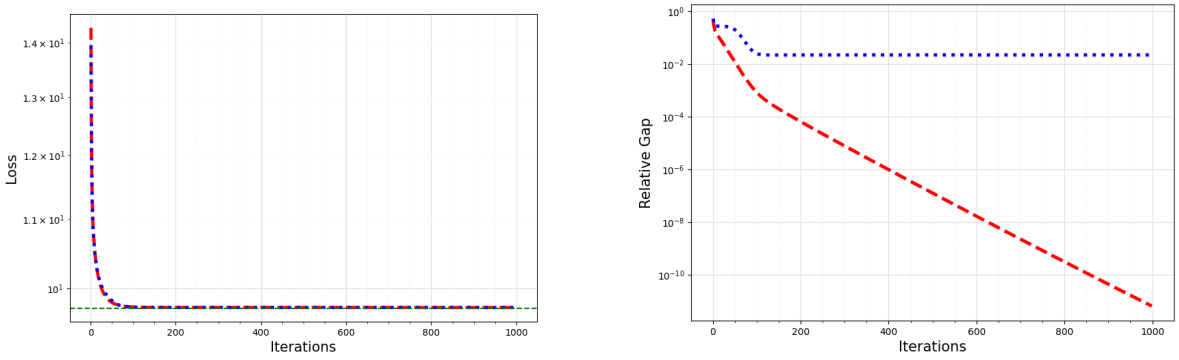
We can observe that the regularized loss and relative gap tends to stay higher among the iterations for higher λ values. Again, even adding Thikonov regularization, if the preconditions are respected, loss and gap are monotonic decreasing throughout iterations according .

Plots also confirm is useful only when the context of the lower-rank approximation problem requires keeping the norms of the U and V matrices balanced, being the regularized approximation generally worse in terms of pure distance from A ($\|A - UV^T\|_F$).

Gap plots seemed to suggest that the algorithm is not converging to the A^*_λ solution estimated with the external solver but somewhere else.

Recalling [4], the fixed point of the series is guaranteed to be a critical point of the problem, but we have no guarantee that it should correspond to A^*_λ). For such reason, we tried to measure gap with respect to the fixed point of our algorithm itself, in order to have at least a visual evidence of the q-linear convergence claimed in the paper.

A target A matrix and a V_0 initialization were fixed randomly. Then the algorithm was initially left looping for a number of iterations "tending to infinite" (actually 10.000), estimating the fixed point of the series $A^{*\prime}_\lambda$. Gap was measured w.r.t. to $A*_\lambda$ for the first 1000 iterations. Then a new initialization V'_0 was sampled and algorithm was executed again on the same A. This time the gap was measured with respect to $A^{*\prime}_\lambda$.



(a) Loss of the two execution

(b) Relative gap curves

Figure 11: Execution of the algorithm computed with respect to A^*_λ and with respect to $A^{*\prime}_\lambda$ (red). The loss corresponding to A^*_λ is report too (green) being minor that the others.

The resulting figure (11) shows that the regularized algorithm, differently from its un-regularized counterpart, does not reach a global minima, and neither A^*_λ, but returns a critical point worse in terms of loss. Convergence speed followed the expectation confirming being q-linear with regularization too.

6.7 Algorithm limit case

This section aims to explore algorithm's weakness and limit cases that came out during a preliminary experimental phase, mentioning the numerical problems encountered and the underlying causes (if known and understood) and the adopted solutions.

Sparse A and V_0 matrices Early experiments have exposed a weakness of our algorithm related to sparse initial A or V_0 matrices when they are initialized with default random sparse method or with binary values method (only zeros and one elements) for V_0. In these case studies, we have noted a frequent numerical error invalidating the achievement of a solution. Tables 4 and 3 show some example of the frequency of error occurrence out of 50 runs of the algorithm - where the sparse initial matrix is randomly initialized each time. These data prove that our algorithm was initially unable to handle with sparsity property affecting the final performance since the expected computations were not optimized to handle a relevant presence of zero elements, thus, going to result in numerical errors.

Density (λ_U, λ_U)	0.0	0.2	0.5	0.8	1.0
(0.2, 0.2)	36	50	43	26	43
(0.5, 0.5)	40	50	42	27	45
(0.8, 0.8)	40	50	44	40	46
(1.0, 1.0)	43	50	43	32	45

(a) $m = 150, n = 100, k = 50$ and default sparse matrix initialization technique.

Density (λ_U, λ_U)	0.0	0.2	0.5	0.8	1.0
(0.2, 0.2)	42	50	40	31	19
(0.5, 0.5)	47	50	46	34	25
(0.8, 0.8)	44	50	45	30	24
(1.0, 1.0)	37	50	44	32	18

(b) $m = 150, n = 100, k = 50$ and binary values initialization technique.

Table 3: Occurrences of numerical error for sparse matrices V_0.

Density (λ_U, λ_U)	0.0	0.1	0.2	0.3	0.4	0.6	0.8	1.0
(0.1, 0.1)	0	48	17	1	0	0	0	0
(0.4, 0.4)	0	46	18	5	2	0	0	0
(0.6, 0.6)	0	44	18	5	0	1	0	0
(0.8, 0.8)	0	44	17	5	3	0	0	0
(1.0, 1.0)	1	44	17	5	3	0	0	0

(a) $m = 15, n = 15, k = 5$

Density (λ_U, λ_U)	0.0	0.1	0.2	0.3	0.4	0.6	0.8	1.0
(0.2, 0.2)	0	40	7	2	0	0	0	0
(0.4, 0.4)	0	34	9	0	0	0	0	0
(0.6, 0.6)	0	35	6	2	0	0	0	0
(0.8, 0.8)	0	42	7	1	0	0	0	0
(1.0, 1.0)	0	32	9	2	1	0	0	0

(b) $m = 200, n = 20, k = 10$

Table 4: Occurrences of numerical error for sparse matrices A.

We have observed that starting from sparse matrices A or V_0 the sparsity propagates through iterations of the algorithm leading to sparse matrices U and V, having in some of these cases, one or more whole null columns. Null columns in the matrices generate a numerical error in the Thin QR factorization (pseudo code line) during householder computation. In fact since the householders vector is computed as $u = v/||v||$ a numerical problem came out from such a computation whenever v was a vector of only zero values. The generated `nan` value was then propagated in the subsequent steps invalidating the achievement of a solution. Under such conditions, the lower is density parameter d of the initial matrix A or V_0, the more likely is the numerical error to occur. We solved this problem by introducing a control instruction in the householder computation such that $u = v$ if $v_i = 0 \forall i$, and $u = \frac{v}{||v||}$ otherwise. After this optimization we have been able to successfully carry out experiments related to sparsity properties and all the results has been already proposed in previous section.

Initial rank deficient V_0 When the V_0 matrix is not full column rank, all the nice assumptions concerning monotonicity and convergence decay because of non convexity of subproblems and non uniqueness of the LLS solutions.

As already said, such kind of initialization can be handled just using Thikonov regularization, enforcing a greater $rank(V_0)$ by considering expanded matrices, or equivalently, penalizing solution of the LLS with higher norms. Anyway it must be kept in mind that

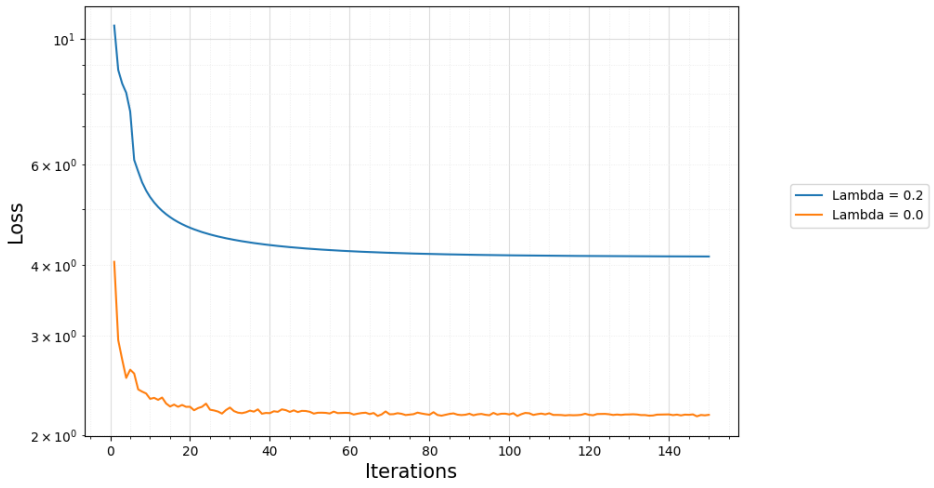


Figure 12: Loss curve for the approximation of a matrix $A \in \mathbb{R}^{20x20}$ with rank deficient initialization ($rank(V_0) = 2$).

the problem solved by the regularized algorithm is anyway a different one (as the loss plot shows 12).

SVD as initialization Considering U, Σ, V the Truncated SVD factorization of the original matrix A, then our algorithm lead to the optimal solution in just one iterations if $V_0 = \Sigma_{1:k,1:k} V^T_{1:n,1:k}$ but lead to no solution (i.e. numerical error) when one ore more singular value of Σ is replaced with zeros before the computations $V_0 = \Sigma_{svd} V^T_{svd}$. This error occurs since no solution for the intermediate LLS is available (Matlab returning matrices of NaN values). Such product of the ablated Σ and V corresponds to a spurious saddle point of the low rank approximation problem [17] and as expected, the algorithm is not able at all to deal with them. Anyway, we can certainly state that when the execution does not crush, they have not been met.

7 Algorithm applications

7.1 Approximating truncated SVD

The QR-ALS algorithm result can also be used to approximate a SVD decomposition, given the final approximation $\tilde{A}$:

$$\tilde{A} = \tilde{U}\tilde{V}^T = Q_{1(\tilde{U})} R_{1(\tilde{U})} R^T_{1(\tilde{V})} Q^T_{1(\tilde{V})} \tag{23}$$

where the two QR are the ones computed in the last iteration, we can compute the SVD of the $k \times k$ product of the two R matrices:

$$R_{1(\tilde{U})} R^T_{1(\tilde{V})} = U_R \Sigma_R V_R^T \tag{24}$$

and then substitute it back in 23 to obtain:

$$\tilde{A} = (Q_{1(\tilde{U})} U_R) \Sigma_R (V_R^T Q^T_{1(\tilde{V})}) = U_s \Sigma_R V_S^T \tag{25}$$

where the two parentheses contains products of orthogonal matrices, hence U_S and V_s are orthogonal themselves.

These additional step adds to the algorithm a cost of $O(k^3)$ for the SVD computation and $(m+n)k^2$ for what concerns the orthogonal matrices products, allowing to explicitly deal with a cube factor of m or n.

Once this further decomposition has been computed, QR-ALS can also be used to perform SVD based algorithms, like Latent Semantic Analysis (LSA).

7.2 Auto-encoder: idea and hypothesis

Intuition was breaded by Goodfellow et al. [15][p.503, eq. 14.1], claiming that a shallow under-complete auto-encoder with identity activation function converges to the PCA of the data.

Given the strong bound between SVD and PCA, our optimization can be seen as a training algorithm, jointly optimizing the data internal representation, computed at each iteration as the U matrix (the same m patterns are mapped from a n to a k feature space), and the feature weight matrix, corresponding to the V matrix, expressing how much each of the n original features contributes to each of the k internal variables. So adopting the same formulation of [15], we can claim that:

$$f(X) = h \Leftrightarrow f(A) = AV^+ = U$$
$$g(h) = \tilde{X} \Leftrightarrow g(U) = UV^T = \tilde{A}$$

Experimentation will tell us if, under such assumption, our algorithm can compete with back-propagation. Additionally, we can take inspiration from the Machine Learning world to test neural network fashioned principles (like regularization or saturation avoidance) and heuristics on our algorithm.

7.3 AE: adding biases

Formulating the experiment setup we noticed that we can compare results of the two approaches only in terms of results $\tilde{D}$. To actually being able to compare our V and V^+ matrices with the weights of the neural network we should consider also the biases of each unit.

To replicate biased learning for the decoder, we just added a column of ones to the U matrix,

$$V_{biased} \in \mathbb{R}^{n \times k+1} = \arg\min_{V} \|A - [1, U]V^T\|$$

while for the encoder, the column of ones should be added to A (and compute the new U mapping and then the other V to obtain V^+, and the), as it should also be done during the inference phase.

$$\bar{V} \in \mathbb{R}^{n+1 \times k} = \arg\min_{\hat{V}} \|[1, A] - U\bar{V}^T\| = \bar{Q}_{1(\bar{V})}\bar{R}_{1(\bar{V})}$$
$$V^+_{biased} \in \mathbb{R}^{n+1 \times k} = \bar{V}^+ = \bar{Q}_{1(\bar{V})}(\bar{R}^{-1}_{1(\bar{V})})^T$$

so that once the optimization is ended, given an unseen pattern $p \in \mathbb{R}^n$ we can simulate its flow through the AE computing:

$$enc(d) : \mathbb{R}^n \to \mathbb{R}^k = [1, d_1, d_2, ...d_m] \times V^+_{biased}$$
$$dec(\hat{d}) : \mathbb{R}^k \to \mathbb{R}^n = [1, \hat{d}_1, \hat{d}_2, ...\hat{d}_k] \times V^T_{biased}$$

Given the actual weights θ_{enc} and θ_{dec} expect to obtain:

$$\lim_{S \to \infty} V^+_{biased} \approx \theta_{enc} \wedge \lim_{S \to \infty} V^T_{biased} \approx \theta_{dec}$$

With this setup we would be able to see if, the internal features learned by the auto-encoder, less than ordering, share some kinds of correspondence with the space explored by our algorithm. Additionally, it would be interesting to compare convergence and performance of the biased algorithm with the unbiased one.

7.4 AE: "Biased Training" possibilities

Once the formalization of the biased problem has been solved, another important choice is related to its usage during the alternate optimization problem.

Biased training: with such approach, the two sub-problems are solved in their biased version throughout the whole optimization loop (a biased version of U is computed too, using $\hat{V}_{biased}$ and the A matrix expanded with a column of ones.)

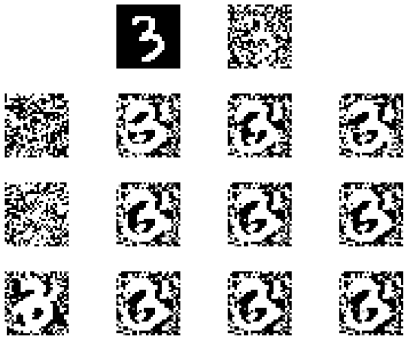
Greedy biasing: with such approach biased sub-problems are solved only once, as soon as the unbiased execution of (A) ends. In this way, the internal representation space of k-features is the one found by our algorithm, and the computation of the two V_{biased} and V^+_{biased} matrices corresponds to simulation of an auto-encoder pointing such manifold.

We would empirically compare the two approaches during the experiments.

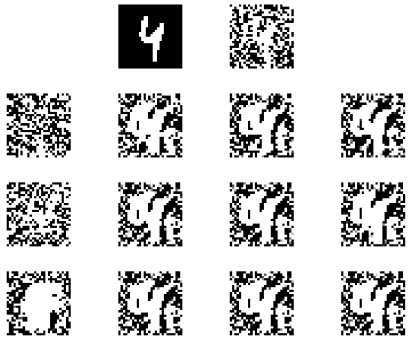
7.5 Evaluation on the MNIST dataset

The main hypothesis to prove with the built assumption is the equivalence of the optimization algorithm with the training of a linear under'complete auto-encoder. In particular we want to prove the effectiveness of the multiplication for V^+ and V to denoise and reconstruct data, and also compare and show similarity of the internal representations produced with the two approaches. To do so the MNIST handwritten digits dataset ([10]) will be used, processing each 28×28 gray-scale image as a 784 elements vector (no convolution). Denoising capabilities of will be evaluated training models with a single digit at time, to obtain $enc(d)$ and $dec(\hat{d})$, and then compute the reconstruction error on all the test instances of such digit, which are data that the algorithm has never seen. Doing so we expect to be able to prove learning and generalization capabilities. Also given the well known convergence behaviour of the QR-ALS, it is not given at all that a large number of iterarions would improve test results, on the contrary, being the amount of corrected error higher in the very first iterations, we expect a small number of repetitions to be sufficient to learn.

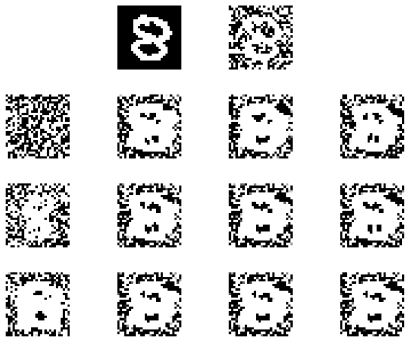
On the other hand, for the internal representation experiments, three digits will be picked up for the training, and then the t-SNE ([7]) algorithm will be applied to the embedding of the test images, observing the results to carry on a purely qualitative evaluation.



(a) Final reconstruction for digits 3.



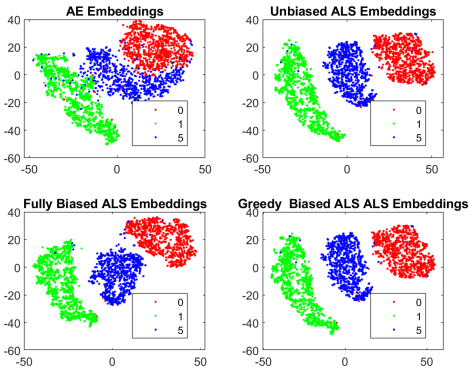
(b) Final reconstruction for digits 4.



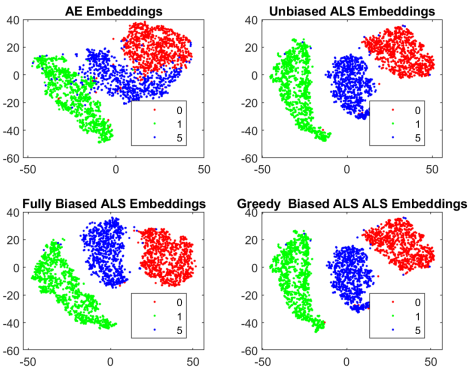
(c) Final reconstruction for digits 8.

Figure 13: The first line of each figure illustrate the original image and image corrupted with some Gaussian noise. The second line of each figure report the denoised version of the image obtained using classical Denoising AE, Fully-biased QR-ALS, Greedy-biased QR-ALS and Unbiased QR-ALS after 5 epochs. At same way, third and fourth row of each image show the denoised version of the image obtained using classical Denoising AE, Fully-biased QR-ALS, Greedy-biased QR-ALS and Unbiased QR-ALS after respectively 25 and 100 epochs.

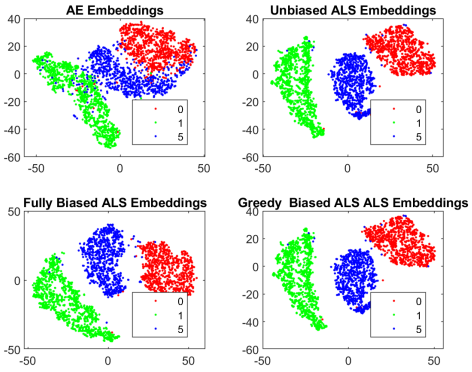
Experiments showed that the internal representations of our algorithm appear very similar to the ones of the AE: the shapes of the clusters look very similar at the net of few rotations. Such qualitative result shows that the intuition about learning capabilities of our algorithm were well defined. It's also possible to notice here the impact of the biasing: the greedy approach produced a subspace almost identical to the unbiased one. Hence it may be take into account only to achieve the same identical shape of the AE weights matrices. On the other hand, the fully biased training produced a slightly different representation, which requires a larger number of iterations to reach a "stable" configuration if compared to the other variants.



(a) t-SNE visualization of hidden representation after 5 epochs.



(b) t-SNE visualization of hidden representation after 25 epochs.



(c) t-SNE visualization of hidden representation of the test after 100 epochs.

Figure 14: Explorative analysis of the learned hidden representation of test data (unseen) for three random digits among different number of epochs (rank $k = 32$).

Digit	Epochs	AE error	Fully biased ALS error	Gready Biased ALS error	Unbiased ALS error
3	5	2.8114	0.02297	0.022534	0.022739
	25	0.91008	0.022718	0.022836	0.022853
	100	0.035973	0.022368	0.022537	0.022551
4	5	2.8401	0.020166	0.019128	0.019217
	25	1.0698	0.01862	0.018807	0.018747
	100	0.037151	0.018644	0.018651	0.018616
8	5	3.4999	0.025563	0.024295	0.024376
	25	0.97298	0.022739	0.02303	0.023037
	100	0.035928	0.022742	0.022733	0.022766

Table 5: Denoising results on test dat aafter digit-wise training/optimization. Rank was fixed at $k = 32$

Denoise results showed the QR-ALS has learning and generalization capabilities, and that, on the studied AE setup its fitting capabilities are considerably faster than the back-propagation, though not supporting batch training or non-linear activation functions ("train" on the full ten digits dataset would be computationally painful).

Anyway, having reached, in a few iterations, a significant lower error than a classical AE (as shown in Table 5), QR-ALS can still be considered useful for layer-wise pre-training or transfer learning.

8 Overall conclusions

The QR-ALS algorithm has been built up and expanded with additional features like regularization and stopping conditions. Typical behaviours has been analyzed, including monotonic decreasing of the error, the q-linear convergence to the optimal solution (truncated SVD) and how input dimensions m, n and k influence the computational complexity as far as backward stability.

Several optimization were implemented, including the early inversion of the R matrix in the first sub-problem (1) and the Thin QR with greedy storage of the Householder reflectors. Experiments allowed us validating theoretical claims and helped us better understand several limit cases and impact of several possible uncertainty features (input matrix shapes and density, initial V_0, regularization and ES hyper-parameters etc.), ending up identifying the appropriate usage case for the QR-ALS: strongly rectangular matrix (m $\gg$ n) with possibly low sparsity and with no strict precision constraints.

Robustness of the algorithm to several possible numerical fault has been evaluated and proved, and also some real world possible applications were identified (i.e. LSA). Finally, comparing

the low rank data representation of the algorithm to the one of a linear under-complete AE, it has been possible to prove algorithm learning and generalization capabilities, candidating it for a further number of other tasks, like denoise and outlier detection.

References

[1] Carl Eckart and Gale Young. "The approximation of one matrix by another of lower rank". In: *Psychometrika* 1.3 (Sept. 1936), pp. 211–218. ISSN: 1860-0980. DOI: 10.1007/BF02288367. URL: https://doi.org/10.1007/BF02288367.

[2] Andrei Nikolaevich Tikhonov. "On the regularization of ill-posed problems". In: *Doklady Akademii Nauk*. Vol. 153. 1. Russian Academy of Sciences. 1963, pp. 49–52.

[3] Luigi Grippof and Marco Sciandrone. "Globally convergent block-coordinate techniques for unconstrained optimization". In: *Optimization Methods & Software* 10 (1999), pp. 587–637.

[4] L. Grippo and M. Sciandrone. "On the convergence of the block nonlinear Gauss–Seidel method under convex constraints". In: *Operations Research Letters* 26.3 (2000), pp. 127–136. ISSN: 0167-6377. DOI: https://doi.org/10.1016/S0167-6377(99)00074-7. URL: https://www.sciencedirect.com/science/article/pii/S0167637799000747.

[5] James C. Bezdek and Richard J. Hathaway. "Some Notes on Alternating Optimization". In: *Advances in Soft Computing — AFSS 2002*. Ed. by Nikhil R. Pal and Michio Sugeno. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 288–300.

[6] James Bezdek and Richard Hathaway. "Convergence of alternating optimization". In: *Neural, Parallel Scientific Computations* 11 (Dec. 2003), pp. 351–368.

[7] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/v9/vandermaaten08a.html.

[8] Justin P. Haldar and Diego Hernando. "Rank-Constrained Solutions to Linear Matrix Equations Using PowerFactorization". In: *IEEE Signal Processing Letters* 16.7 (2009), pp. 584–587. DOI: 10.1109/LSP.2009.2018223.

[9] Emmanuel J Candes and Terence Tao. "The power of convex relaxation: Near-optimal matrix completion". In: *IEEE Transactions on Information Theory* 56.5 (2010), pp. 2053–2080.

[10] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[11] Lars Grasedyck, Daniel Kressner, and Christine Tobler. "A literature survey of low-rank tensor approximation techniques". In: *GAMM-Mitteilungen* 36.1 (2013), pp. 53–78. DOI: https://doi.org/10.1002/gamm.201310004. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/gamm.201310004. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/gamm.201310004.

[12] Prateek Jain, Praneeth Netrapalli, and Sujay Sanghavi. "Low-rank matrix completion using alternating minimization". In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013, pp. 665–674.

[13] Liqi Wang and Moody T. Chu. "On the Global Convergence of the Alternating Least Squares Method for Rank-One Approximation to Generic Tensors". In: *SIAM J. Matrix Anal. Appl.* 35 (2014), pp. 1058–1072.

[14] Trevor Hastie et al. "Matrix completion and low-rank SVD via fast alternating least squares". In: *The Journal of Machine Learning Research* 16.1 (2015), pp. 3367–3402.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[16] Joonseok Lee et al. "LLORMA: Local Low-Rank Matrix Approximation". In: *Journal of Machine Learning Research* 17.15 (2016), pp. 1–24. URL: `http://jmlr.org/papers/v17/14-301.html`.

[17] Thomas Hou, Zhenzhen Li, and Ziyun Zhang. *Asymptotic Escape of Spurious Critical Points on the Low-rank Matrix Manifold*. July 2021.

[18] Jun Lu. *Numerical Matrix Decomposition and its Modern Applications: A Rigorous First Course*. 2021. DOI: `10.48550/ARXIV.2107.02579`. URL: `https://arxiv.org/abs/2107.02579`.

[19] Kiryung Lee and Dominik Stöger. "Randomly Initialized Alternating Least Squares: Fast Convergence for Matrix Sensing". In: *arXiv preprint arXiv:2204.11516* (2022).

[20] MatLab. *Optimization Toolbox, Fminunc Find minimum of unconstrained multivariable function*. URL: `https://it.mathworks.com/help/optim/ug/fminunc.html#References` (visited on 03/31/2023).

[21] Wikipedia. *Matrix Sensing via Alternate Least Squares Minimization*. URL: `https://en.wikipedia.org/wiki/Matrix_completion#Alternating_least_squares_minimization`.

Appendix A Experiments configurations

ϵ	A_1 Mean Stop iteration	A_2 Mean Stop iteration	A_1 Mean Gap	A_2 Mean Gap
1e-2	160 ± 47	157 ± 126	3,744E-05 ± 4,103E-05	3,56E-05 ± 3,056E-05
1e-4	474 ± 362	367 ± 273	5,020E-09 ± 4,506E-09	4,10E-09 ± 2,779E-09
1e-6	840 ± 668	611 ± 425	3,162E-13 ± 2,126E-13	4,14E-13 ± 0
1e-8	1030 ± 616	857 ± 578	0	0

(a) Insights about last algorithm iterations w.r.t. to ϵ threshold.

ξ	A_1 Mean Stop iteration	A_2 Mean Stop iteration	A_1 Mean Gap	A_2 Mean Gap
1e-2	30 ± 6	22 ± 3	0,0321 ± 0,0147	1,264E-02 ± 7,397E-03
1e-4	103 ± 10	99 ± 61	0,0003 ± 0	3,340E-04 ± 3,382E-04
1e-6	221 ± 104	240 ± 147	5,629E-06 ± 7,486E-06	4,47E-06 ± 3,541E-06
1e-8	374 ± 268	417 ± 253	6,424E-08 ± 7,319E-08	4,74E-08 ± 3,220E-08

(b) Insights about last algorithm iteration w.r.t. to ξ threshold.

ϵ	A_1 Mean Stop iteration	A_2 Mean Stop iteration	A_1 Mean Gap	A_2 Mean Gap
1e-2	160 ± 47	100 ± 20	2,742E-04 ± 2,74E-05	7,920E-04 ± 3,209E-04
1e-4	477 ± 360	475 ± 194	2,056E-04 ± 7,671E-06	7,514E-04 ± 1,788E-04
1e-6	1613 ± 525	-	2,053E-04 ± 5,175E-07	-
1e-8	-	-	-	-

(c) Insights about last algorithm iteration with regularization ($\lambda = 0.2$) iterations w.r.t. to ϵ threshold.

ξ	A_1 Mean Stop iteration	A_2 Mean Stop iteration	A_1 Mean Gap	A_2 Mean Gap
1e-2	475 ± 124	1003 ± 46	2,058E-04 ± 4,875E-06	5,2504E-04 ± 1,919E-06
1e-4	-	-	-	-
1e-6	-	-	-	-
1e-8	-	-	-	-

(d) Insights about last algorithm iteration with regularization ($\lambda = 0.2$) iterations w.r.t. to ξ threshold.

Table 6: Impact of stopping criterion and corresponding threshold values on final algorithms performance. Whenever a dash is reported it means that the stopping criteria does not activate for a number of times sufficient to make statistical measurement significant (i.e. at least 2).

Parameters	**Values**
V_0 initialization	• Random normally distributed values • Binary values or sparsity $d \in [0, 1]$ • Orthonormal columns • Lower rank, $rank(V_0) = k - 1$ • Random sum to 1 columns (i.e, probability).
A initialization	normally distributed random values
m size	100, 150, 200
n size	50, 100
k rank	5, 20, 30
A density	1
λ_U, λ_V	0.0
Max. iterations	500
ϵ stop criteria	10e-6
ξ stop criteria	10e-12
Total combinations	**18**

Table 7: Settings for $V0$ initialization experiment.

Parameter	**Values**
A initialization	normally distributed random values
V_0 initialization	normally distributed random values
m size	[10, 50:250] stride = 50
n size	[10, 50:250] stride = 50
k rank	[2, 9, 50:200] stride = 50
A density	1.0
$\lambda_U = \lambda_V$	0.0
Max. iterations	500
ϵ - stop criteria	10e-6
Total combinations	**102**

Table 8: Settings for A-shape experiment.

Parameter	**Values**
A initialization	sparse
V_0 initialization	normally distributed random values
A initialization	normally distributed random values
m size	100, 150, 200
n size	50, 100
k rank	5, 20, 30
d density	[0.2:1.0], stride = 0.2
(λ_U, λ_V)	0.0
Max. iterations	500
ϵ - stop criteria	10e-6
ξ - stop criteria	10e-12
Tot. combinations	**192**

Table 9: Settings for A-sparsity experiment.

Parameter	**Values**
A initialization	normally distributed random values
n size	[25:500], stride = 25
m size	[525:1000], stride = 25
Total combinations	**400**

Table 10: Settings for QR efficiency experiment.

Parameter	**Values**
A initialization	normally distributed random values
V_0 initialization	normally distributed random values
m size	[50:250], stride = 25
n size	[50:250], stride = 25
k rank	10
A density	1
$\lambda_U = \lambda_V$	0.0
Max. iterations	500
ϵ - stop criteria	10e-6
Total combinations	**162**

Table 11: Settings for Solver efficiency experiment.