Non-negative Matrix Factorization and Robust PCA

MATH 6490: Optimization Methods for Data

Andrés Vargas

1 Introduction

In this lab, I compared the performance of several algorithms, which I fully implemented in MATLAB (did not use built-in MATLAB functions), for non-negative matrix factorization and robust pca; applying the former to the swimmer dataset and the latter to the escalator dataset. All algorithms achieved good performance, and some of them even managed exceptional performance.

2 Non-negative Matrix Factorization

Non-negative matrix factorization is instrumental in a myriad of data mining applications including text mining, bioinformatics, and nuclear imaging. It aims to find two nonnegative matrices $W \in \mathbb{R}^{m\times r}$ and $H \in \mathbb{R}^{n\times r}$ such that their product WH^T approximates a given nonnegative matrix $X \in \mathbb{R}^{m\times n}$. Under the assumptions of Gaussian noise, the nonnegative matrix factorization task can be formulated as

$$\underset{W,H}{\operatorname{argmin}} \frac{1}{2}||WH^T - X||_F^2 \text{ s.t. } W \in \mathbb{R}_+^{m\times r} \text{ , } H \in \mathbb{R}_+^{n\times r}, \text{ and } ||W_j|| \leq 1 \forall j \in 1, 2, ..., r \text{ ;}$$

where W_j is the j^{th} column of W and a "plus-sign" subscript indicates the positive orthant. The idea is, at each iteration, to take a gradient step and then project onto the feasible region defined by the constraints. To this end, I wrote MATLAB routines for full projected gradient descent, alternating minimization, and alternating projected gradient descent. I obtained the best performance using alternating projected gradient and the worst performance with full projected. Specifically, after 10000 iterations, full projected achieved a frobenius error of 1.0887, alternating minimization took only 1000 iterations to reach an error of 1.2e-03, and alternating projected took 1000 iterations to reach an error of 5.91e-04. To compare per-iteration performance, I plotted the objective function value for the first 1000 iterations.

We see in figure 1 that, although the alternating projected method ultimately achieved a lower error, it converged much more slowly than the alternating minimization. This should come as no suprise: no optimal solution is found at each iteration of alternating projected, whereas for alternating minimization, an optimal solution for each block is found at each iteration. So, in the beginning, alternating minimization trumps alternating projected. However, the same thing that makes alternating minimization converge fast also makes it greedy; meaning that it is highly susceptible to local optima. On the other had, alternating projected avoids many of the drawbacks of greediness by not full minimizing each block successively at each iteration. We see evidence of this in the fact that alternating projected eventually converges to a lower optimum than alternating minimization. Fully projected gradient performs significantly worse on all counts; I am pretty sure that the reason for this was that I didn't set the step size to the inverse of the induced 2-norm of the second derivative of my objective function. The reason I did not do this was because I wasn't sure how to compute the second derivative. In the next subsection, where I derive the step sizes, I explain the issue I had with the deriving the step size for the full projection.

2.1 Deriving Step Sizes

The objective function for the non negative matrix factorization is quadratic, which means we can set the step size to the inverse of the induced 2-norm of the second derivative. The first derivative of the problem in W is $(WH^T - X)H$. This is a matrix, and so differentiating it with respect to another matrix W yields a rank four tensor. To avoid having to deal with more dimensions than two, we can equivalently find the derivative of $vec((WH^T - X)H$ with respect to $vec(W)$, where $vec(\cdot)$ is the vectorization operation. First, note that

$$vec((WH^T - X)H) = (H \otimes I_m)^T[(H \otimes I_m)vec(W) - vec(X)] \text{ ,}$$

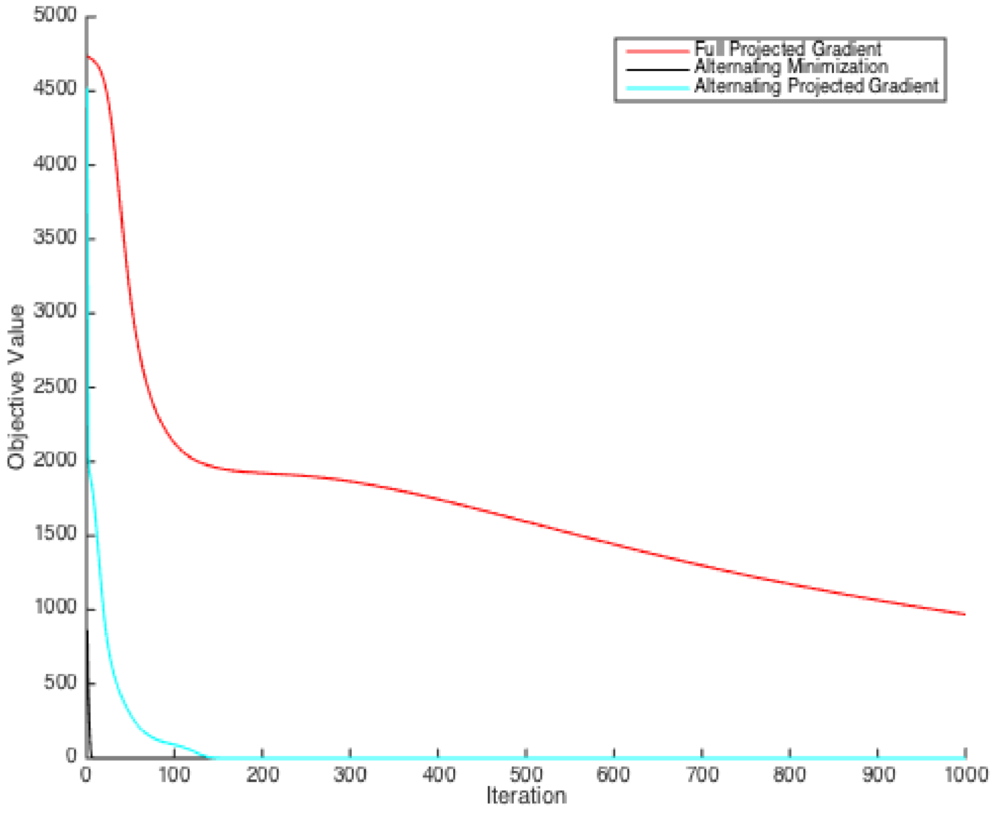

Figure 1: Objective Function Values for First 1000 Iterations

where $\otimes$ is the kronecker product operation. Then

$$\frac{\partial vec((WH^T - X)H)}{\partial vec(W)} = (H \otimes I_m)^T (H \otimes I_m) = (H^T \otimes I_m)(H \otimes I_m) = H^T H \otimes I_m \ .$$

As it turns out, $||A \otimes I||_2 = ||A||_2$ for any matrix A, where $|| \cdot ||_2$ is the induced matrix 2 norm and I is the conformable identity. Thus, we set the step size for the problem in W to $1/||H^T H \otimes I_m||_2 = 1/|H^T H||_2$. A similar derivation leads to a step size of $1/||W^T W||$ for the problem in H. But for the full problem, $H^T H$ and $W^T W$ are only the block diagonal elements of second derivative. We still need to find the off-diagonal elements. Even using the vectorized form, I wasn't sure how to do the math for this, but below I've written as far as I was able to get:

Another way of writing $vec((WH^T - X)H)$ is $vec(WH^T H) - vec(XH) = (I_r \otimes W)vec(H^T H) - (I_r \otimes X)vec(H)$. The corresponding off diagonal block of the second derivative matrix would then be the derivative of the foregoing with respect to $vec(H)$. A similar procedure (now differentiation with respect to $vec(W)$ would be required for the other off-diagonal block) would yield the other off diagonal block. Once the off diagonal blocks were determined, the step size for the full problem could be set to the inverse of the 2-norm of the matrix of block derivatives

$$\begin{bmatrix} \frac{\partial^2 f}{\partial vec(W)^2} & \frac{\partial^2 f}{\partial vec(H) \partial vec(W)} \\ \frac{\partial^2 f}{\partial vec(W) \partial vec(H)} & \frac{\partial^2 f}{\partial vec(H)^2} \end{bmatrix} ,$$

where f is the smooth part of the objective function. Unfortunately, I wasn't sure how do evaluate $\frac{\partial vec(H^T H)}{\partial vec(H)}$, and so I was not able to go any further. Instead, I resorted to setting the step size manually.

2.2 Visualizing W

Below, I display the images that resulted from each of the three different algorithms. For each algorithm, each image is one of the columns of W (reshaped into a 32×32 image)

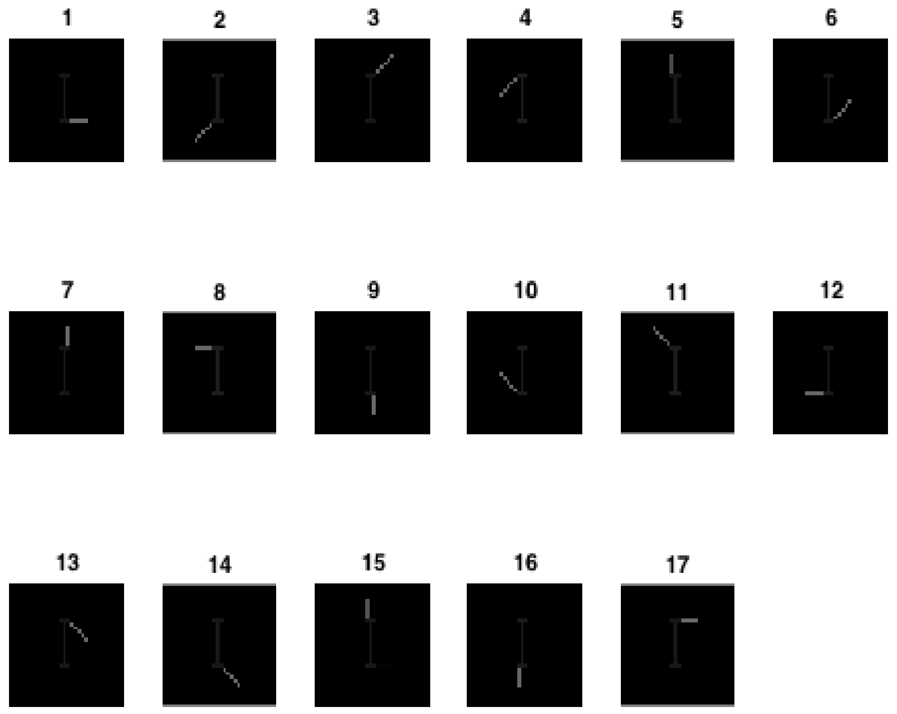

Figure 2: Visualization of Columns of W for Full Projected

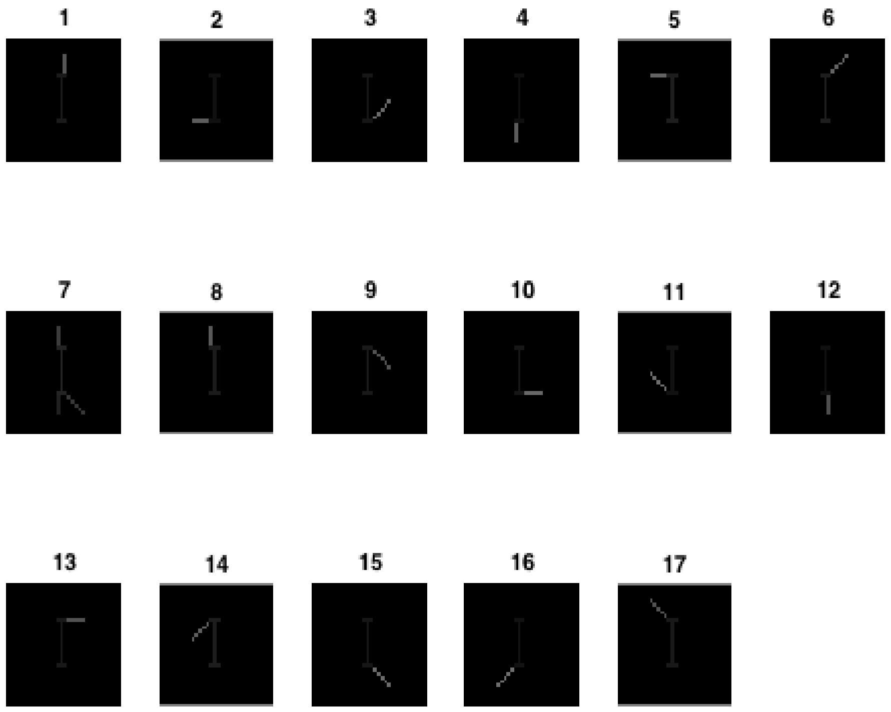

Figure 3: Visualization of Columns of W for Alternating Minimization

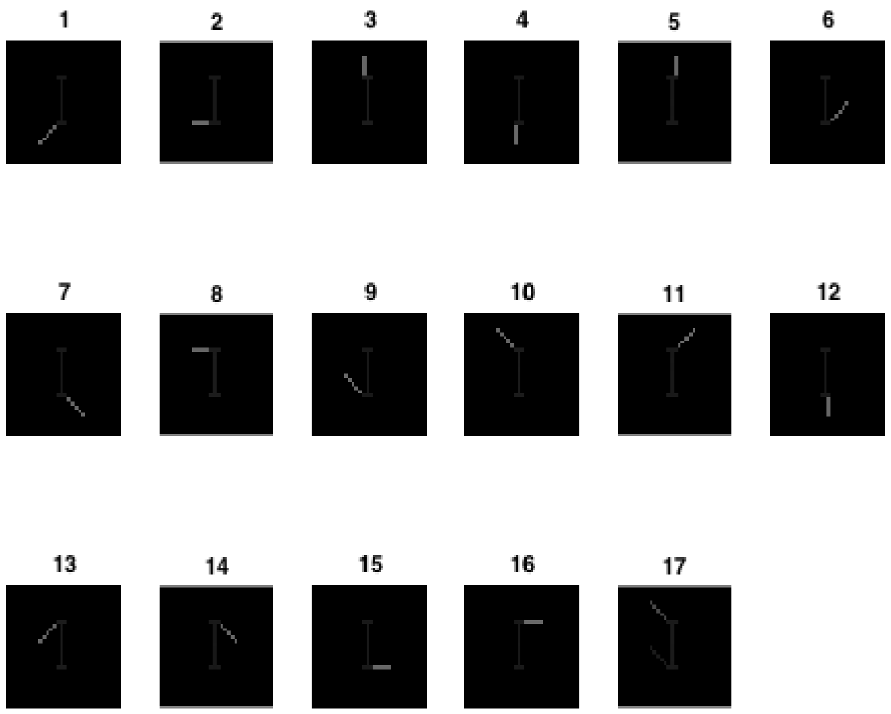

Figure 4: Visualization of Columns of W for Alternating Projection

We see that, in all three algorithms, each column of W corresponds to a salient aspect of the swimmer images.

2.3 Results

Below, I summarize my results for non-negative matrix factorizaion. maxits is the number of iterations performed. Note that the step size for the alternating algorithms is NA because the lipschitz constant changes at each outer iteration, and depends on the block to which we are minimizing with respect to. Specifically, for finding $W^{(k)}$, we set the step size to $1/||(H^{(k-1)})^T H^{(k-1)}||$, and for $H^{(k)}$ we set the step size to $1/||(W^{(k)})^T W^{(k)}||$.

Table 1: Non Negative PCA Results

Algorithm	Step Size	maxits	Frob. Err.
Full Proj.	1e-3	10000	1.0887
Alt. Min.	NA	200	0.0030
Alt. Proj.	NA	800	0.0030

2.4 Discussion

When interpreting the results in Table 1, it is important to to keep in mind the difference between iteration count and run time. In particular, alternating minimization is takes a lot of time to perform each iteration because it needs to fully solve two optimization problems in each iteration, whereas alternating projected merely takes two steps; in directions dictated by the gradient with respect to the different block coordinates. The table above is slightly misleading if one only looks at the iteration count, because it might lead to the conclusion that alternating minimization is "faster" than alternating projection. The truth is that alternating projected, though it took 600 more iterations to converge to the same objective value, finished running in a fraction of the time that it took alternating minimization. Table 1 demonstrates the significant impact of not choosing the step size appropriately. The algorithm much longer to converge, and achieved poor performance.

3 Robust PCA

Principal Component Aanalysis (PCA) is notoriously sensitive to noisy data. The robust PCA algorithm was first developed as a way of addressing this issue. The algorithm name is actually a bit of a misnomer, as it doesn't perform any sort of principal component analysis. It merely factors a data matrix X into the sum of a low rank matrix L and sparse matrix S by imposing constraints and regularization on L and S. By factoring as such, we hope that any noise in the data will be banished to S and only important information will remain in L. Only then is standard PCA conducted on L. Thus, a perhaps better name for the algorithm would be "Denoising Data for More Robust Principal Component Analyses". At any rate, the robust PCA task can be formulated as

$$\underset{L,S}{\operatorname{argmin}} ||L||_* + \lambda||S||_1 + ||L + S - X||_F^2 \ ,$$

where $||\cdot||_*$ is the nuclear norm i.e. the sum of the singular values of $\cdot$ and $||\cdot||_1$ is the elementwise 1-norm i.e. the sum of the absolute values of the elements of $\cdot$. The nuclear norm regularization term promotes low-rankness in L, which we hope will result in L preserving the "most important" information in X, and the 1-norm term promotes sparsity in S, which we hope will result in the noise being sparsely banished to S. However, these two terms also make the above optimization problem non-smooth, so gradient descent on its own won't suffice. The idea is to take a gradient step on the smooth part and then compute the nuclear norm proximal operator followed by the 1-norm proximal operator. To this end, I implemented alternating proximal gradient descent and full proximal gradient descent.

3.1 Tuning Parameters and Results

In tuning my parameters, I kept in mind the fact that I wanted to encourage sparsity and low-rankness, but not to the point where the error between $L + S$ and X was enormous. I self-imposed the requirement that the error be at least less than 1. Thus, my parameter tuning goal was to have as much low-rankness and sparsity as possible subject to the hard constraint of a frobenius error less than 1. I found that the best way of acheiving this goal was setting λ and the initial β to very small values, and that the number of iterations was an extremely important parameter. This is because, due to the penalization term, we need to increase the value of β at each iteration in order to approach a feasible solution to the true optimization program. But, if we set the step size to the inverse of the lipshitz constant: $1/\beta$ for the alternating problem and $1/(2\beta)$ for the full problem, this step size also controls how strongly low-rankness is enforced at each iteration because we assume a fixed coefficient of 1 for the nuclear norm term. Thus, in the beginning, when β is small, low rankness is encouraged but the iterate is far from feasible, thus yielding a large frobenius error. But as β increases and we approach feasibility (low frobenius error), the emphasis on low-rankness decreases. Thus, more iterations yield lower error but higher rank of L. From the foregoing, it should be apparent that the rate at which β is increased at each iteration also plays an important role. The table below summarizes the best-performing values of the afore-mentioned parameters that I could find, and the subsequent figure shows how the terms of the objective function, and the objective function itself, evolved using these parameters. In Table 2, β_0 is the starting value of β, maxits is the number of iterations for which the algorithm was run, and expand is the constant by which β was multiplied after each iteration.

Table 2: Non Negative PCA Results

Algorithm	λ	β_0	maxits	expand	Frob. Err.	$rank(L)$	$\%nonzero(S)$
Alt. Min. Proximal	1e-3	1e-2	10	3	0.0065	2	27.65
Full Proximal	5e-3	1e-3	15	2	0.4756	61	25.03

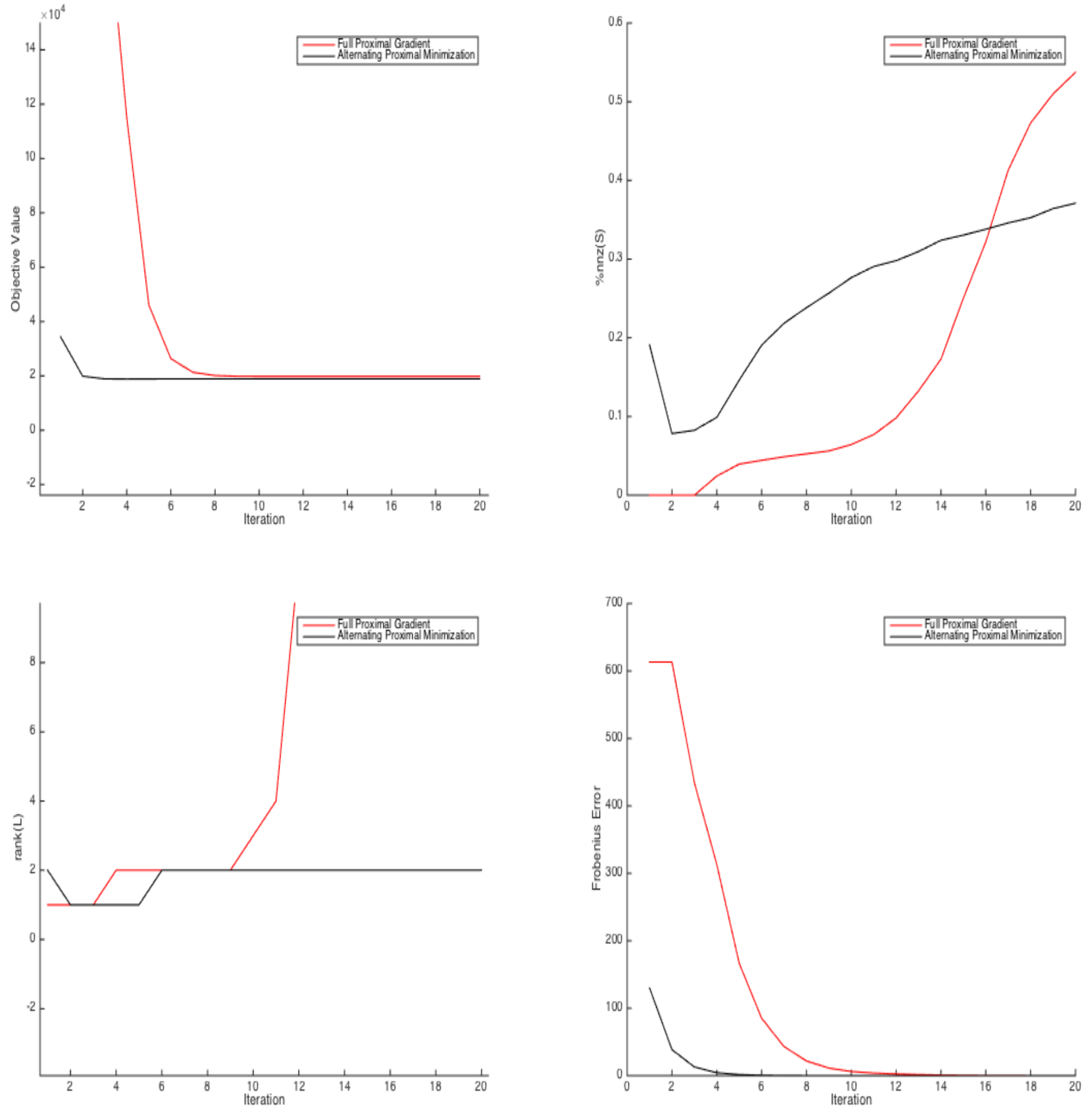

Figure 5: Monitoring Objective, Rank, Sparsity, and Error by Iteration (Note the Objective is a Linear Combination of the Other Three)

3.2 Discussion

As can be seen above, my best results were obtained by using a low λ, starting with a small β, and terminating the algorithm after few iterations. In the plots, we directly observe the algorithm progressively placing larger and larger penalization on the frobenius error, which quickly dominates the l1 and nuclear terms on account of the small value of λ and increasing β; respectively. Thus, only the very early iterations are responsible for low rankness and sparsity, and latter iterations are trying to reduce the error. Continuing the run the algorithm for many more iterations would result in a very good frobenius error, but with poor sparsity and low-rankness.

It should also be noted that the objective function for full proximal converges more slowly than for alternating. This is because, at each outer iteration of alternating, we are fully solving the optimization problem for one of the blocks, and thus make more progress (albeit greedily).

3.3 Visualizing L and S

Below I have displayed columns 5 and 28 of both the reshaped L and reshaped S. In both of these images, we see that the alternating algorithm did a better job of putting the noise in S and capturing the structure of the stagnant escalator in L.

Using implay to run the columns S one after the other, I have observed that the movement of the escalator conveyor belt in S is much more apparent in the alternating minimization than it is in full projected. Doing the same for L, I observed that the L from alternating minimization is almost unchanged for the entire frame sequence, whereas for full projected, we can see a subtle shadow of the the conveyer belt moving as we loop through the columns of L. This corroborates the results from Table 2, where we have seen that alternating projected achieved vastly better low-rankness and error, but only slightly worse sparsity than fully projected. To see the frame-by-frame animation, the reader should run the last two sections of the MATLAB source code "nnMatFac_RobPCA.m". Bear in mind, the last two sections can only be run after running some of the previous sections. See the comments in the code for details.

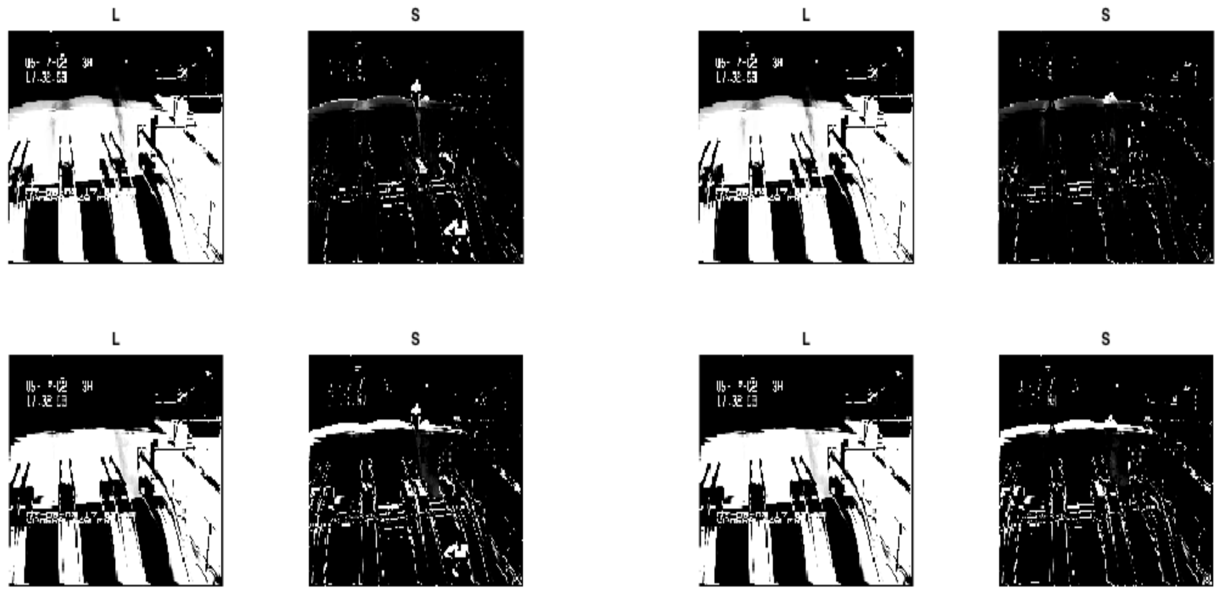

Figure 6: Columns 5 and 28 (left to right) of reshaped L and S. The top row is full project gradient, bottom row is alternating.