Albert-Ludwigs-Universität
Freiburg

A Neural Network for RPCA

A project in

Stochastic Machine Learning

Eric Brunner, Aron Distelzweig, Mirko Grimm, Bastian Schnitzer, Simon Beyer

December 20, 2020

Contents

1 Introduction

In Principal Component Analysis (PCA) we aim at finding the principal components of a set of data points. The principal components of a set of points in $\mathbb{R}^n$ are a sequence of n vectors. They are recursively defined as the i^{th} vector being the direction of a line that best fits the data while being orthogonal to the first $i-1$ vectors. The line is obtained through minimizing its average squared distance from the data points. Intuitively, one can think of PCA as fitting a p-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. Mathematically this corresponds to the eigenvectors of the datas' covariance matrix. Hence, the principal components form an orthonormal basis of the space $\mathbb{R}^n$. PCA is heavily used as an exploratory tool in data analysis. One big application is in quantitative finance (or generally time series analysis), where one is interested in the principal components of the empirical covariance matrix of certain financial assets. Focusing only on the largest principal components, which still hold the most important information about the data, leads to a dimensional reduction of the large dataset such that an efficient analysis of the data is still feasible.

In this project we specifically look at a modification of PCA, which is the Robust Principal Component Analysis. Here our aim is to recover a low rank Matrix L from, possibly highly, corrupted matrices M, in the sense that some matrix entries are faulty, for example through imprecise measurements. More precisely, we start with a data matrix (or its covariance matrix) $M \in \mathbb{R}^{m\times n}$ with corrupted entries and attempt to find a decomposition into a sum $M = L + S$. L is a matrix of low rank while the corrupted entries are filtered out into a sparse matrix S (which means that a lot of entries are zero). In this project we will restrict to the RPCA of symmetric positive semi-definite matrices M, which are for example given as empirical covariance matrices.

The usual state-of-the-art algorithms for (R)PCA are computationally demanding and, hence, impractical in some applications like finance, where instantaneous calculation might be required. Therefore, the use of a neuronal network is seen to yield a valuable alternative for the design an efficient decomposition tool. In this project our aim is to implement the algorithm "Denise" (see [**herrera2020denise**]) that aims at solving the robust PCA for semi-definite matrices through direct learning of the decomposition map $M \mapsto L + S$ via a deep neural network. We train Denise on a randomly generated synthetic dataset and evaluate the performance of the deep neural network on synthetic and real-world covariance matrices. The advantage of a neural network compared to the standard methods is that once trained the neural network delivers a function that outputs the desired decomposition of a dataset instantaneously, where using standard methods like convex optimization are computationally expensive. The algorithms are matrix specific (meaning that one has to use the algorithm for every new data), which is, therefore, very slow especially when data becomes large. For example in finance, one needs robust low rank estimation of covariance matrices of hundreds of assets instantaneously where the high performance of a neural network is very practical. As argued in [**herrera2020denise**], the results achieved by Denise are comparable to several state-of-the-art algorithms in terms of decomposition quality but outperforms all existing algorithms by computation time. As explained above, this is achieved by learning a single evaluation function that takes a matrix as an input and outputs the desired decomposition.

In section 2 the considered algorithm is explained in detail. Especially we introduce the objective function on which we train the neural network and describe the network architecture. Subsequently in section 3 we introduce the principal component pursuit (PCP) algorithm, which will be used as benchmark to asses the performance of the neural networks approach. We evaluate PCP as well as the trained Denise network on a Portfolio correlation matrix of five exemplary stocks in the DAX 30 and compare the resulting decompositions. Finally, in section 4, we summarize approaches on how to develop our project further, and which additional tasks we would like to carry out in the course of this semester.

2 Denise

We consider $\mathbb{S}_n \subseteq \mathbb{R}^{n\times n}$ to be the set of n-by-n symmetric matrices and $P_n \subseteq \mathbb{S}_n$ to be the subset of positive semi-definite matrices and $P_{k,n} \subseteq P_n$ the subset of matrices with rank at most k. As the input of our deep neuronal network we consider a Matrix $M = [M_{i,j}]_{i,j} \in P_n$. M is to be decomposed as a sum $M = L+S$ where $L = [L_{i,j}]_{i,j} \in P_{k,n}$ is of rank at most k and a sparse matrix $S = [S_{i,j}]_{i,j} \in P_n$. Since L is assumed to be symmetric, by the Cholesky decomposition, we can represent it as $L = UU^T$, where $U = [U_{i,j}]_{i,j} \in \mathbb{R}^{n\times k}$. Therefore M can be expressed as $M = UU^T + S$.

Loss function We input M in the neural network, which should faithfully output the low rank matrix U. Hence, we want to minimize the difference between M and UU^T, which is equal to S. A convenient choice of loss-function for the considered neural network is, therefore, given by the l_1-matrix-norm of S

$$\|UU^T - M\|_{l_1} = \|S\|_{l_1}$$

The l_1-norm is a common choice to guarantee sparsity of S.

Architecture As the matrix M is symmetric, we can reduce the input from n^2 to $n(n+1)/2$ by taking the triangular lower matrix of M. The lower matrix is then transformed into a vector using the operator h:

$$h : S^n \to \mathbb{R}^{n(n+1)/2},\ M \mapsto (M_{1,1}, M_{2,1}, M_{2,2}, \ldots, M_{n,1}, \ldots, M_{n,n})^T$$

Similarly we convert the output vector of the neural network into a matrix with the operator g defined as

$$g : \mathbb{R}^{nk} \to \mathbb{R}^{n\times k},\ X \mapsto \begin{pmatrix} X_1 & \cdots & X_k \\ \vdots & & \vdots \\ X_{(n-1)k+1} & \cdots & X_{(n-1)k+k} \end{pmatrix}$$

Besides the output layer, our multi-layer feed-forward neural network $\mathcal{N} : \mathbb{R}^{n(n+1)/2} \to \mathbb{R}^{nk}$ has three hidden dense layers, each exhibiting ReLU-activation function and $n/2$ nodes.

Using h and g the matrix U is the output of the neural network $U = g(\mathcal{N}(h(M)))$ and we get the desired matrix $L = \rho(\mathcal{N}(h(M)))$ for

$$\rho : \mathbb{R}^{rd} \to P_{r,d}, X \mapsto g(X)g(X)^T$$

Generation of training data For the training of Denise a synthetic dataset, consisting of randomly generated matrices with appropriate decomposition properties, is created. In detail, we construct a sample of positive semidefinite n-by-n matrices M that can be decomposed as

$$M = L_0 + S_0$$

where L_0 is a known matrix of rank $k_0 \leq n$ and S_0 a known matrix with a given sparsity s_0. Here, we undestand a *sparse matrix* as a matrix containing a lot of zeros and define the ratio between the number of zero-valued entries and the total number of entries as its *sparsity*. In the process of data generation, we follow a reverse approach by constructing first the matrices L_0 and S_0 with the required properties and merge it together to a matrix M which has by definition the desired decomposition.

For the construction of the low-rank matrix L_0, we collect nk_0 samples of independent standard normal random variables into an n-by-k_0 matrix U and set $L_0 = UU^T$. This construction scheme guarantees symmetry and positive semidefiniteness of L_0 as well as rank $L_0 \leq k_0$.

To construct the symmetric positive semidefinite sparse matrix S_0, we first take a sample of a uniformly random pair (i, j) with $1 \leq i < j \leq n$ that defines four non-zero entries of an n-by-n matrix $\tilde{S}_0$. The off-diagonal elements (i, j) and (j, i) are set to value b as the realization of a uniformly random variable in $[-1, 1]$ while the diagonal elements (i, i) and (j, j) are set to value a as the realization of a uniformly random variable in $[|b|, 1]$. Due to its construction $\tilde{S}_0$ is positive semidefinite. Finally, we receive the matrix S_0 by summing the realizations $\tilde{S}_0^{(i,j)}$ over the pairs (i, j) until the pursued sparsity is reached.

According to the described approach, we define a class *SyntheticMatrixSet* which is, by its methods, able to create a randomly generated matrix M together with L_0 and S_0 as both parts of its decomposition $M = L_0 + S_0$ based on the userspecified arguments: dimension n, rank k_0 and sparsity s_0. This class enables the generation of highly extensive datasets for various training purposes.

3 Testing and Comparison

After the neural network has been trained with the synthetic data, as described in section 2, it can be tested on synthetic data, where the decomposition is known by construction, as well as on real world data. To compare the decompositions of real world data, another algorithm is needed for benchmarking, which calculates the decomposition of an arbitrary positive semidefinite matrix M into a low rank, symmetric and positive-semidefinite matrix L_0 plus a sparse matrix S_0:

$$M = L_0 + S_0. \tag{3.1}$$

With *Principal Component Pursuit* (PCP), there exists an algorithm, which, under some suitable assumptions, calculates the decomposition exactly via singular value decomposition (SVD) [**candes2009robust**]. The assumptions and the main ideas of this algorithm is presented in the first subsection. In the second subsection the results of the decomposition of portfolio correlation matrices via the AI algorithm Denise with the results of the Principal Component Pursuit algorithm.

3.1 The Minimization Problem solved by PCP

Let M be a given element of $\mathbb{R}^{n_1 \times n_2}$. $\|\cdot\|_*$ denotes the nuclear norm, i.e. the sum over the singular values of a matrix $\|M\|_* := \sum_i \sigma_i(M)$. $\|\cdot\|_1$ is the well known ℓ_1 norm $\|M\|_1 = \sum_{ij}|M_{ij}|$. The PCP algorithm solves the convex optimazation problem

$$\text{minimize} \quad \|L\|_* + \|S\|_1, \quad \text{where } L + S = M \tag{3.2}$$

exactly, if the the low-rank component L_0 fulfills a "incoherence" condition, and that the sparse component is "reasonably sparse". The meaning of this "incoherence" condition for L_0 and the "reasonable" sparsity of S_0 is explained in [**candes2009robust**]. We summarize the main points real for quadratic matrices:

(i) Let $U\Sigma V^\top$ the singular singular value decomposition of $L_0 \in R^{n\times n}$ with rank $k \geq n$, i.e.

$$L_0 = U\Sigma V^\top = \sum_{i=1}^{k} \sigma_i u_i v_i^\top, \tag{3.3}$$

where $U = (u_1, \ldots, u_k), V = (v_1, \ldots, v_k) \in \mathrm{O}(n)$, $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0) \in \mathbb{R}^{n\times n}$. $\sigma_1, \ldots, \sigma_k$ are the singular values and u_i and v_i, $i = 1, \ldots, k$, are the left-singular and right-singular vectors for σ_i, respectively. Then the matrix L_0 is called incoherent, with parameter μ, if

$$\max_i \|Ue_i\|^2 \geq \frac{\mu k}{n^2}, \quad \max_i \|Ve_i\|^2 \geq \frac{\mu k}{n^2}, \quad \left\|UV^\top\right\|_\infty \geq \frac{\sqrt{\mu k}}{n}. \tag{3.4}$$

e_i are the canonical basis vectors of $\mathbb{R}^n$.

(ii) The positions of the nonzero elements of the sparsity matrix are selected uniformly random.

If (i) is fulfilled, the matrix L_0 is considered as not sparse. With (ii) we try to prevent, that the nonzero elements are only in one, or few columns of the sparsity matrix. For example if the entries of S_0 except the first column are all zero, and the first column of S_0 is the negative of the first column of L_0, then it is impossible to recover the low rank component and sparse component exactly. To avoid, such variety of possibilities for the decomposition (ii) is a reasonable assumption.

3.2 Evaluation of trained Denise on Financial Data

To get a first impression of Denise performance on real-world data, we apply Denise on a 5-by-5 covariance matrix Σ of five stocks out of DAX 30, namely Allianz, BASF Bayer, Beiersdorf and BMW. The empirical covariances are calculated based on closing prices of the aforementioned stocks within the last 6 months starting from December 11th in 2020. In total, we included the prices of 128 trading days in our calculation.

The stock prices required to calculate the empirical covariance matrix are downloaded from *Yahoo! Finance* and then transferred into a CSV file. Then the CSV file is imported into *Jupyter Notebook* in order to calculate the empirical covariance matrix Σ according to

$$\Sigma_{i,j} = Cov(X^{(i)}, X^{(j)}) = \frac{1}{n-1}\sum_{k=1}^{n}(X_k^{(i)} - \bar{X}^{(i)})(X_k^{(j)} - \bar{X}^{(j)}) \tag{3.5}$$

where $X_k^{(i)}$ is the price of the i-th stock at the end of the k-th trading day and n describes the total number of analyzed trading days within the relevant time period.

3.3 Decomposition of Portfolio Correlations from DAX 30 with PCP and Denise

For test purposes we applied the PCP algorithm to the empirical covariance matrix based on the prices of five share certificates of the companies Allianz, BASF Bayer, Beiersdorf and BMW of the last six months. The following covariance matrix is obtained

$$\begin{aligned}&(\mathrm{Cov}(x_i, x_j))_{i,j} \\ &= \begin{pmatrix} 142.67041515 & 28.06338926 & 30.56147946 & 2.52487121 & 31.18558268 \\ 28.06338926 & 14.54671933 & -10.75409144 & -2.97700557 & 20.0735403 \\ 30.56147946 & -10.75409144 & 65.84451884 & 11.30075662 & -28.04105723 \\ 2.52487121 & -2.97700557 & 11.30075662 & 10.38498412 & -5.84017695 \\ 31.18558268 & 20.0735403 & -28.04105723 & -5.84017695 & 33.39091805 \end{pmatrix},\end{aligned} \tag{3.6}$$

where $x = (x_1, \ldots, x_5) =$ (Allianz, BASF Bayer, Beiersdorf, BMW). The PCP algorithm returns the decomposition

$$L_0 = \begin{pmatrix} 33.15288862 & 19.03429426 & -2.38353047 & 2.5249416 & 24.04138395 \\ 19.03429426 & 14.54671744 & -10.7540876 & -2.97707117 & 20.07354269 \\ -2.38353047 & -10.7540876 & 24.51612531 & 11.30069593 & -17.99311351 \\ 2.5249416 & -2.97707117 & 11.30069593 & 5.60790277 & -5.84023382 \\ 24.04138395 & 20.07354269 & -17.99311351 & -5.84023382 & 28.30038404 \end{pmatrix}, \tag{3.7}$$

$$S_0 = \begin{pmatrix} 109.51752654 & 9.029095 & 32.94500994 & 0. & 7.14419873 \\ 9.029095 & 0. & -0. & -0. & -0. \\ 32.94500994 & -0. & 41.32839353 & 0. & -10.04794373 \\ 0. & -0. & 0. & 4.77708135 & 0. \\ 7.14419873 & -0. & -10.04794373 & 0. & 5.09053401 \end{pmatrix}. \tag{3.8}$$

The rank of L_0 is 2. This is obviously a exact decomposition $(\mathrm{Cov}(x_i, x_j))_{i,j} = L_0 + S_0$.

4 Outlook

In this section we give a brief outlook on some ideas which we like further devolop in the course of this project.

4.1 Convolutional Network

It is clear that the described neural network ansatz (see section 2), once trained yields a very fast and robust methodology to solve the RPCA problem, compared to state-of-the-art algorithms (as for example the PCP discussed in 3.1). However, the training and the generation of training data can be computationally quite demanding for large matrices. One the one hand, larger matrices require much larger training data sets for obtaining reasonable generalization properties. On the other hand, the necessary network parameters in the dense network implementation will also scale up. To circumvent this scaling problem (at least on the training side) we would like to compare the current dense network topology to a more sparse one, building on convolutionary layers. These architectures, typically, exhibit much fewer trainable parameters then dense networks.

Convolutionary networks are usually used in image recognition. In such scenarios one typically expects strong correlations within the data (pixels of the images), since usually some red pixels will be most probable surrounded by further red ones. By exploiting these correlations, a convolutionary network (with much fewer parameters then a dense one) is often able to perform equally well while allowing for much faster training. Although the considered corrupted SPD-matrices at input might not boast a similar structure then images, the positive definite character also will lead to correlations between the individual matrix entries. Therefore, it seems reasonable to test a sparser convolutionary network architecture also in this setting.

4.2 RPCA for arbitrary matrices

As described in section 2, the RPCA of a symmetric positive semi-definite matrix M is given by decomposing

$$M = UU^T + S$$

with a sparse matrix S and a rank k matrix $U \in \mathbb{R}^{n\times k}$. It is clear that this approach not directly generalizes to arbitrary matrices such as data matrices, since UU^T is by definition positive semi-definite. However, any (also non-quadratic) rank k matrix $M \in \mathbb{R}^{n\times m}$ can be decomposed as $M = UV^T$ with $U \in \mathbb{R}^{n\times k}$ and $V \in \mathbb{R}^{m\times k}$. Therefore, the RPCA problem of arbitrary M is given by finding a sparse matrix S and rank k matrices U, V as above that fulfill

$$M = UV^T + S\,.$$

A first approach to this problem, based on neural networks, is to employ two collaborating networks, each similar to the network used so far. Each networks task is to learn the

mapping from input matrix M to low-rank k matrix U, respectively V, by minimizing the l_1-distance $\|M - UV^T\|_{l_1}$ to ensure (as before) sparsity of S. The training of both networks can be accomplished in an alternating manner. E.g. we first initialize V at random and train the first network to approximate U, while in a second step we fix the first network and its output U to train the second network on the same data to output V. Based on this (ideally better choice of V) we can now repeat this procedure up to a point where both networks converged up to a reasonable degree and stabilize. The collaborative training of both network will naturally take approximately two times as long as the single network employed so far and exhibits, hence, similar computational complexity.

4.3 Matrix Completion (Data with "Missing Entries")

In natural science one often is confronted with incomplete knowledge of the investigated systems in form of measurement errors and imprecision.

4.4 Data from Neuroscience