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 12, 2021

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). One contribution of our project is to generalize the network approach of RPCA for symmetric positive semi-definite matrices M, to the *robust singular value decomposition* (RSVD) of arbitrary (non-quadratic) 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 and generalize the algorithm "Denise" (see [1]) that aims at solving the robust PCA 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 [1], 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 Neural network approach to robust singular value decomposition

Mathematically, PCA is the eigenvalue decomposition of correlation-, respectively covariance-matrices, obtained from data. Such matrices M are always positive semi-definite. Therefore, they are diagonalizable $M = \tilde{U}\Lambda\tilde{U}^T$ with orthogonal matrix U and diagonal matrix $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$ having the eigenvalues of M on the diagonal. The columns of U are the eigenvectors of M. We can absorb the diagonal matrix Λ in $\tilde{U}$ and $\tilde{U}^T$ by multiplying each column of U with the square root of the corresponding eigenvalue $U_{i,j} = \tilde{U}_{i,j}\sqrt{\lambda_j}$ for all i, j, or equivalently $U = \sqrt{\Lambda}\tilde{U}$ with $\sqrt{\Lambda} = \mathrm{diag}(\sqrt{\lambda_1}, \dots, \sqrt{\lambda n})$. This results in the decomposition $M = \tilde{U}\sqrt{\Lambda}\sqrt{\Lambda}\tilde{U}^T = UU^T$. Note that usually U is not orthogonal, anymore. On the other hand, from a given decomposition $M = UU^T$ of a positive semi-definite M we can always obtain the eigenvalues λ_i of M as the square of the 2-norm of the columns of U and by this obtain the ordinary eigenvalue decomposition $M = \tilde{U}\Lambda\tilde{U}^T$ with $\tilde{U}_{i,j} = U_{i,j}/\sqrt{\lambda_j}$.

The above, obviously, only works for square-matrices which are additionally positive semi-definite. However, for *any* matrix M, given for example a $n \times m$ data-matrix of m observations of n features, we can calculate the singular value decomposition (SVD) $M = \tilde{U}(\Lambda, 0)\tilde{V}^T$ (here, we assume $m \geq n$; the other case is very similar). U is a orthogonal $n \times n$-matrix, V is an orthogonal $m \times m$ matrix and $(\Lambda, 0)$ is a diagonal rectangular $n \times m$ matrix with $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$ in the first n columns and 0 everywhere else. The columns of $\tilde{U}, \tilde{V}$ are the left, respectively right, singular vectors of M and $\lambda_1, \dots, \lambda_n$ are its singular values. As above in the case of the eigenvalue decomposition, the SVD of M is equivalent to the decomposition $M = UV^T$ with $n \times n$-matrix $U = \tilde{U}\sqrt{\Lambda}$ and $n \times m$-matrix $V^T = (\sqrt{\Lambda}, 0)\tilde{V}^T$. Note that the SVD is not unique, but is always exists. Moreover, the eigenvalue decomposition is a special case of the more general SVD which, if applicable, is in fact unique up to permuting the eigenvectors in U.

In the following we describe an approach based on neural networks (recently put forward in [1] under the name *Denise*), to obtain the *robust* eigenvalue decomposition (i.e. RPCA) of positive semi-definite matrices. Additionally, we generalized this approach to the *robust* singular value decompostition (RSVD) of arbitrary (not even quadratic) matrices. This decomposition should be of low rank $k < n$ and should be able to be robust against corruptions of a given (data) matrix M. This means that we want to restrict to an approximate decomposition $M \approx UV^T + S$ with $n \times k$-matrix U and $m \times k$-matrix V. $L = UV^T$ is the low rank k estimate of M, while S is included to filter out the corrupted matrix-entries of M. In this approximation, the columns of U, V approximate the k left, respectively right, singular vectors corresponding to the k largest singular values of M. If the remaining $n - k$ singular values are indeed very small in comparison with those, then the rank-k decomposition is justified an yields a valid approximation. We tackle the rank-k decomposition $M \approx UV^T$ by learning the map from $M \mapsto U, V$ by a neural network trained on a suitable synthetic dataset $(M_i, (U_i, V_i))_i$ of such matrices, as described in detail subsequently.

2.1 Robust eigenvalue decomposition of positive semi-definite matrices

We start with the decomposition of symmetric positive semi-definite matrices M, as a special case. 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

$$\|UU^T - M\|_{\ell_1} = \|S\|_{\ell_1}$$

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:

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

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 the Denise network, 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.

2.2 Robust singular value decomposition of arbitrary matrices (RSVD)

In the preceding section, we aimed at learning the mapping $M \mapsto U$ for the decomposition $M = UU^T$ by means of a neural network through minimizing the loss $\|UU^T - M\|_{\ell_1}$. To obtain the low rank-k ($k < n$) SVD $M = UV^T$ of a arbitrary $n \times m$-matrix M into $n \times k$-matrix U and $m \times k$-matrix V we proceed in a similar fashion. However, in this case we rely on a *collaborative network approach* in which we train two neural networks $\mathcal{N}_U$ and $\mathcal{N}_V$, one for each mapping $M \mapsto U, M \mapsto V$, by minimizing the loss

$$\|UV^T - M\|_{\ell_1} .$$

The ℓ_1 norm results is used, as above, to force sparsity of the difference $S = UV^T - M$. Obviously, the loss function depends on the predictions of both neural networks. After training, the prediction of the combined network for a given input M_{truth} is given by

$$M_{\text{pred}} = \mathcal{N}_U(M_{\text{truth}})\mathcal{N}_V(M_{\text{truth}})^T, .$$

Training the networks parallel at the same time is computationally expensive for large matrices M. Therefore, we propose to train both networks in an alternating manner, through successively optimizing the weights of one network by keeping the weights of the others fixed, and vice versa. This procedure must be repeated for several iterations to obtain a reasonable prediction accuracy. This training design, furthermore, utilizes a reasonable balancing between exploring the loss landscape in which we minimize, and convergence to a suitable minimum.

Architecture The architectures of both networks $\mathcal{N}_U, \mathcal{N}_V$ are chosen similar to the network $\mathcal{N}$ of the previous section. We choose three hidden dense layers, densely connected to the output layer of dimension $n \times k$, respectively $m \times k$. The first and third hidden layer have $n * m$ nodes, each, while the second layer has $n * m/2$ nodes. Compared to the decomposition of positive semi-definite matrices above, we are not allow to restrict to the lower triangular part of M as input for the network, since M is not necessary symmetric (not even quadratic).

Generation of training data The training data is generated similar to above. However, here we generate U and V by sampling their components from independent standard normal distributions. The sparse matrix S, representing possible corruptions of the data, is at first initialized as $n \times m$-matrix with zero entries everywhere. Subsequently we uniformly choose entries (i, j) and filling these with random real number following a uniform distribution on $[-1, 1]$. This es repeated until the fraction of non-zero entries $r/n * m > 0.05$ (r is the number of non-zero entries) arrives at a value larger then a pre-defined threshold (we choose here 0.05). From U, V, S we generate the matrix $M = UV^T + S$, which will be given to the neural networks as training data. Differently then above, we do not need a positive semi-definite M here, hence we do not need to use a special generation procedure to enforce positivity of S.

Testing For a fist impression of the performance of the neural network SVD, we test the trained two-fold network $(\mathcal{N}_U, \mathcal{N}_V)$ on synthetic test-matrices generated by the same procedure as the training data. For testing, we choose 5×4-matrices and aim at a rank-2 decomposition $M = UV^T$ in 5×2-matrix U and 4×2-matrix V. We train the network on a training set of size 100000 (from which we split of 0.2 as validation dataset) for 20 iterations, where in each iteration each network part is trained for 5 epochs. This results in a total number of 200 epochs. We do this with batch-size 64. The progression of the loss-function and a metric, measuring the sparsity of $S = M - UV^T$, during training is depicted in Fig. 1 for, both, the training and validation data. The trained neural network is then tested on three random test matrices M, generated as described above. The results are shown in Fig. 2. The neural network prediction $L = \mathcal{N}_U(M_{\text{truth}})\mathcal{N}_V(M_{\text{truth}})^T$ are quite

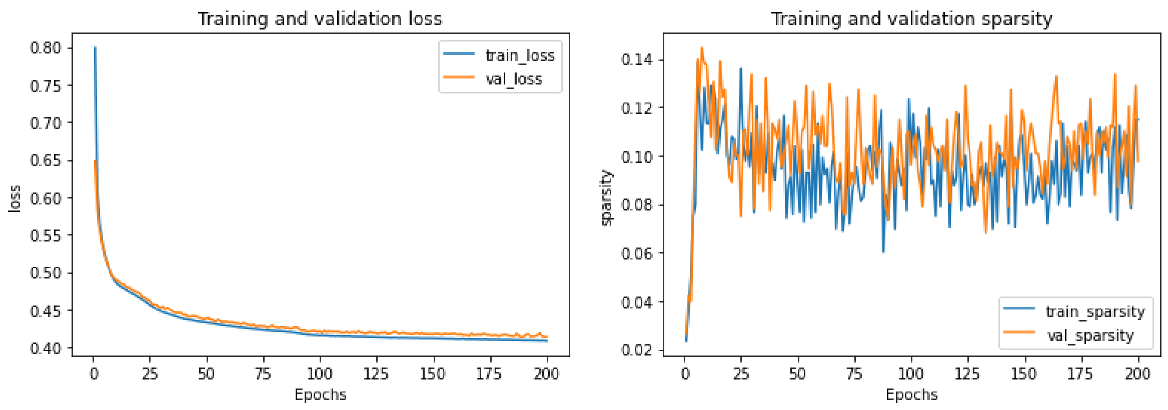

Figure 1: Evolution of loss and sparsity-metric on the training and validation data during training over $20 \times 5 \times 2$ epochs (see main text for explanation)

accurate approximations of actual input matrices M_{truth}, as one can observed from the matrix-plots. Also the matrices $S = M - L$ are relatively sparse.

This first test, shows the general capabilities of the presented neural network approach to calculate the robust SVD of arbitrary matrices M. However, in the following, we will mostly restrict to the simpler robust low-rank decomposition $M = UU^T$ of a positive semi-definite input matrix M. This is a very common situation in practice and of great relevance, for example in the PCA of correlation matrices.

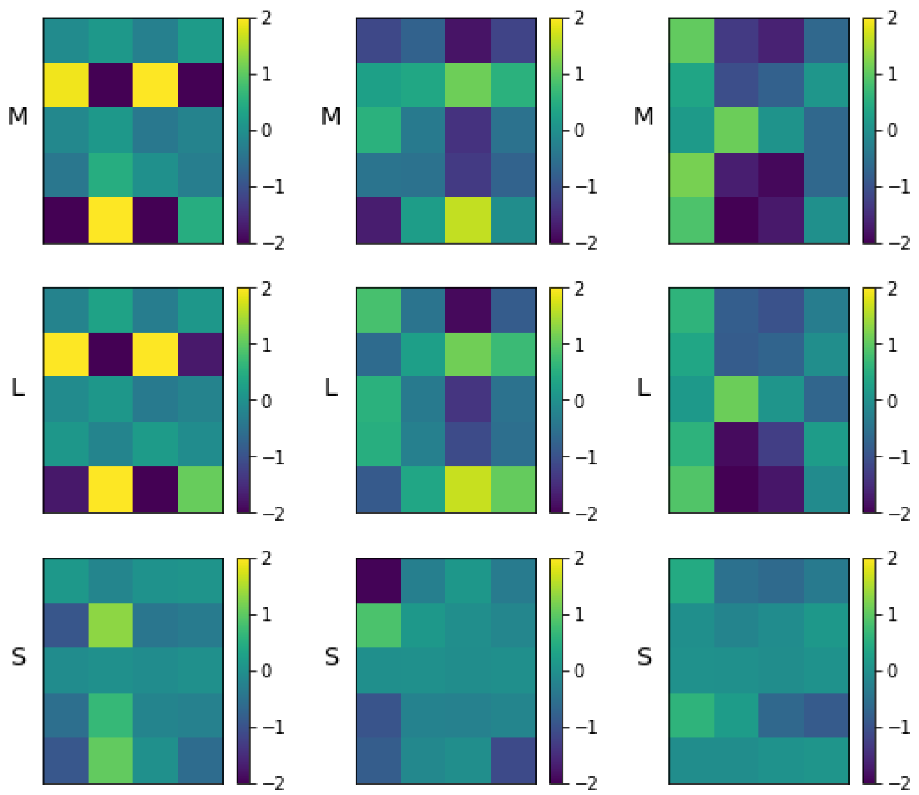

Figure 2: Testing the $M = L + S$ decomposition (with $L = UV^T$) for U, V calculated from the trained combined neural network $(\mathcal{N}_U, \mathcal{N}_V)$.

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) [2]. 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 [2, subsection 1.3]. 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 $U = (u_1, \dots, u_k), V = (v_1, \dots, v_k) \in \mathrm{O}(n)$, $\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_r, 0, \dots, 0) \in \mathbb{R}^{n\times n}$. $\sigma_1, \dots, \sigma_k$ are the singular values and u_i and v_i, $i = 1, \dots, k$, are the left-singular and right-singular vectors for σ_i, respectively. Then the matrix L_0 is called incoherent, with parameter μ, if

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

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 PCP Algorithm

In this subsection, a brief description of the PCP algorithm, which we use for comparison with Denise, is given. There are different strategies to solve the problem (3.2) numerically. As described in [2], we consider an *augmented Lagrange multiplier*. This is why Candes et al. named this method the ALM method. (3.2) is equivalent to the minimization of the following *augmented Lagrangian*

$$\mathcal{L}(L, S, Y) = \|L\|_* + \lambda \|S\|_1 + \langle Y, M - L - S\rangle + \frac{\mu}{2} \|M - L - S\|_F^2 . \tag{3.5}$$

Here $\langle \cdot, \cdot \rangle$ is defined as $\langle A, B \rangle = \operatorname{tr}\left(A^\top B\right)$, with real quadratic matrices A, B. $\|\cdot\|_F$ is the Frobenius norm. One can show, that

$$\arg\min_S \mathcal{L}(L, S, Y) = \mathcal{S}_{\lambda\mu}(M - L + \mu^{-1}Y), \tag{3.6}$$

$$\arg\min_L \mathcal{L}(L, S, Y) = \mathcal{D}_{\mu}(M - S - \mu^{-1}Y), \tag{3.7}$$

where $\mathcal{S}_\tau : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n} : (X_{ij})_{ij} \mapsto (\operatorname{sgn}(X_{ij}) \max(|X_{ij}| - \tau, 0))_{ij}$, is the extension of the shrinkage operator in $\mathbb{R}$ to $\mathbb{R}^{n\times n}$. $\mathcal{D}_\tau(X)$ is defined as $\mathcal{D}_\tau(X) = U\mathcal{S}_\tau(\Sigma)V^\top$, where $U\Sigma V^\top$ is the SVD of X. Hence, the following algorithm, taken from [2, p. 29], is productive

1. **Initialize**: $\mathrm{S}_0 = Y_0 = 0, \mu > 0$.

2. **While** not converged **do**

$$L_{k+1} = \mathcal{D}_\mu(M - S_k - \mu^{-1}Y_k) \tag{3.8a}$$

$$S_{k+1} = \mathcal{S}_{\lambda\mu}(M - L_{k+1} + \mu^{-1}Y_k) \tag{3.8b}$$

$$Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1}) \tag{3.8c}$$

3. **Return**: L,S.

With the calculations of the second step, it is avoided to solve a sequence of convex programs. To archieve good relativ accuracy, only a few iteration steps are neccessary [2, section 3].

3.3 Evaluation of trained Denise on Financial Data

To get a first impression of Denise performance on real-world data, we apply Denise on a set of 10-by-10 correlation matrices of ten stocks out of DAX 30, namely Allianz, BASF, Bayer, Beiersdorf, BMW, Continental, Siemens, Merck, Daimler, VW. As a data basis, we exploit the price data of the aforementioned stocks from the last 2 years, starting from January 15th in 2021. The empirical correlation matrices are determined retrospectively with an offset of one day, while the correlations in time step n are calculated based on closing prices within the last 35 days. In the end, we obtain 472 correlation matrices $\mathbf{R}^{(n)}, n = 1, 2, ..., 472$ for further numerical tests by this approach.

The stock prices required to calculate the empirical correlation matrices are downloaded from *Yahoo! Finance* and then transferred into a CSV file. This CSV file is imported into *Jupyter Notebook* and the price data is stored in vectors $X^{(i)}, i = 1, ..., 10$, beginning with the latest value, in order to calculate the empirical correlation matrices $\mathbf{R}^{(n)}, n = 1, 2, ..., 472$ according to

$$\mathbf{R}^{(n)}_{i,j} = \frac{\sum_{k=n}^{n+N-1}(X_k^{(i)} - \bar{X}^{(i)})(X_k^{(j)} - \bar{X}^{(j)})}{\sqrt{\sum_{k=n}^{n+N-1}(X_k^{(i)} - \bar{X}^{(i)})^2 \sum_{k=n}^{n+N-1}(X_k^{(j)} - \bar{X}^{(j)})^2}}, \tag{3.9}$$

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

3.4 Portfolio Correlations from DAX 30 with PCP and Denise

For test purposes we applied the PCP algorithm on 472 empirical correlation matrices of the ten DAX 30 companies, mentioned before. Due to our approach in calculating the different correlation matrices, their structure changes quite slowly. Hence, we concentrate our investigations only on 10 matrices with equidistant time steps.

We train the neural network on a synthetic data set of 1000000 10×10-matrices of rank three. Training and validation loss are illustrated in Fig. 3. Fig. 4 shows the training and validation sparsity over the epochs. Unfortunately, the reached sparsity of density is not much higher than $\sim 2\%$. The sparsity, that is obtained in the orignal article, is around $\sim 99\%$ for the finacial data sets [1, Table 2.]. This is a strong hint, that there is still improvement potential for our neural network. A small training set, aswell as a unsuitable architecture, could responsible that our neural network is not able to reduce the correlation in the data on three main components. The comparison of the RPCA decomposition of the correlations matrices into rank 3 matrix L and sparse S for both methods is shown in Fig. 5. Representatively, we have choosen only one decomposition, because the color plots of the ten matrices are quite similar. This simularity allows us to restrict further investigations concerning the attributes of the decompositions on only one single matrix. As one can see, the decomposition obtained from our neural network suffers significant outliers on the diagonal elements in L as well as S, which is not apparent in the PCP decomposition. One explanation for the occurence of the outliers on the diagonal can be the structure of the traning matrices. The normalization of the correlation matrices causes, that the diagonal elements are all set to 1. Since this particular property is missing in the training data, the network could not process the diagonal elements accurately. Due to this outlier not much structure is visible in L obtained from the neural network. However, apart from the diagonal elements the S-matrices of both methods seem to agree to some approximation (respecting the corresponding color code). As a metric, to measure how well the two algorithms agree, we determine the relative errors $\epsilon_{\text{rel}}(S_{\text{PCP}}, S_{\text{NN}})$, $\epsilon_{\text{rel}}(L_{\text{PCP}}, L_{\text{NN}})$ as the relative l_2-distance

$$\epsilon_{\text{rel}}(A, B) = \frac{\|A - B\|_{l_2}}{\|A\|_{l_2}} \tag{3.10}$$

between $S_{\text{PCP}}, S_{\text{NN}}$, respectively between $L_{\text{PCP}}, L_{\text{NN}}$ (the PCP-, respectively neural-network-prediction for S and L):

$$\epsilon_{\text{rel}}(S_{\text{PCP}}, S_{\text{NN}}) = 3,234\,, \quad \epsilon_{\text{rel}}(L_{\text{PCP}}, L_{\text{NN}}) = 0,909\,. \tag{3.11}$$

The running time for the neural network calculation of the RPCA is $0,0197$ seconds. The PCP algorithm needed $0,131$ seconds. Hence, the neural network approach is approximately an order of magnitude faster.

3.5 Correlation matrix of personality features with PCP and Denise

Here we compare the performance of the RPCA of our neural network approach and the benchmark PCP algorithm on 25 personality self report items obtained from approx-

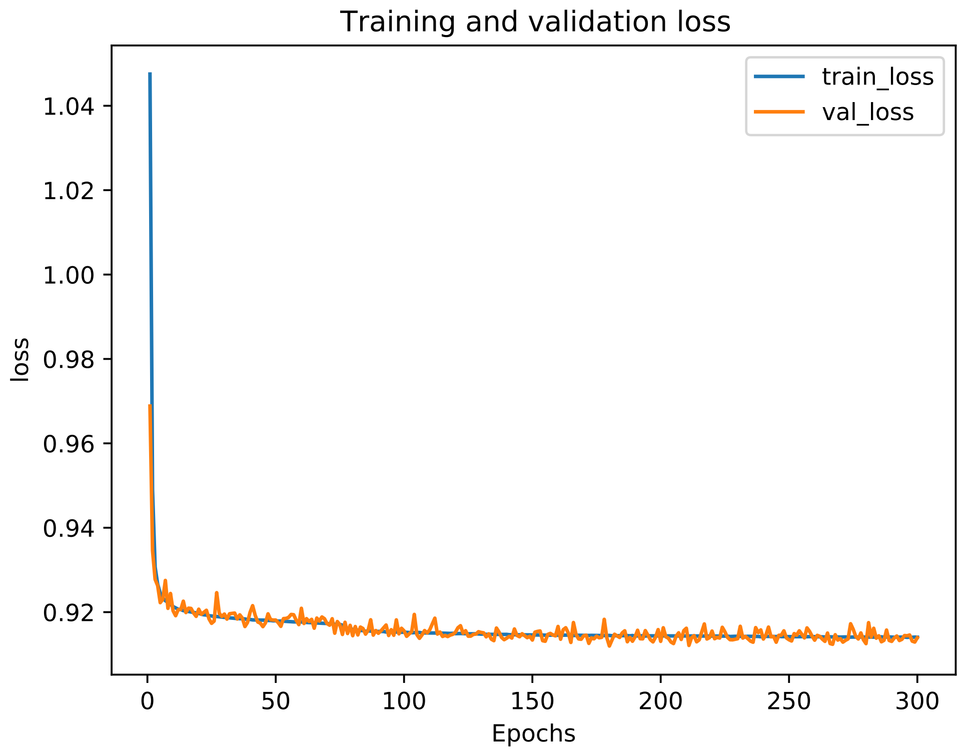

Figure 3: Training and validation loss of the network training for finance data.

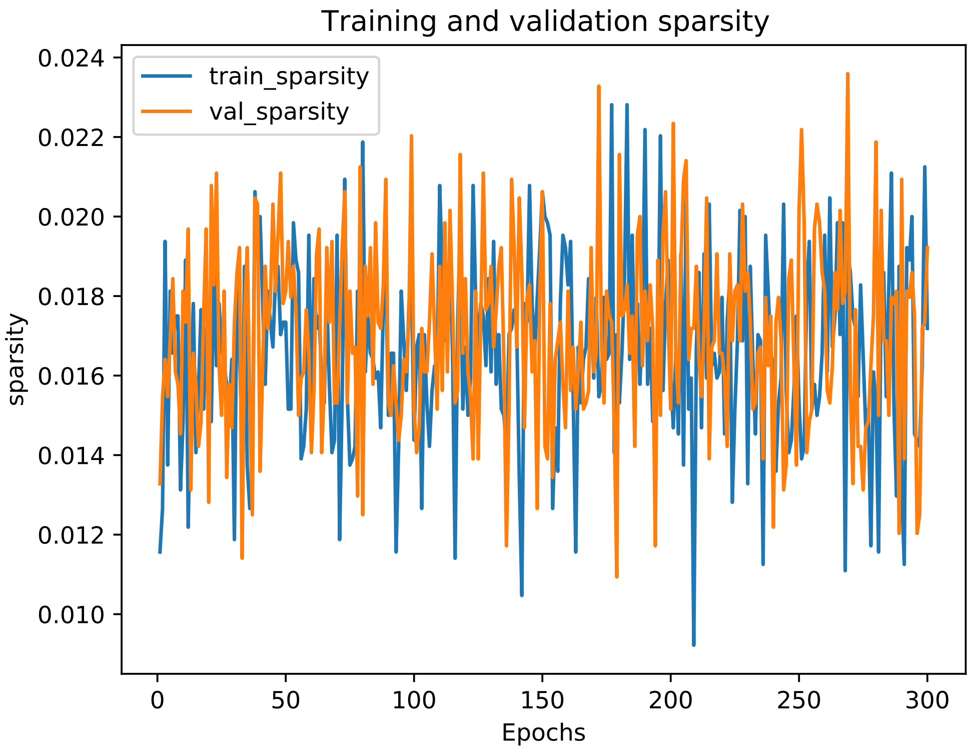

Figure 4: Training and validation sparsity evolution over the epochs.

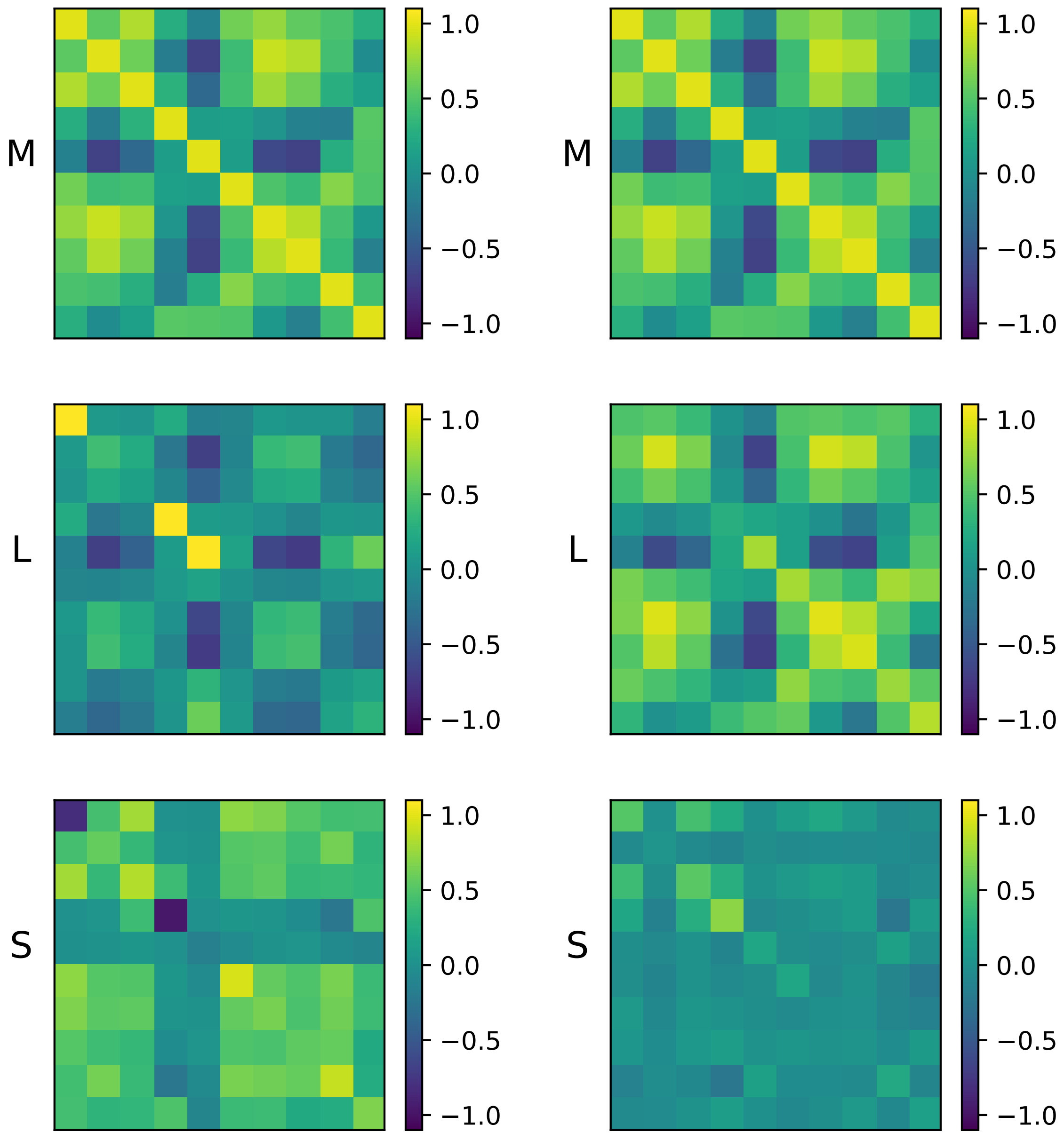

Figure 5: Comparison of RPCA on the empirical 10×10 covariance matrix based on the price development of 10 share certificates (see main text) from the DAX 30. The left panel shows the resulting decomposition $M = L + S$ obtained from the neural network approach, while the right panel shows the results from PCP.

imately 2800 individuals. The studied 25 personality features are grouped in the five categories agreeableness, conscientiousness, extraversion, neuroticism, and openness, see [personality-project]. The aim of PCA is to recover these five underlying putative factors from the data.

In this subsection we compare the performance of PCP and our neural network approach to obtain the RPCA-decomposition of the correlation matrix of the 25 personality features. The correlations are defined as the normalized empirical covariances between the variables x_i

$$\mathrm{Corr}(x_i, x_j) = \frac{\mathrm{Cov}(x_i, x_j)}{\sqrt{\mathrm{Var}(x_i)\mathrm{Var}(x_i)}} \,. \tag{3.12}$$

As in the previous subsection, we calculate the RPCA-decomposition $M = L+S$ by means of our neural network approach as well as by using the PCP method introduced in 3.2. The obtained low rank matrix L and the sparse part S containing corruptions of the input matrix are shown in Fig. 8. The low-rank ($k = 10$) is determined by the PCP-algorithm, and subsequently used in the definition of the neural network. The training performance of the neural network (training and validation loss) is depicted in Fig. 6. The evolution of the training and the validation sparsity can be seen in Fig. 7. The training was performed on 80000 synthetic 25×25 positive semi-definite matrices of rank 10.

As in the analysis of the financial data, the neural network decomposition exhibits very few strong outliers on the diagonal, which suppress the visual structure of the obtained matrices L, S to some extend. As before, these outliers are not present in the PCP results. Hence, the PCP-method seems to function more stable than the network ansatz. We quantify the discrepancy of the resulting decomposition of both approaches by their relative distances

$$\epsilon_{\mathrm{rel}}(S_{\mathrm{PCP}}, S_{\mathrm{NN}}) = 2{,}938 \,, \quad \epsilon_{\mathrm{rel}}(L_{\mathrm{PCP}}, L_{\mathrm{NN}}) = 1{,}820 \,. \tag{3.13}$$

The comparison of both methods, in combination with the striking results reported in [1], may be explained by a sub-optimal choice of network-architecture (to few nodes per hidden layer) of our network. Another source of instability might result from the small training sets or not ideally chosen hyper-parameters such as batch-size. This will be analyzed in detail in future work. Moreover, we will analyze, how we can use RPCA to obtain characteristic information about the studied data. For example, it will be interesting to investigate in which way we can recover the five underlying personality categories agreeableness, conscientiousness, extraversion, neuroticism, and openness from the principal components.

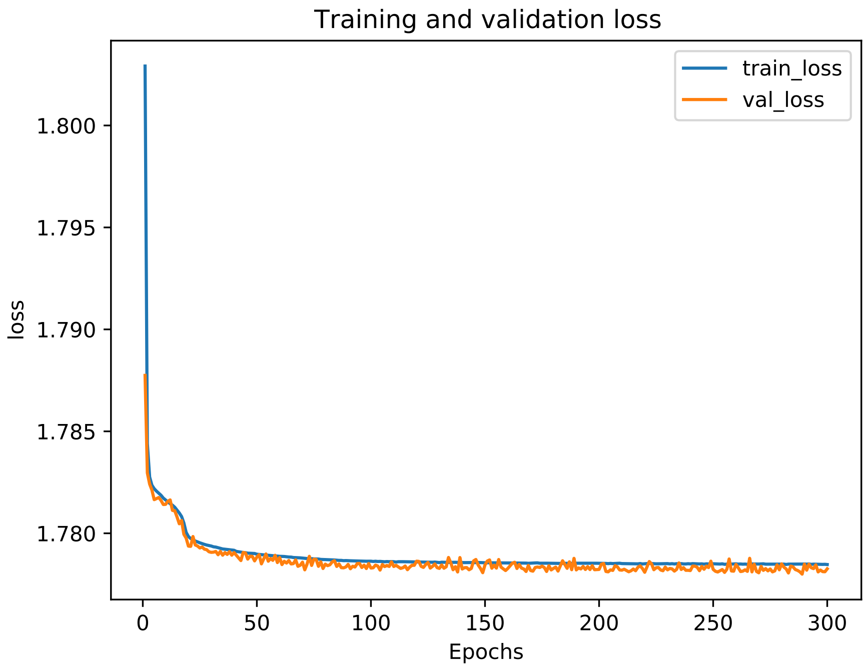

Figure 6: Training and validation loss of the training of our neural network for the personality data. Curiously, the validation loss is always smaller then the training loss.

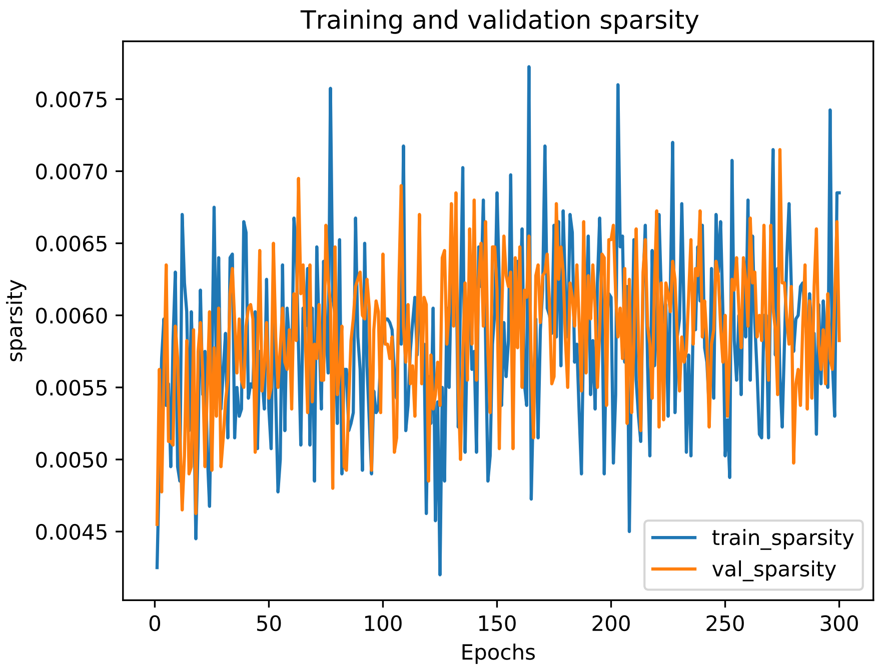

Figure 7: Training and validation sparsity evolution over the epochs.

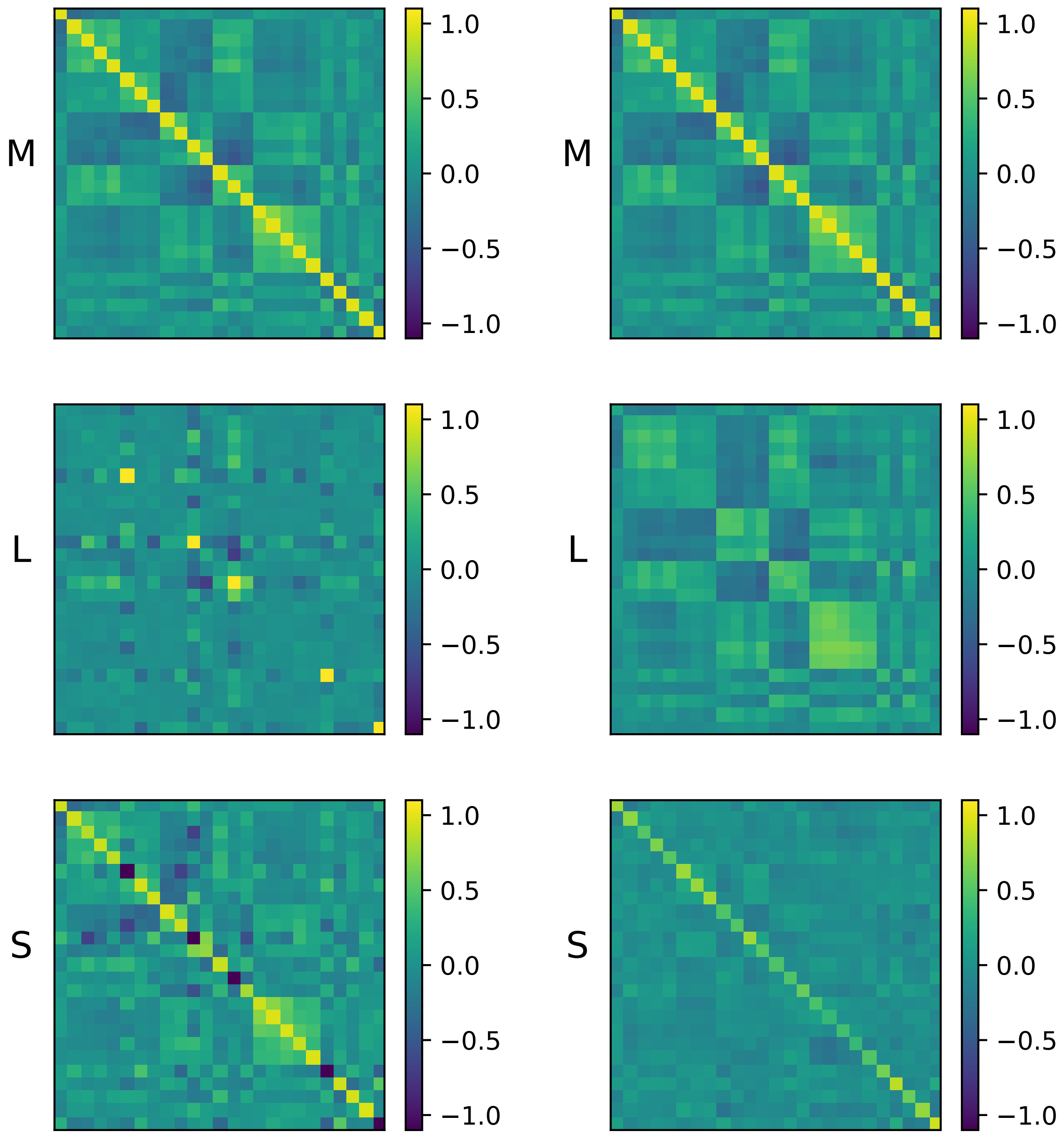

Figure 8: Comparison of RPCA of the empirical 25×25 covariance matrix of 27 personality features of roughly 2800 individuals (see main text). The left panel shows the resulting decomposition $M = L+S$ obtained from the neural network approach, while the right panel shows the results from PCP.

4 Conclusion and further work

In this project we studied a neural network approach for finding a low-rank approximation of a given, partially corrupted, positive semi-definite matrix M (e.g. a covariance matrix of observed data) into low-rank k matrix $L = UU^T$ and sparse matrix S containing the corrupted matrix-entries:

$$M = L + S\,.$$

This decomposition solves the problem of *robust principal component analysis*, which essentially breaks down to the search of k approximate eigenvectors of M describing most of the observed variance in the data. The problem is tackled by learning the map $M \mapsto U$ via a deep neural network, in the spirit of [1]. This approach proved to yield a fertile methodology to estimate the low-rank approximations L in comparison to state-of-the-art algorithms such as *Principal component pursuit.* Furthermore, if the network is already trained it out-performs these standard algorithms in terms of speed.

We further generalized the methodology of [1], to the *robust singular value decomposition* (RSVD) of an arbitrary (non-symmetric) matrix $M = UV^T + S$ into low-rank approximation $L = UV^T$ and sparsity matrix S containing the corrupted entries. This is accomplished through a *collaborative network* ansatz simultaneously training two networks, one to approximate each of the mappings $M \mapsto U, M \mapsto V$. Training both networks in an alternating manner, proved to yield a reasonable trade-off between exploration of the loss landscape and convergence to a minimum. The principal feasibility of this approach was demonstrated on synthetic test sets of small matrices.

In future work it would be very exciting to see how the RSVD performs on realistic data sets of large matrices. Singular value decomposition is one of the major tools in unsupervised machine learning protocols for pre-processing and compression of real world data. Hence, fast and reliable tools to solve this problem, which are additionally stable to noise and corruptions in the data, are of substantial interest in modern applications.

One drawback of the presented neural network approach to PCP, respectively SVD, is the need to know the low rank k by which one wants to approximate the decomposition previous to the design and training of the neural network. Typically, one justifies an effective lower rank $k < n$ of a $n \times n$-matrix by calculating eigenvalues and observing that only a few, say k, of them contribute substantially while the remaining eigenvalues are very small. It would be very interesting to explore possible network architectures to solve the RPCA similar to the presented methodology, but which do not require a predefined, target rank. One way would be to incorporate the rank of the approximation as an additional parameter to be learned. Learning in this automated way an *optimal* low rank which allows for a best low-rank estimate, would widen the range of possible applications significantly.

References

[1] Calypso Herrera et al. *Denise: Deep Learning based Robust PCA for Positive Semidefinite Matrices*. 2020. arXiv: 2004.13612 [stat.ML].

[2] Emmanuel J. Candes et al. *Robust Principal Component Analysis?* 2009. arXiv: 0912.3599 [cs.IT].