Fast Pseudoinverse Method for Sparse Feature Matrices

Numerical Linear Algebra course

Grigory Yaremenko, Iskander Akmanov, Kundyz Onlabek, Nina Aleskerova

December 2020

`https://github.com/yaremenko8/FAST_PSEUDOINVERSE`

Contents

1 Moore-Penrose crew

The distribution of the work between our team members is as follows:

- Grigory Yaremenko did the programming part and carried out the experiments
- Iskander Akmanov completed the presentation
- Kundyz Onlabek and Nina Aleskerova made this technical report

Besides, together we discussed the problem and the approach to handle it and recorded the video presentation.

2 Introduction

A **pseudoinverse** is a generalized inverse method for all types of matrices that play an important role in obtaining best-fit solutions to the linear systems even in cases where unique solutions do not exist. However, the applications of pseudoinverse have limitations to small data because of their high computational complexity. To be precise, the Moore-Penrose inverse is the most used pseudoinverse, and it is usually computed by utilizing Singular Value Decomposition (SVD). The time complexity of a full-rank SVD for an $m \times n$ matrix is $min(O(n^2m), O(nm^2))$ [1]. There are various techniques that were implemented to reduce the complexity, but they are not so practical and costs can still can be improved.

In our project we study the FastPI (Fast PseudoInverse) method of Jinhong Jung and Lee Sael (2020) [2]. It is a novel approximation algorithm for obtaining pseudoinverse efficiently and accurately based on sparse matrix reordering and incremental low-rank SVD. The main motivation under FastPi comes from the fact that in the real world many matrices are highly sparse and skewed.

The authors of the main acticle that we use conducted the experiments in real-world multi-label linear regression problems. They demonstrate that the method approximates the pseudoinverse of a large and sparse matrix faster and more efficiently than other methods without the loss of accuracy. We implemented their method of computing pseudoinverse.

3 Preliminaries

In this section the preliminaries on pseudoinverse and singular value decomposition (SVD), as well as formal definition of the problem and its applications will be described.

In most of machine learning models the representation of the training data is done by a feature matrix denoted by $A \in R^{m\times n}$, where m - the number of training instances and n - the number of features.

3.1 Pseudoinverse and SVD

Learning optimal model parameters often involves pseudoinverse $A^\dagger \in R^{n\times m}$. The Moore-Penrose inverse is the most accurate and widely used generalized matrix inverse that can be solved using SVD as follows.

Let A be decomposed as:

$$\mathbf{A} = \mathbf{U_{m\times r}\Sigma_{r\times r}V_{r\times n}^\top},$$

where $U_{m\times r}$ and $V_{r\times n}$ are orthogonal matrices and $\Sigma_{r\times r}$ is diagonal with r singular values.

If r is the rank of A, the pseudoinverse $A^\dagger$ is given by

$$\mathbf{A}^\dagger = \mathbf{V}_{n\times r}\mathbf{\Sigma}^\dagger_{r\times r}\mathbf{U}^\top_{r\times m}.$$

Otherwise, for a given target rank r it is the best approximation of pseudoinverse:

$$\mathbf{A}^\dagger \approx \mathbf{V}_{n\times r}\mathbf{\Sigma}^\dagger_{r\times r}\mathbf{U}^\top_{r\times m}.$$

The state-of-the-art low-rank SVD is randomized-SVD with the computational complexity of $O(mnlog(r) + (m+n)r^2)$. However, accurate approximations of pseudoinverses require relatively large rank approximations of SVDs and the costs of these computations are still too heavy.

3.2 Target Application of Pseudoinverse

The target application for the authors, **multi-label linear regression** based on pseudoinverse, is described as follows:

Given feature matrix $A \in R^{m\times n}$ and label matrix $Y \in R^{m\times L}$, where $m > n$, L is the number of labels, and each row of Y is a binary label vector of size L, the goal is to learn parameter $Z \in R^{n\times L}$ satisfying $AZ \simeq Y$ to estimate the score vector $\widehat{y} = Z^\top a$ for a new feature vector $a \in R^n$.

The linear system for unknown Z is over-determined when $m > n$; thus, the solution for Z is obtained by minimizing the least square error $\|AZ - Y\|_F^2$, which results in the closed form solution $Z = A^\dagger Y$. Hence, the approximate pseudoinverse of matrix A needs to be computed via SVD decomposition.

4 The method description

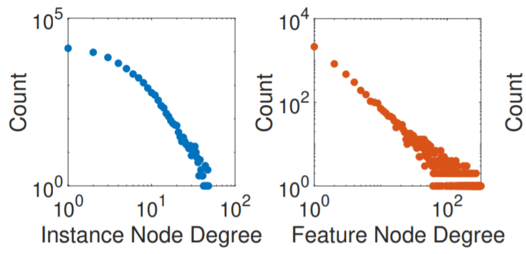
In this section we first describe theoretical inducement of the proposed method and then provide the algorithm for computing the pseudoinverse of a feature matrix.

The main ideas for accelerating the pseudoinverse computation are as follows:

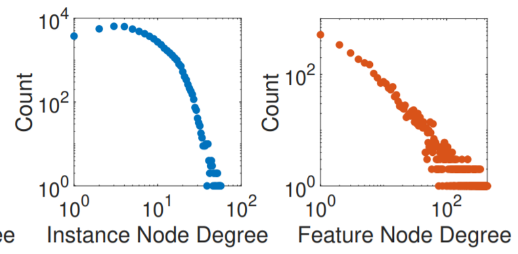
- Many feature matrices are highly sparse and skewed and they can be reordered in a way that its non-zero elements are concentrated at the bottom right corner leaving a large sparse area at the top left of the feature matrix.
- The reordered matrix is then divided into four submatrices, where one of the submatrices is the large and sparse rectangular block diagonal matrix, whose SVD can be obtained easily.
- The final SVD result of the feature matrix is efficiently obtained by incrementally updating the SVD result of the sparse submatrix

4.1 Graph representation of the feature matrix

As we already mentioned, in the real-world datasets the feature matrices are often highly sparse and skewed. This fact naturally leads to interpretation of $\mathbf{A}$ as a sparse bipartite network.
Definition (Bipartite Network from Feature Matrix) *Given $\mathbf{A} \in R^{m\times n}$, a bipartite network $G = (V_T, V_F, E)$ is derived from $\mathbf{A}$, where V_T is the set of instance nodes, V_F is the set of feature nodes, and E is the set of edges between instance and feature nodes. For each non-zero entry a_{ij}, an edge (i, j) is formed in G, where $i \in V_T$ and $j \in V_F$.*
An illustration of this relation between a matrix and a graph can be seen on the figure 3.
On the figure 1, the degree distributions of instance and feature nodes in each bipartite network derived from the Amazon and RCV feature matrices, respectively, is shown. One can notice that there are only a few high degree nodes that are called hubs, and the neighbor nodes of a hub are called spokes.



(a) The Amazon dataset



(b) The RCV dataset

Figure 1: Degree distributions of instance and feature nodes in a bipartite network derived from real-world feature matrices. There are few high degree nodes while the majority of nodes have low degrees, implying skewness on the degree distributions.

Real-world networks can be shattered by removing sets of highest degree nodes and the formation of small disconnected components by spokes whereas the majority of the nodes still would still be forming a giant connected component [3]. And this shattering property of real-world networks is also applied to bipartite graphs for the corresponding feature matrix reordering in Algorithm 1 of FastPI.

4.2 Matrix reordering

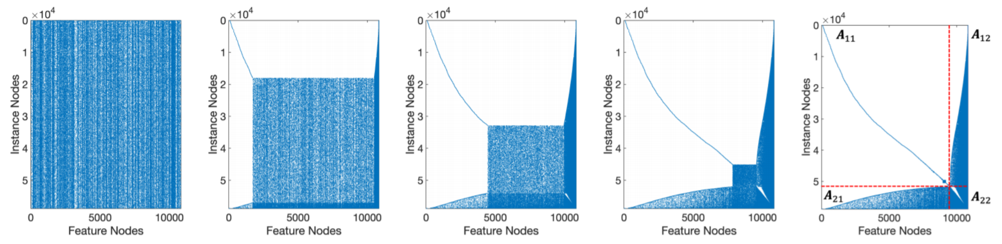
Given a bipartite network G derived from feature matrix $\mathbf{A}$, FastPI obtains permutation arrays $\pi_T : V_T \rightarrow \{1, \cdots, m\}$ for instance nodes and $\pi_F : V_F \rightarrow \{1, \cdots, n\}$ for feature nodes, such that the non-zero entries of the feature matrix are concentrated as seen in Figure 2(e).

The mechanism of the matrix reordering procedure is summarized in the following algorithm.

Matrix reordering algorithm
Input: bipartite network $G = (V_T, V_F, E)$ derived from feature matrix $\mathbf{A}$ and hub selection ratio k
Output: permutation arrays $\pi_T : V_T \rightarrow \{1, \cdots, m\}$ and $\pi_F : V_F \rightarrow \{1, \cdots, n\}$

1. select $m_{hub} \leftarrow \lceil k \times |V_T| \rceil$ hubs in V_T and $n_{hub} \leftarrow \lceil k \times |V_F| \rceil$ hubs in V_F respectively
2. place the selected hubs in V_T and V_F to the end of π_T and π_F, respectively; and remove them from G to generate new graph G' which is split into two parts: 1) giant connected component, 2) spokes
3. find the connected components in G' using breadth first search; and place nodes belonging to each non-giant connected component at the beginning of π_T and π_F respectively
4. set $G = (V_T, V_F, E)$ to be the giant connected component (GCC) of G'
5. repeat until the number of nodes in V_T or V_F of the GCC is smaller than current m_{hub} or n_{hub}, respectively

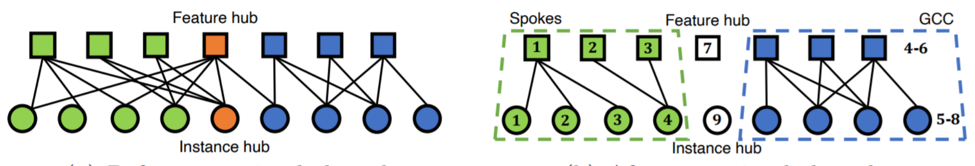
Figure 2 depicts the matrix reordering in the algorithm for the feature matrix of the Amazon dataset. The spy plot of the feature matrix before reordering is in Figure 2(a). Figures 2(b) $\sim$ 2(d) show the intermediate matrices of the reordering process. As iterations proceed, the non-zero entries of the feature matrix are concentrated at the bottom right corner of the feature matrix as shown in Figure 2(e). The final reordered matrix can be divided into four submatrices, where the top and left submatrix contains small rectangular blocks at the diagonal area, and those blocks are formed by the spokes nodes.



(a) The original matrix (b) After thirty iterations (c) After eighty iterations (d) After 115 iterations (e) After the final (119) iteration

Figure 2: The matrix reordering process of FastPI on the feature matrix. (a) depicts the original matrix, (b-d) are the reordered matrix after several iterations in the algorithm, and (e) is the matrix after the final iteration. As shown in (e), the non-zero entries of the feature matrix are concentrated by the matrix reordering such that it is divided into four submatrices where A_{11} is a large and sparse rectangular block diagonal matrix.

Here is an example of how one iteration of the matrix reorientation algorithm works for a small matrix represented as a bipartite network.



(a) Before removing hub nodes (b) After removing hub nodes

Figure 3: A bipartite network after one iteration, where a square indicates a feature node and a circle indicates an instance node. FastPI assigns the highest id to the feature node (id 7) and instance node (id 9), respectively. The nodes in spokes get the lowest ids and the GCC receives the remaining ids. We remove multiple hub nodes in each iteration of the algorithm to sufficiently shatter the graph.

4.3 Incremental SVD Computation of FastPI

The reordered matrix $\mathbf{A}$ is divided into $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$, where $\mathbf{A}_{11} \in R^{m_1 \times n_1}$, $\mathbf{A}_{12} \in R^{m_1 \times n_2}$, $\mathbf{A}_{21} \in R^{m_2 \times n_1}$, and $\mathbf{A}_{22} \in R^{m_2 \times n_2}$. Note that m_1 and n_1 are the number of spoke instance and feature nodes, respectively, and m_2 and n_2 are the number of hub instance and feature nodes, respectively.

First, the computation of the low-rank SVD of $\mathbf{A}_{11}$., which is large and sparse, with small rectangular blocks on the diagonal (Figure 2(e).), is efficiently obtained by computing SVD of each small block in $\mathbf{A}_{11}$. For i-th block $\mathbf{A}_{11}^{(i)} \in R^{m_{1i} \times n_{1i}}$, suppose $\mathbf{U}^{(i)}\mathbf{\Sigma}^{(i)}\mathbf{V}^{(i)\top}$ is the low-rank approximated SVD with the target rank $s_i = \lceil \alpha n_{1i} \rceil$ (let $m_{1i} > n_{1i}$ without loss of generality). Then, the SVD result of $\mathbf{A}_{11}$ is as follows:

$$\mathbf{U}_{m_1 \times s}\mathbf{\Sigma}_{s \times s}\mathbf{V}^\top_{s \times n_1} = bdiag\left(\mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(B)}\right) \times bdiag\left(\mathbf{\Sigma}^{(1)}, \cdots, \mathbf{\Sigma}^{(B)}\right) \times bdiag\left(\mathbf{V}^{(1)\top}, \cdots, \mathbf{V}^{(B)\top}\right) \tag{1}$$

where B is the number of blocks and bdiag $(\cdot)$ is the function returning a rectangular block diagonal matrix with a valid block sequence. $\mathrm{V}^\dagger_{s \times n_1}$ are orthogonal matrices, and $\mathbf{\Sigma}_{s \times s}$ is diagonal, so this is indeed the SVD of the matrix $\mathbf{A}_{11}$.

Next, the SVD result for a vertical concatenation of $\mathbf{A}_{11}$ and $\mathbf{A}_{21}$ is calculated by incremental SVD given the SVD of A $_{11}$ ([4], [5]). The derivation for this incremental computation with the target rank $s = \lceil \alpha n_1 \rceil$ is the following:

$$\begin{bmatrix} \mathbf{A}_{11} \\ \mathbf{A}_{21} \end{bmatrix} \simeq \begin{bmatrix} \mathbf{U}_{m_1 \times s}\mathbf{\Sigma}_{s \times s}\mathbf{V}^\top_{s \times n_1} \\ \mathbf{A}_{21} \end{bmatrix} = \begin{bmatrix} \mathbf{U}_{m_1 \times s} & \mathbf{O}_{m_1 \times m_2} \\ \mathbf{O}_{m_2 \times s} & \mathbf{I}_{m_2 \times m_2} \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_{s \times s}\mathbf{V}^\top_{s \times n_1} \\ \mathbf{A}_{21} \end{bmatrix}$$
$$\simeq \begin{bmatrix} \mathbf{U}_{m_1 \times s}\mathbf{O}_{m_1 \times m_2} \\ \mathbf{O}_{m_2 \times s}\mathbf{I}_{m_2 \times m_2} \end{bmatrix} \underbrace{\tilde{\mathbf{U}}_{(s+m_2) \times s}\tilde{\mathbf{\Sigma}}_{s \times s}\tilde{\mathbf{V}}^\top_{s \times n_1}}_{Low-rank\ approximation\ with\ s} = \mathbf{U}_{m \times s}\mathbf{\Sigma}_{s \times s}\mathbf{V}^\top_{s \times n_1} \tag{2}$$

where $\mathbf{\Sigma}_{s \times s} = \tilde{\mathbf{\Sigma}}_{s \times s}, \mathbf{V}^\top_{s \times n_1} = \tilde{\mathbf{V}}^\top_{s \times n_1}, \mathbf{O}$ is a zero matrix, and $\mathbf{I}$ is an identity matrix. Note that $\mathbf{U}_{m \times s} = \begin{bmatrix} \mathbf{U}_{m_1} \times_s & \mathbf{O}_{m_1} \times m_2 \\ \mathbf{O}_{m_2 \times s} & \mathbf{I}_{m_2 \times m_2} \end{bmatrix} \tilde{\mathbf{U}}_{(s+m_2) \times s}$ is orthogonal since the product of two orthogonal matrices is also orthogonal.

Finally, the incremental update of the SVD result in equation (2) for $\mathbf{T} = [\mathbf{A}_{12}; \mathbf{A}_{22}]$ is performed as follows:

$$\begin{bmatrix} \mathbf{A}_{11}\mathbf{A}_{12} \\ \mathbf{A}_{21}\mathbf{A}_{22} \end{bmatrix} \simeq [\mathbf{U}_{m \times s}\mathbf{\Sigma}_{s \times s}\mathbf{V}^\top_{s \times n_1}\mathbf{T}] = [\mathbf{U}_{m \times s}\mathbf{\Sigma}_{s \times s}\mathbf{T}] \begin{bmatrix} \mathbf{V}^\top_{s \times n_1}\mathbf{O}_{s \times n_2} \\ \mathbf{O}_{n_2 \times n_1}\mathbf{I}_{n_2 \times n_2} \end{bmatrix}$$
$$= \underbrace{\tilde{\mathbf{U}}_{m \times r}\tilde{\mathbf{\Sigma}}_{r \times r}\tilde{\mathbf{V}}^\top_{r \times (s+n_2)}}_{Low-rank\ approximation\ with\ \ r} \begin{bmatrix} \mathbf{V}^\top_{s \times n_1}\mathbf{O}_{s \times n_2} \\ \mathbf{O}_{n_2 \times n_1}\mathbf{I}_{n_2 \times n_2} \end{bmatrix} = \mathbf{U}_{m \times r}\mathbf{\Sigma}_{r \times r}\mathbf{V}^\top_{r \times n}, \tag{3}$$

where $\Sigma_{r \times r} = \tilde{\mathbf{\Sigma}}_{r \times r}, \mathbf{U}_{m \times r} = \tilde{\mathbf{U}}_{m \times r}$, and $\mathbf{V}^\top_{r \times n} = \tilde{\mathbf{V}}^\top_{r \times (s+n_2)} \begin{bmatrix} \mathbf{V}^\top_{s \times n_1}\mathbf{O}_{s \times n_2} \\ \mathbf{O}_{n_2 \times n_1}\mathbf{I}_{n_2 \times n_2} \end{bmatrix}$ are also orthogonal.

4.4 Computational complexity of the algorithm

In the table below, the computational complexity of each step and total complexity is provided.

Line	*Task*	*Computational Complexity*
1	*Reorder* $\mathbf{A}$ *using Algorithm* 1	$O(T(m\log(m)+\lvert\mathbf{A}\rvert))$
2	*Compute SVD of* $\mathbf{A}_{11}$	$O\left(\sum_{i=1}^{B} m_{1i}n_{1i}s_i\right)$
3	*Update the SVD result for* $\mathbf{A}_{21}$	$O\left(m_1r^2+n_1r^2+m_2n_1r\right)$
4	*Update the SVD result for* $\mathbf{T}=[\mathbf{A}_{12};\mathbf{A}_{22}]$	$O\left(n_1r^2+mr^2+mn_2r\right)$
Total	$O\left(mr^2+n_1r^2+mn_2r+m_2n_1r+\left(\sum_{i=1}^{B} m_{1i}n_{1i}s_i\right)+T(m\log(m)+\lvert\mathbf{A}\rvert)\right)$	

The dominant factor is mr^2 in the complexity of the analysis of FastPI. It is faster than the performance of traditional methods of computing pseudoinverse and of Randomized SVD.

5 Our experiments and results

In our experiments we computed the pseudoinverse for different kinds of sparse matrices to assess the influence of different parameters on the performance of FastPI. Solving the multilabel linear regression problem using the resulting pseudoinverse matrices is a particular problem and we didn't handle it in our project only concentrating on the method computational effectiveness and possible weaknesses.

5.1 Parameters vs performance

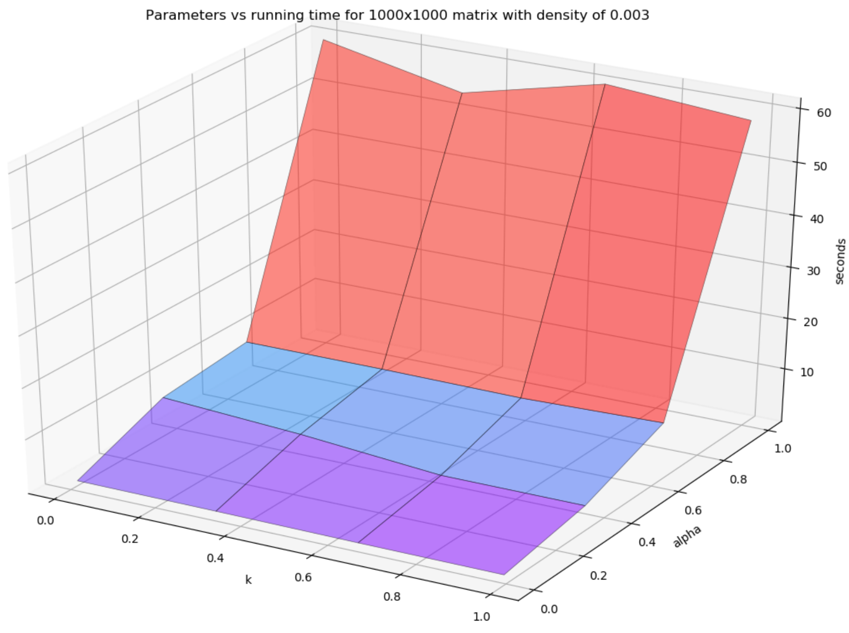


Figure 4:

There are two parameters that can affect the computational performance and precision of Fast Pseudoinverse algorithm. The first one is the target rank ratio α $(0<\alpha\leq 1)$ which is equal to the ratio of target rank and the rank n of the feature matrix $\mathbf{A}$. The second one is the reordering rate k, which is the number of features and samples that are permuted at once during a single iteration of the reordering algorithm, Numerical experiments show that the choice of k has a negligible effect on performance, whereas the choice of α determines the feasibility of the method (Figure 4).

5.2 On density and efficiency of reordering

A slight increase in order of density can render the method ineffective. For density of 0.003 the reordering yields a matrix with a block-diagonal submatrix that covers the majority of its entries (Figure 5). Increasing the density threefold will yield a matrix, for which the block diagonal matrix

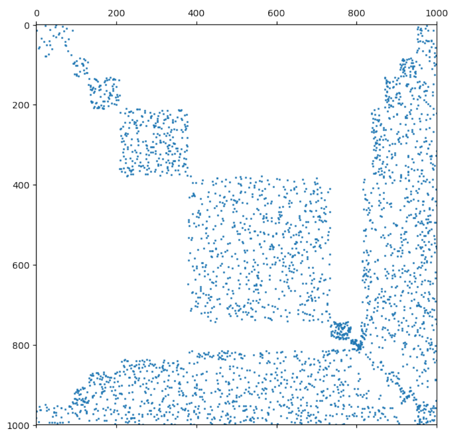


Figure 5: Sparsity pattern for a matrix with a density of 0.003

covers well under a quarter (Figure 6); at this point the proposed method is already failing. because this kind of reordering does not provide any significant boost to performance. If we further increase

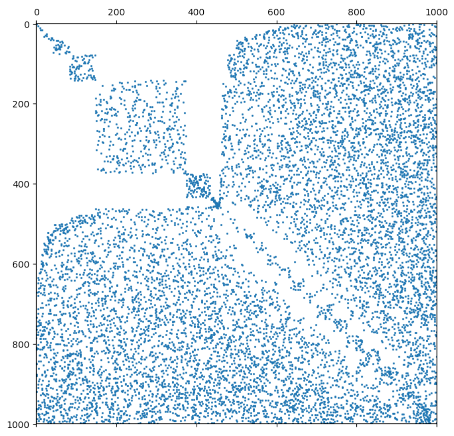


Figure 6: Sparsity pattern for a matrix with a density of 0.01

the density twofold, the proposed method would actually produce worse results than conventional methods, since the reordering appears to have little to no effect, while still taking a toll on total computation time (Figure 7). Still, the matrix on this figure is actually pretty sparse, it only has density of 2%, but the algorithm, nevertheless, cannot handle it.

As a matter of fact the demand for high sparsity gets worse as dimensions grow. The algorithm is only efficient if spokes are severed at more or less constant rates. This way many small diagonal submatrices are produced. So for the algorithm to be efficient each GCC should ideally be close to being a tree. So if a median node has a degree of 10 or more, the algorithm is highly unlikely to produce any spokes even after 50% of iterations have already gone by. With a density of 1% and a dimension of 1000 an average node will have a degree of 10, but if the dimension were to increase tenfold, so would the degree of an average node. Figure 8 depicts the sparsity pattern of a matrix that has the same density as the matrix in figure 7, but with double the dimension.

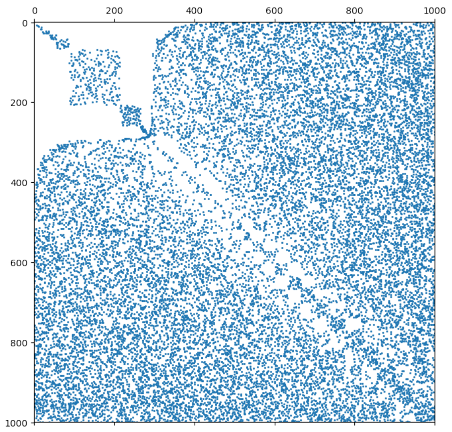


Figure 7: Sparsity pattern for a matrix with a density of 0.02

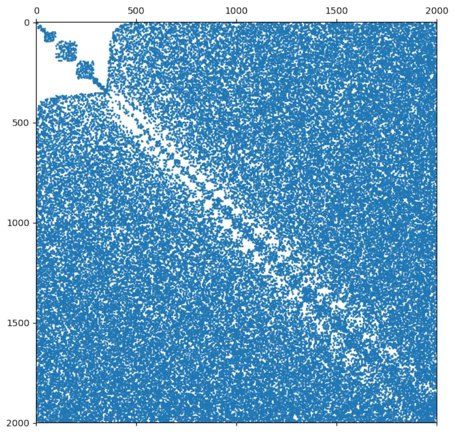


Figure 8: Sparsity pattern for a 2000×2000 matrix with a density of 0.02

5.3 On relative reconstruction error and apparent instability

The relative reconstruction error highly depends on the feature matrix **A**. For different sparse random matrices the relative error fluctuate in the range of four orders of magnitude: $[10^{-12}, 10^{-8}]$. These fluctuations seem to affect the result even more than the choice of parameters.

6 Conclusion

In our project, we implemented the results from the paper proposing how to speed up the approximate pseudoinverse calculation faster than the state-of-the-art low-rank approximation method for computing pseudoinverses with usage of feature matrix reordering.

From the experiments we can conclude that with appropriate choice of parameters there is a significant performance boost. The proposed reordering technique is not a computationally expensive operation as observed from numerical experiments.

The approach yields suboptimal results even on certain fairly sparse matrices.

The FastPI algorithm can potentially have a significantly higher performance than regular approaches, however it implies a very specific domain of application: linear least squares problems. Otherwise the result wold have to be adjusted to negate the effects of prior reordering.

The other significant constraint is matrix density. The greater the matrix dimensions are, the sparser it has to be for the approach to be useful. This constraint is heavy, since even for a 1000×1000 matrix a density of 2% is already completely unacceptable. Also, as our experiments show, the stability issue requires further investigation.

References

[1] Trefethen LN, Bau III D (1997) Numerical linear algebra, vol 50. Siam

[2] Jung J, Sael L (2020) Fast and accurate pseudoinverse with sparse matrix reordering and incremental approach. Machine Learning, 109(12), 2333-2347, https://arxiv.org/pdf/2011.04235.pdf

[3] Lim Y, Kang U, Faloutsos C (2014) Slashburn: Graph compression and mining beyond caveman communities. IEEE Trans Knowl Data Eng 26(12):3077–3089, DOI 10.1109/TKDE.2014.2320716, URL http://doi. ieeecomputerso-ciety.org/10.1109/TKDE.2014.2320716

[4] Brand M (2003) Fast online svd revisions for lightweight recommender systems. In: Proceedings of the 2003 SIAM international conference on data mining, SIAM, pp 37–46

[5] Ross DA, Lim J, Lin RS, Yang MH (2008) Incremental learning for robust visual tracking. International journal of computer vision 77(1-3):125–141