Ukrainian Catholic University

Faculty of Applied Sciences

Data Science Master Programme

Noise reduction techniques based on Singular Value Decomposition

Linear Algebra final project report

Authors:
Nazariy Perepichka
Volodymyr Lut

January 24, 2019

Abstract

Singular Value Decomposition algorithm, undoubtedly, belongs to the Hall of Fame of linear algebra algorithms. There are plenty of materials available on the web about its nature and applications. Despite a massive amount of information, most of the well-written materials and examples concentrate on image compression problem. We decided to dedicate this article to noise reduction techniques based on SVD, because of lack of well-structured materials regarding this application. Our primary goal is to structurize information, that we processed and to help readers make the first steps in this direction. In this paper, we are applying SVD to the audio stream trying to remove noise and echo from the sound.

1 Introduction

The paper is organized as follows. In Section 2 we briefly remind central concepts of linear algebra to understand general context of our work and trying to provide readers with an intuition behind SVD, in Section 3 we describe the structure of GitHub repository which contains implementations of algorithms described in Section 4. In Section 5 we draw conclusions and discuss further possible improvements.

2 Theory behind SVD

Let's start our journey with brief reminding of the core definitions of linear algebra and introducing the SVD method. In case of lack of understanding, feel free to check out refereces[1][2] for a more profound explanation of them.

2.1 Eigenvalues and Eigenvectors

Let A be defined as follows:

$$A \in \mathbb{R}^{n \times n}, n \in \mathbb{N}$$

If there exists any non-zero vector v such that

$$Av = \lambda v$$

we call λ an **eigenvalue** of A and v is called an corresponding **eigenvector** of A.

2.2 Rank

For matrix $A \in \mathbb{R}^{m \times n}, m, n \in \mathbb{N}$ where $rank(A)$ is equal to number of linearly independent columns $rank(A)$ is the dimension of the vector space generated by its columns.

2.3 Orthogonal matrix

Matrix $A \in \mathbb{R}^{n \times n}, n \in \mathbb{N}$ is an orthogonal matrix if its columns and rows are orthogonal unit vectors (also called orthonormal vectors). Orthogonal matrices have following properties:

$$A^T A = AA^T = I,$$

$$A^{-1} = A^T$$

2.4 Singular Value Decomposition

Theorem. $\forall A \in \mathbb{C}^{m\times n}, m, n \in \mathbb{N}$ exist factorization called a 'singular value decomposition', which looks as following:

$$A = U\Sigma V^T$$

Where:

- U is a matrix of size $m \times m$ of **left singular vectors**
- Σ is diagonal matrix of size $m \times n$ with non-negative real numbers on the diagonal, so called **singular values**(square root of eigenvalues of $A^T A$ matrix)
- V is a matrix of size $n \times n$ of **right singular vectors**

This can be rewritten in following form:

$$A = \sum_i \sigma_i u_i \cdot v_i^T$$

Where U, Σ are unique, U and V are orthogonal.
Theorem. Reconstruction A_k of matrix A, given as

$$A_k = \sum_{i=1}^{k} \sigma_i u_i \cdot v_i^T$$

is the best approximation of A of rank k.
All matrices have an SVD, which makes it more stable than other methods, such as the eigendecomposition. As such, it is often used in a wide array of applications including compressing, denoising, and data reduction.

2.5 Geometrical interpretation

As shown on Figure 1: geometrical interpretation of the SVD is following:
Any $m \times n$ matrix A maps the 2-norm unit sphere in $\mathbb{R}^n$ to an ellipsoid in $\mathbb{R}^r(r \leq min(m, n))$.

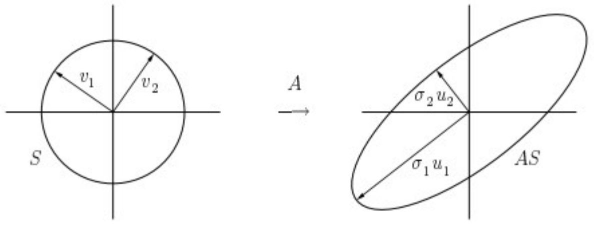

Figure 1: Geometrical interpretation of SVD

2.6 Intuition

You probably know that feeling when you can spoil a dish with an inappropriate ingredient or with an unexpected amount of needed ingredient? There is some nature behind dishes, and it's common sense that they rely deeply on their ingredients. Adding too many mushrooms and just a bit of beet would make borsch more like mushroom soup. On the other hand, adding fennel or sour cream are not so crucial for borsch to be borsch.[3]
In the context of flavor, SVD allows us to decompose matrix of taste in vectors corresponding to each ingredient unique flavor and magnitude of its impact.
Later with SVD, we can reconstruct the most borsch-like approximation with a limited amount (k) of ingredients to use.

3 Structure of GitHub repository

All the supportive code can be found in our GitHub repository [4]. The repository consists of three folders, .gitignore file, and Jupyter notebook. Below we provide the description for folders:

- `data` - contains .wav files used for noise filtering and results of different approaches for filtering
- `utils` - contains three scripts:
  1. `filter_util.py` - contains implementation of windowed algorithm
  2. `svd_util.py` - contains implementation of reduced svd and function for restoring matrices with given U, E and V matrices
  3. `wav_util.py` - functionality for manipulation with .wav files

In `noise_reduction_demonstration.ipynb` notebook you can find implementation of all the steps discussed in "Noise filtering" section.

4 Noise filtering

Let's assume that u is a bidimensional clean original data. u_0 is a version of u, which we receive, the one polluted with the noise n.

$$u_0 = u + n$$

The problem, which we want to solve is to perform manipulations on given u_0 to obtain u. Let's assume, that our input u is represented in matrix form. Let's use SVD on this matrix.

$$A = U\Sigma V^T$$

Let's add additional constraint to diagonal matrix Σ. The following condition should be true - $\Sigma_{11} \geq \Sigma_{22} \geq ... \geq \Sigma_{nn}$ (it's a very important thing about Σ; it's diagonal elements should be decreasing).
Denoising techniques via SVD is based on following assumption: Σ matrix can be partioned into two blocks Σ_u and Σ_n by some threshold index τ.

$$\Sigma = \begin{bmatrix} \Sigma_u & 0 \\ 0 & \Sigma_n \end{bmatrix}$$

Therefore, by performing the reconstruction of the original matrix using τ singular values and corresponding vectors we should receive a denoised signal.
We will call this approach to noise filtering a **noise suppression via compression**.

4.1 Naive algorithm

Our baseline implementation consisted of 4 steps.

- Preprocess raw .wav files into frame matrices
- Decompose matrices using self-implemented reduced and full SVD algorithms
- Reconstruct frame matrices using some percentage p of top singular values and corresponding vectors
- Write reconstructed structures to new .wav files

All of the above steps can be viewed in our GitHub repository[4]. From noise filtering point of view, results of this naive approach were poor. Based on choosing of p hyperparameter, we received some noise filtering effect, but it was far from adequate results.
Still, if we look at this approach from data compressing point of view, the results were quite fascinating. Using an only quarter of all elements, the difference between restored and original file were barely distinguishable for the human ear.

4.2 "Window" process

Our algorithm for more sophisticated noise filtering was highly inspired by a paper about denoising images [5]. "Window" process, which we perform on matrix A is intuitively similar to Convolution layers in Deep Neural Networks.
Let A be the frame matrix, defined as follows:

$$A \in \mathbb{R}^{m \times n}, m, n \in \mathbb{N}$$

We try to receive $\hat{A}$ frame matrix which is a denoised version of A

$$\hat{A} \in \mathbb{R}^{m \times n}, m, n \in \mathbb{N}$$

We choose values p and l:

$$p, l \in \mathbb{N}, p < \frac{m}{2}, l < \frac{n}{2}$$

Additionaly, we choose value stride $s \in \mathbb{N}$.
Starting from left edge of matrix A we select subsection of A of size $p \times l$ and perform SVD on this part. We add k approximation of original subsection to corresponding area of predicted output $\hat{A}$, and move the start of the next subsection on s columns and repeating this step, while we don't reach the right end of matrix $A(r * s + p < m$, where r - number of iterations). We follow the same algorithm to go down the rows of A. In the end we average values of $\hat{A}$.

4.3 Windowed algorithm

Our algorithm can be described in the following steps:

- Preprocess raw .wav files into frame matrices
- Retrieve $\hat{A}$ by running ”window” process
- Write $\hat{A}$ to new .wav files

4.4 Results analysis

Due to computational complexity, objective measurements will be provided in our GitHub repository.
From a subjective point of view, the windowed algorithm performs a better job of denoising signal, but the process of choosing hyperparameters is highly unobvious and outrageously mechanical.

5 Comparison with other algorithms

Nowadays, state-of-the-art solution for the noise reduction on signal is Deep Recurrent Neural Networks. They provide an output of the excellent quality and shaping a new area in business already[6]. The main idea is to combine classic signal processing with deep learning to create a real-time noise suppression algorithm that's small and fast.
Despite much better performance, it is hard to grasp an intuitive understanding of how those algorithms work. Computational complexity for training such a system is very high, therefore, training such Nets requires powerful and expensive hardware.
Speaking about more generic solutions, partial differential equations are used for both image and sound noise reduction. Fourth-order differential equations are derived as the process that seeks to minimize a function proportional to the absolute value of Laplacian of the intensity function. With time they are converging to the output of good quality. Example of the combination of this approach and SVD denoising can be found in [7].
It is also worth saying that there are pure statistical methods of noise suppression. Speaking about Bayesian methods, it is effective to treat sound's data as a Bayesian prior and the auto-normal density as a likelihood function, with the resulting posterior distribution offering a mean or mode as a denoised sound.
SVD-based noise suppression has a good tradeoff when we speak about complexity and the quality of output. It is easy to understand and to implement, though, unfortunately, have no implementation in business since small and elegant deep learning models already exist, providing better output.

6 Conclusion

As we see from the previous section, SVD-based algorithms are not providing the best result in terms of noise reduction problem. However, it is a good example for understanding of how SVD works. We hope, that implemented python code and this paper would help readers with understanding the practical method implementations overall.
Noise reduction is a common task and algorithms for solving it are mostly developed in a supervised environment. However, algorithms which we implemented allow performing

a computationally cheap approximation of denoised signal in terms of unsupervised environment.
Overall SVD is a brilliant and easy-to-use tool to understand the matrix nature and can be used in various problems.
Along the solving noise reduction problem, we one more time reassured ourselves in power of SVD for data compression problem.

References

[1] David C, Lay, Stephen R. Lay, Judi J. MacDonald. *Linear Algebra and Its Applications.* Pearson Education Limited, 5th edition, 2016.

[2] Stephen Boyd, Lieven Vandenberghe. *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares.* Cambridge University Press, 2018

[3] Interview with former Ukrainian president: Borsch Philosophy
`https://youtu.be/KLxHLeVIB7c?t=700`

[4] Supportive code
`https://github.com/baaraban/noise_reduction`

[5] Tsegaselassie Workalemahu. *Singular Value Decomposition in Image Noise Filtering and Reconstruction.* Thesis, Georgia State University, 2008.
`https://scholarworks.gsu.edu/math_theses/52`

[6] Business implementation of noise filtering with Deep Reccurent Neural Networks
`https://2hz.ai/samples/index.html`

[7] George Baravdish, Gianpaolo Evangelista, Olof Svensson. *PDE-SVD BASED AUDIO DENOISING.*
`http://liu.diva-portal.org/smash/get/diva2:535821/FULLTEXT01.pdf`