Unveiling the limitations of SVD: An Amortized Analysis

Aksh Garg
Computer Science
Stanford University
akshgarg@stanford.edu

Abstract

With ever-growing datasets and image resolutions, dimensionality reduction has grown increasingly popular in recent years - and for good cause. Efficient dimensionality reduction methods allow us to store more data, avoid overfitting (which can easily occur when dealing with millions of features as common in images), and train and run machine learning models in lower time and latency. Singular Value Decomposition, and specifically its application in low-rank approximations, is a common technique for reducing the dimensionality of images - and a go-to for many practitioners due to its simplicity, ease of implementation, and clear mathematical explainability. However, SVD is limited in (1) its ability to only find linear transformations of the input dataset and (2) its limited dimension reduction capabilities. This project exposes this second weakness of SVD. Specifically, using a very-basic dataset of hand images, it unveils that even on simple tasks, SVD fails to offer dimension reductions that are at par with more sophisticated non-linear machine learning methods like autoencoders. I show that rank-reduction methods would require 7x as much space per-image as one using an, auto-encoder. Next, I examined some of the additional overheads with using an autoencoder and identify use-cases where SVD for reduction would be preferred.

1. Introduction

Researchers estimate that we generated 94 zettabytes (1 billion terabytes) of data in 2022, an astronomical number by any metric. [1]. We are living in an age of information overload, and the exponential growth in data is unlikely to stop soon. In many ways this is good. Larger datasets allowed for the creation of powerful generative models like ChatGPT - and unlocking the true power of AI hinges upon maximizing the information sources available to us.

That said, however, this voluminous growth in data also poses unique challenges. For one, high-dimension data can be hard to visualize and understand. As the number of dimensions of any input feature increases, plotting it and understanding it's underlying patterns becomes equally harder. For another, as we work with higher-dimension datasets, the machine learning models we train must also become more complex to accurately capture patterns within it. This often leads to overfitting, where the model captures minute trends in the training dataset but is not able to generalize well to unseen inputs. Finally, training these larger models requires more computational resources that might not always be available. Even in cases, where these resources are available, high dimensions inevitably mean larger training times and higher latency in predictions, both of which are factors practitioners would like to minimize.

If the face of these obstacles, dimensionality reduction has emerged as a promising salve. As a high-level overview, dimensionality reduction algorithms work by projecting a higher feature representation of any input into a lower-dimensional one, with the overall goal of preserving as much information as possible. Assuming no information loss, this has significant advantages. First, by mapping features into a lower-dimension space, they make it easier to plot, analyze, and understand. Secondly, reduced dimension sizes for inputs mean that we can train smaller models for capturing trends within our input data sources. This reduces the risks of overfitting, which is equivalent to boosting generalizability, one of the greatest challenges with modern-day machine learning. Finally, reduced dimensionality means faster compute times, which as discussed above can be important.

Having established the importance for dimensionality reduction, let's now consider two commonly used methods for dimensionality reduction - singular value decomposition and autoencoders.

2. Background

2.1. Singular Value Decomposition

Singular value decomposition (SVD) is a matrix factorization technique that decomposes a matrix into three matrices: U, a left singular matrix containing the left singular eigenvectors of A; Σ, a diagonal matrix containing the "singular values" of A; and V, a right singular matrix containing the right singular eigenvectors of A. The decomposition is given by the following equation:

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

and can be incredibly useful for dimensionality reduction. In particular, if we rewrite A as

$$A = U\Sigma V^T = \sum_{i=1}^{n} \sigma_i u_i v_i^T$$

then we notice that we can re-express A as a sum of rank 1 matrices formed by the product of the left and right singular eigenvectors, $u_i v_i^T$. Eckhart and Young built upon this property of SVD's and found that the best rank-k approximation of a matrix A with respect to the Frobenius norm, B, is given by using a smaller version of the SVD, where the last $r - k$ entries are zeroed out [2]. Thus,

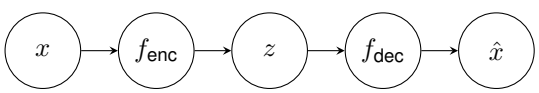
$$B = \sum_{i=1}^{k} \sigma_i u_i v_i^T.$$

In the context of image, dimensionality reduction, this means that we can store simply the first k left and right singular eigenvectors along with the singular value and use that for approximating our image.

Let's briefly examine the reductions we get. When using a rank-k approximation on a matrix A with dimensions $m \times n$, our input image has mn pixels. Now, when using a $k - rank$ approximation, we have to store k u_i vectors, k v_i vectors, and k singular values, which ends up amounting to $km + kn + k$ values to store. Thus, overall, the reduced images are $\frac{km+kn+k}{mn}$ of the original size.

2.2. Autoencoders

Autoencoders are a type of neural network that are commonly used for dimensionality reduction [3]. It has two parts - an encoder and a decoder. The encoder takes an input image and maps into a lower-dimensional representation. The decoder doesn the opposite. It takes a vector in this lower-dimensional space and attempts to convert it back to the original space. The diagram below illustrates the nature of an autoencoder,



Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 512)	401920
leaky_re_lu (LeakyReLU)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131328
leaky_re_lu_1 (LeakyReLU)	(None, 256)	0
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32896
leaky_re_lu_2 (LeakyReLU)	(None, 128)	0
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 64)	8256
leaky_re_lu_3 (LeakyReLU)	(None, 64)	0
dropout_3 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 32)	2080
leaky_re_lu_4 (LeakyReLU)	(None, 32)	0

Total params: 576,480
Trainable params: 576,480
Non-trainable params: 0

Figure 1. Encoder Model

We train the autoencoder to minimize the error between x and $\hat{x}$. For my formulation, I try to minimize the mean squared error between the input and predicted pixels, which is similar to measuring the frobenius distance between images and is, therefore, similar, in effect to what we were doing with low-rank approximations in SVD.

My encoder is composed of five dense layers, which encode my input image into a latent vector of size LATENT SIZE = 32, as illustrated in figure 1.

Similarly, the decoder is composed of a series of dense layers that convert the latent vector back into a matrix of the shape of the image, as illustrated in figure 2.

3. Dataset and Training

I tested both approaches on the MNIST handwritten digits dataset [4]. The dataset, which consists of images of handwritten digits ranging from 0-9 is considered to be one of the simplest and most classic image recognition problems.

To reduce images for SVD, I simply find the decomposition for every single matrix, single out the trailing singular values, and generate a low-rank approximation. To reduce images via the autoencoder, I first train an autoencoder to reduce and reconstruct the input images and then gauge its performance on unseen test images. I train the autoencoder using the adam-optimizer, which builds in gradient momentum from previous iterations and normalizes gradients in each direction by the absolute value of past gradients, for a total of 15 epochs. Furthermore, I keep track of the validation loss while training

Layer (type)	Output Shape	Param #
dense_5 (Dense)	(None, 64)	2112
leaky_re_lu_5 (LeakyReLU)	(None, 64)	0
dropout_4 (Dropout)	(None, 64)	0
dense_6 (Dense)	(None, 128)	8320
leaky_re_lu_6 (LeakyReLU)	(None, 128)	0
dropout_5 (Dropout)	(None, 128)	0
dense_7 (Dense)	(None, 256)	33024
leaky_re_lu_7 (LeakyReLU)	(None, 256)	0
dropout_6 (Dropout)	(None, 256)	0
dense_8 (Dense)	(None, 512)	131584
leaky_re_lu_8 (LeakyReLU)	(None, 512)	0
dropout_7 (Dropout)	(None, 512)	0
dense_9 (Dense)	(None, 784)	402192
activation (Activation)	(None, 784)	0
reshape (Reshape)	(None, 28, 28)	0

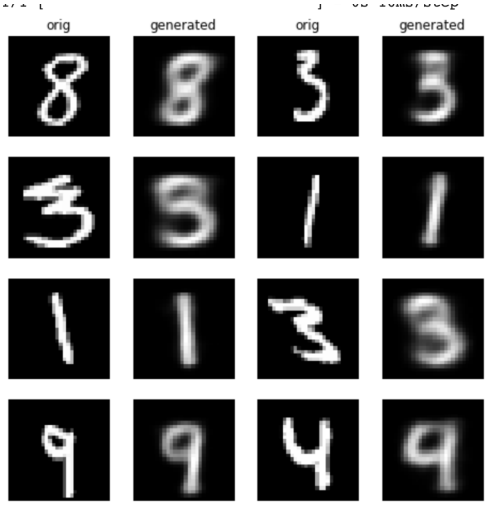
Total params: 577,232
Trainable params: 577,232
Non-trainable params: 0

Figure 2. Decoder Model

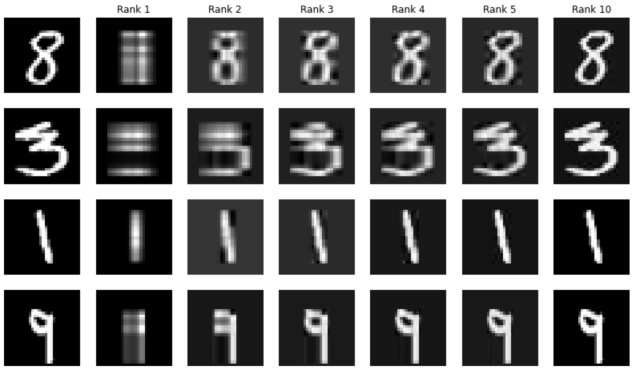
4. Results

Here, I present the downsampled images from SVD as well as the reconstructed images as found via the autoencoder.

Looking first at the autoencoder, we find that with a latent space of size 32, the model is able to capture the overall pattern of the images pretty well. This is noticeable in figure 3, where our reconstructed images, match the original images pretty well. This is indeed very impressive, when we consider that our autoencoder only had a 32-dimensional feature vector, and thus is reconstructing a lot in the final image from just that.



Next, if we look at the images generated from the low-rank approximations, we note something interesting. Specifically, the rank-1 approximation, which contains (28 + 28 + 1 = 57) entries doesn't resemble the input image at all. In fact, it takes a rank-4 approximation to start matching the results offered through the auto-encoder, which already is a size of 228 (28*4 + 28*4 + 4). This is 7.125 times bigger than the representation obtained via the auto-encoder. Therefore, even from this very simple case, we start to observe that the performance abilities of SVD's can be outperformed by a very shallow autoencoder.



5. Discussion

5.1. Autoencoder vs SVD

We noticed that the autoencoder was able to capture just as much information as a rank-4 approximation of the input image, but using roughly 7 times less parameters. Astute critics might comment, however, on how the autoencoder model itself is bigger and storing the model itself has an additional burden. That reader would, in fact, be correct. The autoencoder model does in fact have a considerable size. In fact, one can see that it has upwards of 500,000 parameters. And this is where the amortized aspect of this paper comes in. When selecting between an autoencoder and SVD approach for dimensionality reduction, users have to closely examine their use case.

Specifically, with SVD, we have to compute a decomposition for every single image. As a result, the reduced image size scales with the number of images. On the other hand, the model storage cost of an auto-encoding model is a one-time cost. Therefore, in the case of large numbers of images, let's say even a moderate amount of 10,000 (which is far smaller than how many images most modern datasets have), the net cost per image comes down to approximately 50 features, making it still more memory efficient than an SVD approach. With larger numbers of images in the dataset, this cost would approach 0 features, making autoencoders a suitable encoding choice.

On the other hand, when we are working with a sparse dataset, SVD might be a much more favorable approach. This is because (1) the additional overhead of storing an autoencoder would likely not be worth it and (2) we might not have enough images to train a good enough autoencoder in the first place. (2) is an important point to consider. Specifically, SVD is nearly always guaranteed to work, albeit requiring a higher rank approximation for cases where the eigenvalues are more similarly distributed. On the other hand, the performance of the autoencoder is highly contingent on the autoencoding model trained - and autoencoders and GANs are typically considered to be some of the hardest machine learning models to train, often requiring extensive hyperparamter turning.

5.2. Limitations

This paper served as a proof of concept for a case where SVD's are less space efficient than more sophisticated dimensionality reduction machine learning algorithms. However, due to its limited scope, it doesn't fully rule out the possibility that there might be datasets, where autoencoders don't perform quite as well. Doing so would require a systematic analysis of the "different types" of datasets that exist, which was hard to quantify during the timespan of this project and accurately define generally.

6. Conclusion

This paper serves as a case-study, where using an autoencoder offers a much better per-image dimensionality reduction capacity than rank-approximating SVD methods. We showed that an autoencoder was able to reduce the number of features in an input image 7.125 times more effectively than a rank-reduction method. We chose not to delve deeply into the ability of autoencoders to model non-linear transformations, which SVD can't do, as that has been studied extensively within the literature. However, even within a more linear transformation case where we are working with a basic dataset like MNIST, SVD was outcompeted. We recommend repeating this analysis for more types of datasets, where "type" may be approached in the context of field like medicine, autonomous vechicles, etc. as well as the size of the input images.

References

References

[1] Andre, L. (2022, November 4). 53 important statistics about how much data is created every day. Financesonline.com. Retrieved December 14, 2022, from https://financesonline.com/how-much-data-is-created-every-day/ 1

[2] Strang, G. (2019, July 18). 7. Eckart-Young: The closest rank K matrix to a. YouTube. Retrieved December 14, 2022, from https://www.youtube.com/watch?v=Y4f7K9XF04k 2

[3] Bank, D., Koenigstein, N., amp; Giryes, R. (2021, April 3). Autoencoders. arXiv.org. Retrieved December 14, 2022, from https://arxiv.org/abs/2003.05991 2

[4] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142. 2