Autoencoder and Principal Component Analysis

Max Levermann
University of Würzburg
Institute for Mathematics
Emil-Fischer-Straße 30, 97074 Würzburg
E-mail: max.levermann@uni-wuerzburg.de

02.10.2022

Abstract

Autoencoders are a widely known tool for data compression or dimensionality reduction using neural networks. The network architecture forces the data to be compressed while being able to recover major parts of the original data. In this paper it is shown how linear autoencoders can naturally be linked to principal component analysis, a matrix factorization approach for dimensionality reduction. A comparison is made between both approaches regarding the reconstruction performance as well as some embedding properties within the hidden layer.

Keywords: autoencoders, principal component analysis, unsupervised learning, compression, embedding

1 Introduction

In most fields of machine learning one is often confronted with large amounts of data. Being able to reduce the dimension of the data is, therefore, something that is always desirable, even more so if it contains sparse or redundant information. The challenge in doing so, is to keep the maximum amount of information. Principal component analysis (PCA) is a commonly used method that achieves this by computing the directions with the highest variance in the data. Another more versatile yet simple approach are autoencoders (AE). They are a specific type of a neural network, that tries to learn a coded version within the hidden layers of the network. This is done in an unsupervised fashion by setting the target values to the input values. The network architecture can be split into encoder and decoder part. The encoder is used to compress the input by feeding the data through layers which sequentially become smaller until a desired compression is reached. The decoder is normally a symmetric counterpart of the encoder, that passes the compressed data through the mirrored layers with the goal of reconstructing the input. The idea of using a network that can learn a meaningful representation by setting the target values to the input was first introduced in 1986 by Hinton, Rumelhart and Williams [1]. Ever since, autoencoders have been a constant subject of research, and their broad spectrum of use makes them one of the most essential frameworks in machine learning.

Autoencoders and PCA are fundamentally connected, and the main motivation behind this work is to derive a better theoretical understanding of this connection. We begin in Section 2 by giving a detailed derivation of PCA, allowing us to establish and fully understand the relationship to autoencoders. Section 3 is devoted to a comparison between the two approaches. We consider the reconstruction performance as well as the ability to find embeddings in a lower dimensional space. Lastly we draw a conclusion of the results in Section 4.

2 Principal component analysis

Principal component analysis finds a lower dimensional representation of a given set of data by using a factorization of the data matrix. We will consider $X \in \mathbb{R}^{d \times N}$ as our data, where N gives the number of samples and d the number of features each sample consists of. The goal is then to reduce the number of features per sample to $k < d$, but to keep the maximum information. There are two different perspectives to take when formulating the objective of PCA. For the first and more intuitive one we consider a orthogonal projection (orthonormal rows) $U^T \in \mathbb{R}^{k \times d}$ onto a k dimensional subspace and then minimize the reconstruction loss:

$$\min_U \sum_{i=1}^{N} \|X_i - UU^T X_i\|^2 = \min_U \|X - UU^T X\|_F^2 \tag{1}$$

The second perspective formulates the objective as a maximization of the *Variance of the projected data*. To make sence of the variance we need to have some kind of random variable. For our data X we can interpret each row, which corresponds to one specific feature, as a one dimensional random variable with N realizations. The same works for the projected data $U^T X$, where the j'th row can be written as $u_j^T X$ with u_j^T being the j'th row of U^T. Note that for $u_j = e_j$ we simply obtain the j'th row of X which is precisely the random variable corresponding to the j'th feature. The maximization is now formulated iteratively row-wise over U^T. For the first row there's only the unit length constrained and no orthogonality constrained:

$$\max_u Var(u^T X) \qquad s.t.\ \ u^T u = 1 \tag{2}$$

Using the definition of the variance we calculate:

$$\begin{aligned}
Var(u^TX) &= \mathbb{E}[(u^TX - \mathbb{E}(u^TX))^2] = \frac{1}{N}\sum_{i=1}^{N}(u^TX_i - \frac{1}{N}\sum_{j=1}^{N}u^TX_j)^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}(u^TX_i - u^T\bar{X}) = u^T\frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})(X_i - \bar{X})^Tu \\
&= u^TC_xu
\end{aligned}$$

Where $\bar{X}$ is the center of the data and C_x is the covariance matrix of X. The optimization problem now becomes

$$\max_u u^TC_xu \qquad s.t.\ u^Tu = 1 \tag{3}$$

with Lagrangian $L = -u^TC_xu + \lambda(u^Tu - 1)$. The first order necessary condition yields the eigenvalue problem $C_xu = \lambda u$. Hence, all candidates are eigenvectors of the covariance matrix, and the maximum is obtained for the one with the largest eigenvalue. The eigenvector for the second largest eigenvalue naturallay satisfies the orthogonality constrained w.r.t. the first eigenvector, and is, therefore, the solution for the second iteration. Continuing like this, we can construct an orthogonal projection onto a k dimensional space that maximizes the variance of the projected data by taking the first k eigenvectors of C_X and defining a matrix consisting of those eigenvectors as rows.
To see how minimizing the reconstruction error is equivalent to maximizing the variance we consider the centered data and calculate:

$$\begin{aligned}
&\min_U \sum_{i=1}^{N} \|(X_i - \bar{X}) - UU^T(X_i - \bar{X})\|^2 \\
&= \min_U \sum_{i=1}^{N}((X_i - \bar{X})^T(X_i - \bar{X}) - \sum_{j=1}^{k} u_j^T(X_i - \bar{X})(X_i - \bar{X})^Tu_j \\
&= \min_U \sum_{i=1}^{N}((X_i - \bar{X})^T(X_i - \bar{X}) - \sum_{j=1}^{k} u_j^TC_Xu_j \\
&= \sum_{i=1}^{N}((X_i - \bar{X})^T(X_i - \bar{X}) \min_U - \sum_{j=1}^{k} u_j^TC_Xu_j \qquad \Leftrightarrow \qquad \max_U \sum_{j=1}^{k} u_j^TC_Xu_j
\end{aligned}$$

The centering is needed, because otherwise, we would need to consider affine projections. The variance, on the other hand, is already independent of the center of the data (the mean is considered in the formula).

2.1 Obtaining principal components via singular value decomposition

The covariance matrix for centered data is given by $C_X = \frac{1}{N}XX^T$. So far we've seen that we can obtain the first k principle components by using the eigendecomposition of the covariance matrix $C_X = \frac{1}{N}UDU^T$ and computing $U^T|_k X \in \mathbb{R}^{k\times N}$ where $U^T|_k$ is truncated after k rows. The eigenvalues of XX^T are the squared singular values of X. The SVD of X is given by $X = U\Sigma V^T$ with the same matrix U as above. We calculate:

$$\begin{aligned} C_X &= \frac{1}{N}XX^T = \frac{1}{N}U\Sigma V^T V\Sigma^T U^T = U\frac{\Sigma^2}{N}U^T \\ \iff C_X u_i &= \frac{\Sigma_i^2}{N}u_i \qquad\qquad \forall i = 1,\ldots,d \end{aligned}$$

Therefore, the left singular vectors of our data matrix X are also eigenvectors of the covariance matrix with eigenvectors $\lambda_i = \frac{\Sigma_i^2}{N}$. The principle components can now be calculated as $U^TX = U^TU\Sigma V^T = \Sigma V^T$. This is often the prefered approach for calculating the principle components since it doesn't require to calculate the covariance matrix first. Even though we can save on this one matrix multiplication, computing the eigenvectors can still be inefficient for large datasets.

3 Autoencoders

To study similarities between PCA and autoencoders we will mainly look at linear autoencoders with identity activation and with only a single hidden layer. The loss function will be the mean squared error which we can think of as a squared and averaged Frobenius norm. Since they are equivalent as norms it won't be nececessary to precisely distinguish between them.

3.1 Linear autoencoders

If we consider a linear autoencoder without bias we can express the learning objective as follows

$$\min_{W_1,W_2} \|X - W_2W_1X\|^2. \tag{4}$$

We call $W_1 \in \mathbb{R}^{k\times d}$ and $W_2 \in \mathbb{R}^{d\times k}$ the encoding and decoding matrices, respectively, where k is the dimension of the embedding space. When computing the gradient w.r.t. W_1 or W_2 we get that W_1 is the pseudoinverse of W_2 (or vice versa). The minimization can now be written as

$$\min_{W_2} \|X - W_2W_2^+X\|^2 \tag{5}$$

which is very similar to the reconstruction loss that PCA is trying to minimize. The only difference is that here we don't have orthonormality constraines on the rows of W_2^+, so that the solutions won't nececessarily be the same. It has however been shown [2] that W_2 is a solution of (5) if and only if its column space is the same as the space spanned by the left singular vectors of X or equivalently by the eigenvectors of C_X. There have also been suggestions on how to recover the precise eigenvectors of C_X from the autoencoder weights [3]. Due to the orthogonality constraints, PCA produces features that are completely uncorrelated, which is not the case for autoencoders. However, PCA is restricted to learn linear features, whereas autoencoders with nonlinear activations aren't.

3.2 Reconstruction

To compare the reconstruction abilities of PCA and our simple autoencoder, we used the MNIST dataset which consists of 28 x 28 gray scale images of handwritten digits. For our analysis we only used the first 1.000 images. For the lower dimensional representation space we chose a dimension of 32. Figure 1 shows the first 100 principal components and the magnitudes of their eigenvalues in percent. The first 32 principle components sum up to about 76 percent. We can think of these 76 percent as the amount of information of the original data that is stored in the first 32 principle components. When reconstructing the compressed data with those 32 principal components we get a mean squared error of 0,016 (Figure 2).

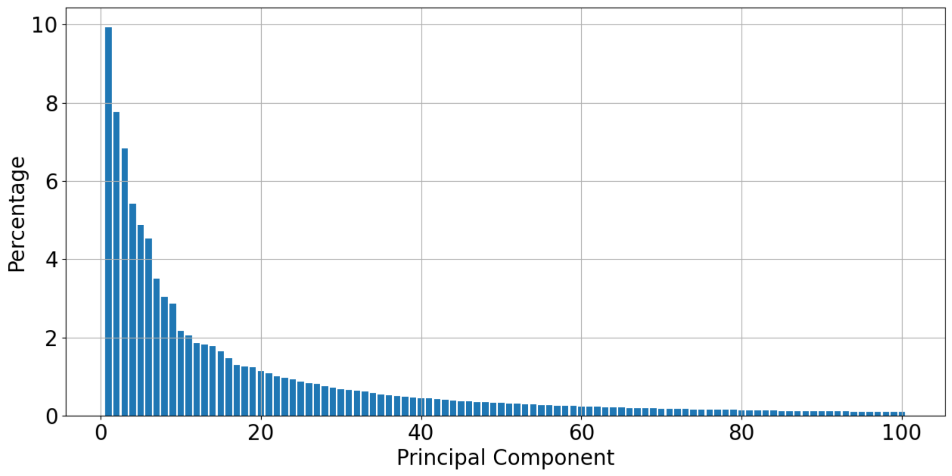

Figure 1: Scree plot showing first 100 principal components of the first 1000 MNIST images and their eigenvalue magnitude in percent

We trained two different autoencoders both with one hidden layer of size 32. The first one is completely linear and therefore tries to minimize the loss (4) introduced above. The other uses the sigmoid function as an activation on the output layer yielding the minimization objective

$$\min_{W_1,W_2} \|X - \sigma\left(W_2 W_1 X\right)\|^2. \tag{6}$$

The error backpropagation is done without using mini batches. In Figure 2 we can see that the linear autoencoder takes about 600 iterations to reach the same loss as PCA. The non linear autoencoder learns faster and reaches a loss of $0,013$ after the same number of epochs while achieving the same loss as PCA after about 280 epochs.

Figure 3 shows the reconstructed images compared to the orginal input. The reconstructions of PCA and the linear autoencoder are pretty similar, which is kind of what we expected with the theoretical background we established before. We can also see, that the non linear autoencoder not only achieves a lower reconstruction loss but also recovers images that look more similar to the original data.

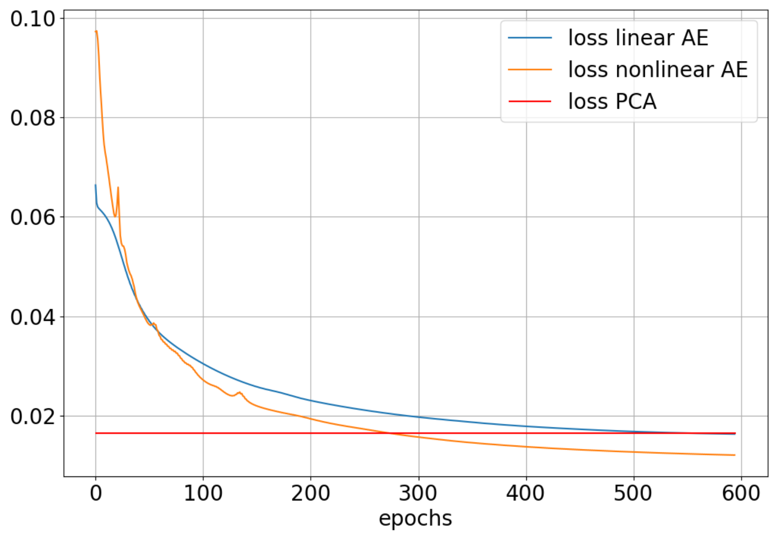

Figure 2: Loss of a linear and nonlinear autoencoder over 600 epochs compared to the loss of PCA

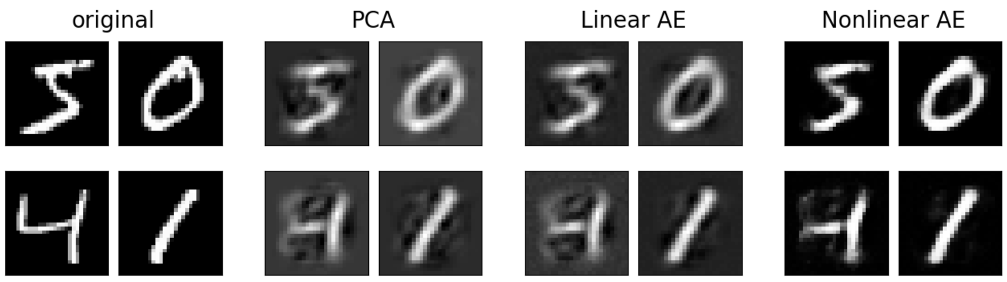

Figure 3: Reconstruction of a sample of MNIST images

3.3 Embeddings from hidden layer

In some areas of machine learning, such as natural language processing or network analysis where graph structured data is studied, it is really crucial to somehow embed the data into a real vector space to be able to calculate distances and errors between data points. To get these embeddings often an autoencoder neural network is used. For the MNIST data set we already have data in the form of real vectors that can be fed into a machine learning framework but it's, nevertheless, interesting to take a look at the embeddings. To be able to see and plot the embeddings we must find a 2 or 3 dimensional representation. In Figure 4 we can see the embedding that a linear autoencoder with 2 hidden neurons learns, compared to the first two principal components. Figure 5 shows the reconstruction that can be obtained using only the information of those two dimensions. The scree plot (Figure 1) tells us that the first two principal components correspond to 17 percent of the total information.

The digit 1 is represented by the orange dots and we can see that both embeddings find a pretty good cluster for this number. The 0 can also be distinguished quite well from the other numbers in both cases. PCA, in contrast to the autoencoder, kind of clusters the 7 and 9 together, but the rest of the digits cannot be told apart. This behaviour is also reflected in Figure 5 where the 0 and 1 are the only digits that can sort of be reconstructed, as well as the 7 and 9 from PCA.

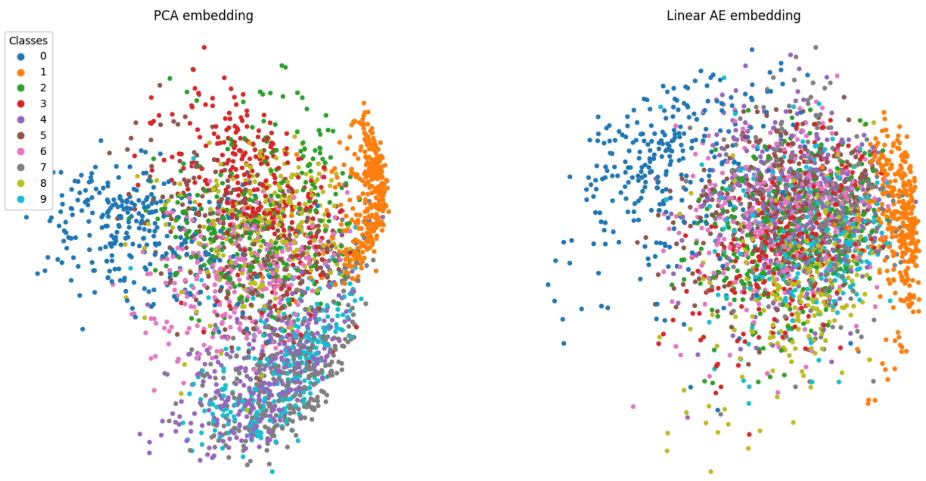

Figure 4: Embedding in a plane

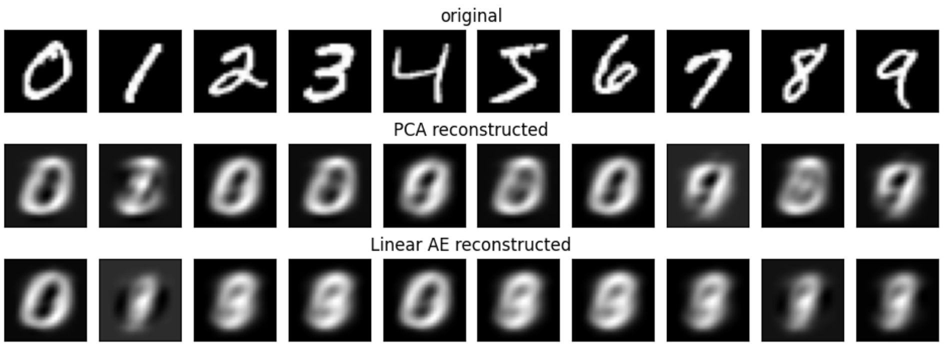

Figure 5: Reconstruction with only 2 components

With these results it's difficult to say which of two did a better job at embedding the structural properties of the different digits. One might even assume that the data structure is simply too complex to be fitted into just two dimensions. But there exist more advanced approaches to this problem [4] that use pretraining and a deep autoencoder architecture where the autoencoder is able to obtain a distinct clustering of the MNIST digits in a two dimensional space and clearly outperforms PCA. With this in mind, a major advantage of autoencoders over PCA is that autoencoders have different hyperparamters, loss functions and net architectures that can be adapted, whereas the results of PCA are static for a given dataset.

4 Conclusion

We introduced autoencoders in their most simple form, and established a connection to the principal components that can be obtained by factorizing the covariance matrix of the data. Both methods have the goal to find a compressed and meaningful representation of the data. As the theory suggested both approaches obtained similar reconstruction and embedding results when a linear autoencoder is used. We've seen that using an activation function improves the reconstruction error and is also able to recover a cleaner version of the input images. This paper was meant to give an introduction to the broad topic of dimensionality reduction, and to motivate how autoencoders with an advanced architecture can be a really powerful tool for many different applications.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[2] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294, 1988.

[3] E. Plaut. From principal subspaces to principal components with linear autoencoders, 2018.

[4] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[5] Q. Fournier and D. Aloise. Empirical comparison between autoencoders and traditional dimensionality reduction methods. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 211–214, 2019.

[6] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR.

[7] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Netw.*, 2(1):53–58, jan 1989.