African Master in Machine Intelligence (AMMI) Report: Equivalence between Principal Component Analysis (PCA) and Linear Autoencode (LAE), Denoising Autoencoder (DAE) and Contractive Autoencoder(CAE)

Group 6: John Kamwele Mutinda, Dieu-Donne Fangnon, Mame Diara Diouf, Khady Gaye, Regis Konan Marcel Djaha, Phanie Dianelle Negho, Mouhamed El Mamoune DIEYE

Machine Learning Foundation course by: **Prof. Moustapha Cisse**

1 Equivalence between Principal Component Analysis (PCA) and Linear Auto Encoder (LAE)

We have familiarised ourselves with PCA and autoencoder as tools for dimensionality reduction used in both data analysis and machine learning. They both have similarities and differences, and while there are some equivalences between them, they are not entirely equivalent. We know that PCA is a linear transformation technique that maps high-dimensional data onto a lower-dimensional space. It seeks to find the principal components of the data, which are the directions along which the data varies the most. These principal components are orthogonal, which means they are uncorrelated with each other. PCA finds the linear combinations of the original variables that maximize the variance of the data in the new coordinate system. On the other hand we are familiar with encoder a neural network architecture that can be used for dimensionality reduction, feature extraction, and data compression. It consists of two parts: an encoder and a decoder. The encoder maps the input data to a lower-dimensional latent space, while the decoder maps the latent space back to the original space. Autoencoders can be used for unsupervised learning and can learn more complex, non-linear transformations compared to PCA. There are several special types of autoencoders that are commonly used in machine learning and deep learning and one of them is linear autoencoder. In this work we seek to explain the equivalence between PCA and a linear autoencoder using theoretical, mathematical and analytical approaches.

1.0.1 Theoretical Equivalence between PCA and LAE

In this section we will look on equivalence between PCA and LAE in theoretical approach

i PCA and LAE are both based on the idea of projecting high-dimensional data onto a lower-dimensional space. In PCA, this is done by finding the principal components that explain the most variance in the data. In autoencoder, this is done by learning a compressed representation of the input data.

ii Both can used for data preprocessing and feature extraction. By reducing the dimensionality of the data, they can help reduce noise and improve the performance of machine learning algorithms.

iii Both PCA and (LAE) aim to minimize the reconstruction error between the original data and the lower-dimensional projection.

iv PCA and LAE both use a linear transformation to map the input data to a lower-dimensional space. In PCA, this transformation is achieved by finding the eigenvectors of the covariance matrix, while in LAE, the transformation is learned by training a neural network.

v Both techniques can be used for data visualization, by representing the high-dimensional data in a lower-dimensional space that is easier to visualize.

vi Both techniques can effectively denoise high-dimensional data by removing the noise in the high-dimensional space and representing the clean data in a lower-dimensional space, which is easier to work with and analyze.

vii PCA and LAE can both be used for unsupervised learning, as they do not require labeled data for training.

viii In PCA the subspace spanned by Principal components is the same subspace spanned by LAE which use linear activation function, and Squared loss function

1.0.2 Mathematical Equivalence between PCA and LAE

In this part we will prove that the loss functions of PCA and LAE are indeed equivalent Lets recall from PCA that the reconstructed data matrix is given by

$$\hat{X} = WX + \mu$$

Where W is the matrix of eigenvectors of the covariance matrix of the standardized data,X is the standardised data matrix and μ is the mean of the original variables. In PCA we seek to minimise the reconstruction error such that the loss function is defined as

$$\begin{aligned} \mathcal{L}(X, \hat{X}) &= ||X - \hat{X}||_2^2 && (1) \\ &= ||X - WX - \mu||_2^2 && (2) \end{aligned}$$

Considering a linear autoencoder which takes d dimension input with one hidden layer such that

$$Z = f(W_1 X + b_1) \quad \textbf{and} \quad \hat{X} = g(W_2 Z + b_2) \tag{3}$$

The loss function for linear autoencoder which aims to minimize the reconstruction error is given by

$$\begin{aligned} \mathcal{L}(X, \hat{X}) &= ||X - \hat{X}||_2^2 && (4) \\ &= ||X - W_2 W_1 X - (b_2 + W_2 b_1)||_2^2 && (5) \end{aligned}$$

And therefore, for linear autoencoder we seek to solve the optimization problem

$$\begin{aligned} \underset{W_2, W_1}{\text{minimize}} \quad & ||X - W_2 W_1 X - (b_2 + W_2 b_1)||_2^2 \\ \text{subject to} \quad & W_2 W_1 = I \end{aligned} \tag{6}$$

The optimization problem (6) is equivalent to (1) and therefore without loss of generality, we conclude that PCA is equivalent to linear autoencoder [1]

1.0.3 Analytical Equivalence between PCA and LAE

We experiment the equivalence between PCA and LAE using a selected data set. Iris data set was used for this case. This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) stored in a 150x4 numpy.ndarray.The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width which form the features. Before performing PCA and LAE, the data was first rescaled. Scaling ensures that features are comparable: If the input features have different scales, then some features may have a larger impact on the principal components or encoding weights than others. Rescaling the data ensures that each feature is on a similar scale, which avoids this issue.

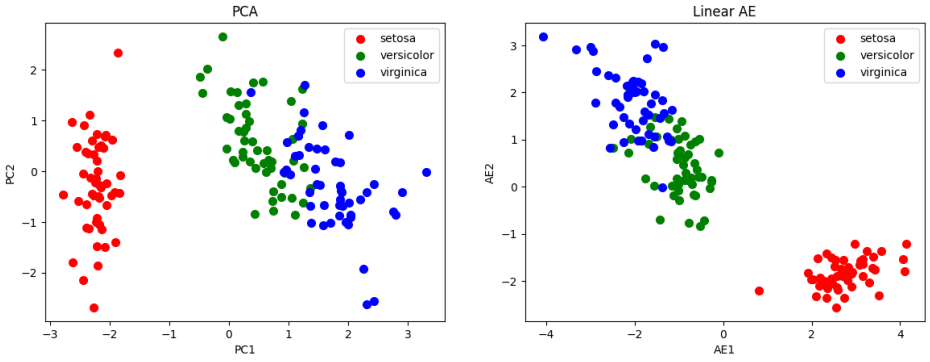

Figure 1: PCA and LAE plots of the first two components

From the results as shown in (1), we can see that both the PCA and linear autoencoder methods produce two components that capture the most important features of the iris dataset. The graphs show the direction and magnitude of these principal components. The plots show similar clustering patterns, this suggests that PCA and linear autoencoder are producing similar low-dimensional representations of the data. This equivalence can be explained by the fact that PCA and linear autoencoder are both linear techniques that seek to capture the main patterns in the data. Therefore, we can conclude that PCA and linear autoencoder are equivalent in the sense that they can both be used to extract principal components from a dataset. The implication of the graphs is that both PCA and linear autoencoder produce similar results, indicating that the linear autoencoder can be used as an alternative to PCA for principal component analysis. Furthermore, the graphs also show that the components learned by both methods are able to separate the different classes of the iris dataset, which is the main goal of dimensionality reduction methods such as PCA and linear autoencoder.

2 Denoising autoencoder(DAEs)

An undercomplete hidden layer can be used for compression as we are encoding the information from input in fewer dimensions. On the other hand, in an overcomplete layer, we use an encoding with higher dimensionality than the input. This makes optimization easier. In DAE, we need to

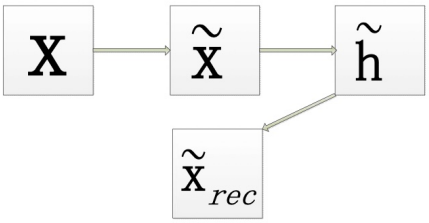

Figure 2: The architecture of denoising autoencoder

corrupt the original data firstly. There are two common ways to corrupt the original data: Additive isotropic Gaussian noise and binary masking noise.

2.0.1 Different type of noises of auto encoders

i **Additive isotropic Gaussian noise**
Here, we denote the corrupted version of the original input sample X as $\tilde{X}$. Then, for additive isotropic Gaussian noise, each component is drawn i.i.d. from a pre-specified noise distribution with mean zero and variance σ^2 . its mathematical formular is giving by:

$$\tilde{X} = X + \mathbf{N}(0, \sigma^2 I) \tag{7}$$

where σ is a small constant maybe usually smaller than 0.5 and $\tilde{x}$ represent our corrupted input. And I is the identity matrix, i.e., with all the elements in the diagonal equal 1 while other elements all equal 0. This ensures that the expectation over the noise remains unbiased, i.e.

$$\mathbf{E}(\tilde{X}) = X \tag{8}$$

Adding noise in the feature is similar to doing an Tikhonov regularisation of our data

ii **Binary masking noise**
In this case, a small fraction of the input sample X is set to be zero, and the fraction can take values on 10%, 25% etc. Noise injection in the input data is the key ingredient of a DAEs. We can extend this idea to apply noise to the hidden units and visible units of a neural network. This creats a computationally inexpensive but powerful regularization—dropout. The term "dropout" means dropping out units (visible and hidden) in a neural network. Dropping a unit out means temporarily removing it from the network together with all its incoming and outgoing connections. The choice of which units to drop is random. Similar to the DAEs, it also can be considered as a process of constructing new inputs by multiplying with noise. As noise is applied to the hidden units, dropout can be seen as performing dataset augmentation at multiple levels of abstraction. In our study, we are going to implement with the Additive isotropic Gaussian noise.

2.0.2 Mathematical formulation of DAEs

We will perform the feed-forward dense layer and the encoder and decoder functions are obtained as follows: Giving the training data constisting of pairs $(X, X), i = 1, ..., N$, where $\tilde{X}$ is the corrupted version of the training data $X \in \mathbf{R}^D$

$$\tilde{h} = \sigma(W_1\tilde{X} + b) \tag{9}$$

Where the $w_1 \in \mathbf{R}^{H\times D}$ and b are the same giving by AEs, and σ is the activation function.
Similarly, the decoder part of our DAEs give us :

$$\hat{X}_{rec} = \sigma(W_2\tilde{h} + c) \tag{10}$$

where $w_2 \in \mathbf{R}^D \times H$ and c is a bias
w_1 and w_2 are weights of the network with hidden dimensionality H And after that, performing the backpropagation in order to compute our loss function as following:

$$L_{DAEs} = \frac{1}{N}\sum_{X\in D}(X - \tilde{X}_{rec})^2 \tag{11}$$

N here represent the number of observation and D the sample of our feature. So, the optimization criterion here is to minimizing this loss function where X the reel input and and X_{rec} is the reconstruction. We can say that DAE increase its robutness by stochastilly traning the model for reconstruction.

2.0.3 Analytical Implementation of DAEs

We used MNIST dataset which was first corrupted to introduce noise to show a denoising encoder works on corrupted data and generate a denoised output. (see link for more details.)

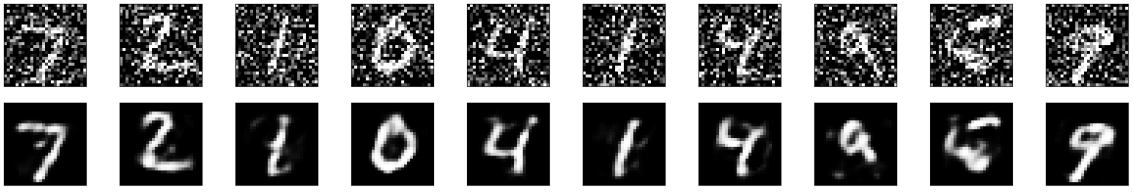

Figure 3: Results before and after denoising the input

The figure (3) shows the result of the model. The first row shows the blurred images are the noisy or the corrupted input that have been fitted to the model. The second row is cleaned or the denoised images which are generated as the output of the DAE . It appears that the model is able to remove the noise and reconstruct the original data with some precision. But, as we see the reconstruction is not completely good. In order to compute the accuracy to the reconstruction, we used the PSNR that computes the peak signal-to-noise ratio (in decibels) between the output and the original input. The PSNR gives a value of 39.53 db that a good result for the reconstruction because in the range of 30 and 50 db. Therefore, it appears that we need to improve to model to have a better reconstruction.

3 Contractive autoencoder (CAEs)

The contractive auto-encoder (CAEs) is another revolutionary advancement in the AEs discipline. This model is created by modifying the conventional reconstruction cost function with a carefully designed penalty term. Furthermore, this penalty term matches the Jacobian matrix of the encoder activations with regard to the input's Frobenius norm. The resulting minimaization of the CAEs can then be expressed as:

$$L_{CAEs} = \sum_{X \in\ D_n} L(X, g(f(X)) + \|J_f(x)\|_F^2 \tag{12}$$

In (12), $J_f(X) = \frac{\partial f}{\partial X}$ is the regularization term that corresponds to the Jacobian of the hidden representation h with respect to the input x. When there is a slight change in the input, this derivative will show us how much our hidden variate changes. Therefore, by minimizing this function, the magnitude of the encoder's first Jacobian matrix is minimized. This will assist us in controlling the loss or in making our loss robust, invariant, or stable so that the encoder does not change even when the input is variable. Additionally, λ is a hyper-parameter controlling the strength of the regularization. The frobenius norm is calculated based of the activation function that we used. Lets consider a case where sigmoid is used as an activation function to capture non-linearity in both encoding and decoding. Contractive provides the implementation of the auto-encoder with a loss of 2 terms: the Reconstruction and the Contraction or Jacobian of encoder (12) Where f is the encoding function g is the decoding function L is the MSE loss function.

$$\|J_f(x)\|_F^2 = \sum_{i,j} (\frac{\partial h_j(x)}{\partial x_i})^2$$

3.0.1 Mathematical Implementation of frobenious norm using the jacobian matrix of the partial derivatives of encoder outputs with respect to inputs

Lets consider the input of the encoder as x_i and output of the encoder as h_i which comes from the sigmoid activation function as shown in equation (13), we aim to compute the partial derivatives of the encoder output with respect to its inputs

$$\begin{aligned} z_j &= W_i x_i + b_j \\ h_j &= sigmoid(z_j) = \sigma(z_j) \end{aligned} \tag{13}$$

Computing the partial derivatives, $\frac{\partial h_j}{\partial x_i}$

$$\begin{aligned} \frac{\partial h_j}{\partial x_i} &= \frac{\partial (z_j)}{\partial x_i} \\ &= \frac{\partial (z_j)}{\partial z_j} \times \frac{\partial z_j}{\partial x_i} \\ &= (z_j)(1 - (z_j))w_i \\ \frac{\partial h_j}{\partial x_i} &= h_j(1 - h_j)w_i \\ \|J_f(x)\|_F^2 &= \sum_{i,j} \frac{\partial h_j}{\partial x_i} = \sum_{i,j} [h_j(1 - h_j)]^2 [w_{ji}]^2 \end{aligned}$$

Therefore, for the case of a sigmoid non linearity, the penalty on the Jacobian norm takes the following expression:

$$\|J_f(x)\|_F^2 = \sum_{j=1}^{d_h}[h_j(1-h_j)]^2 \sum_{i=1}^{dx}[w_{ji}]^2$$

Where d_h is the number of hidden dimension of the encoding layer, d_x is the number of input dimension and w is of shape $[d_h, d_x]$

3.0.2 Analytical Implementation of Contractive Auto Encodersc(CAE)

The MNIST dataset was also used to illustrate how contractive auto encoders works

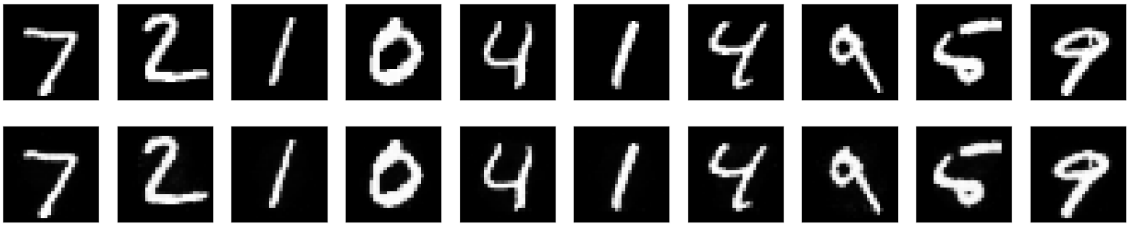

Figure 4: Results before and after denoising the input (non-contractive AE)

Figure (4) is the result of a non-contractive AE i.e an overcomplete AE without a regularization term. The first row shows the original input and the second row shows the reconstructed input. It is observed that the input and output are the same, the differences are not obvious. The PSNR is 50.49 db, this value is out of range and above 50 db. The PSNR shows the model is highly capable of reconstructing the input and it does so better than the input.

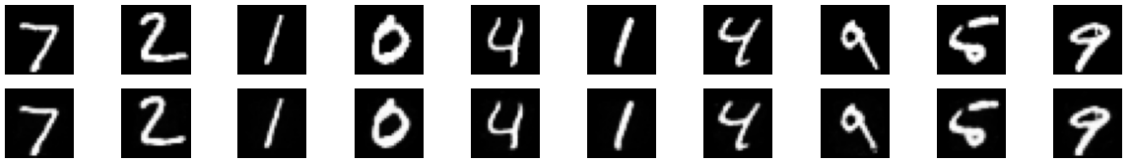

Figure 5: Results before and after denoising the input (contractive AE)

Figure (5) shows the result of a contractive autoencoder. It is observed that from the first row (the original input) and the second row (the reconstructed input). These two (input and output) are quite similar. The computation of the PSNR gives 49.69 db, this very good result and it means that the model is able capture most of the important features. By comparing the this result and the PSNR of the non-contractive AE, we can conclude that the regularization term force the encoder to capture only the important features by preventing overfitting.

3.1 Conclusion

In this work, we have been able to examine the equivalence between PCA and LAE, for PCA and LAE to be equivalent, the encoder and decoder needs to be linear. Both PCA and LAE aimed in minimising the reconstruction error between the original data and reconstructed data. In both cases the data has to be standardized to the same scale before implementation. It has also been observed from the results that, both PCA and LAE gives similar results for the Iris selected data sets, actually the plot for both PC1 and PC2 and AE1 and AE2 have shown that there exists three clusters in the data set. The clusters formulate themselves in closer manner for Versicolor and Virgnica flower species while setoca flower species isolates itself from other flower species. . We have also looked into denoising auto encoder and contractive auto encorder The denoising autoencoder has proven to be a vibrant tool in increasing the robustness of the encoder to the small changes in the training data which was quite similar to the motivation of Contractive Autoencoder. However, the noted differences between the DAE and CAE is: CAEs encourage

robustness of representation encoder, whereas DAEs encourage robustness of reconstruction, which only partially increases the robustness of representation. On the other hand, DAE increases its robustness by stochastically training the model for the reconstruction, whereas CAE increases the robustness of the first derivative of Jacobian matrix.

References

[1] Jangamreddy, Nikhil (2019). *Visualizing and Understanding the Relationship between K-Means Clustering, PCA and Linear Autoencoder.* , Researchgate

[2] Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y. (2011, June). *Contractive auto-encoders: Explicit invariance during feature extraction. In Proceedings of the 28th international conference on international conference on machine learning (pp. 833-840).* , Dept. IRO, Université de Montréal. Montréal(QC), H3C 3J7, Canada

[3] Rifai, S., Vincent, P., Muller, X., Glorot, X., Bengio, Y. (2011, June). *Contractive auto-encoders: Explicit invariance during feature extraction. In Proceedings of the 28th international conference on international conference on machine learning (pp. 833-840).* , Dept. IRO, Université de Montréal. Montréal(QC), H3C 3J7, Canada

[4] Umberto Michelucci (2022) *An Introduction to Autoencoders* , TOELT.AI.

[5] Alireza Makhzani (2018) *Unsupervised Representation Learning with Autoencoders*, Ph.D. thesis, University of Toronto.

[6] William L. Hamilton (1994) *Fundamentals of Machine Learning, Lecture 25 — Autoencoders and self-supervision.*