Auto-encoders

Jean-Sébastien Delineau, Eugene Mettraux, Xiao Xiang

1 Principal component analysis (PCA)

PCA is a method of orthogonal projection of data onto lower-dimensional space. It is a geometric and statisitical approach, geometric since the variables are represented in a new coordinate system and statistic since it looks for the independant axis maximizing the variance of the data. It produces the same number of principal components as there are features in the training dataset. The primary component is determined to account for the maximum variance in the initial features. The subsequent component is perpendicular to the first and captures the most remaining variance after considering the first component.

1.1 Mathematical formulation

Let $\{x_i\}_{i=1}^N$ be our set of observations, and $\mathbf{X} = [x_1, \cdots, x_N] \in \mathbb{R}^{n \times N}$ our feature matrix such that $n \leq N$. Assuming our feature matrix is centered (otherwise consider $\mathbf{X_0} = \mathbf{X} - \bar{x}\mathbb{1}_N^T$). Using that $\mathbf{X}$ is not random, we get that its covariance matrix is:

$$\text{cov}(\mathbf{X}, \mathbf{X}) = E[(\mathbf{X} - E[\mathbf{X}])(\mathbf{X} - E[\mathbf{X}])^T] = E[\mathbf{X}\mathbf{X}^T] = \mathbf{X}\mathbf{X}^T.$$

The L_2 unitary vector corresponding to its greatest variance is:

$$p_1 = \max_{w_1} \text{cov}(w_1^T\mathbf{X}, \mathbf{w}_1^T\mathbf{X}) = \max_{w_1} w_1^T\mathbf{X}\mathbf{X}^T w_1 \quad \text{s.t} \quad w_1^T w_1 = 1.$$

The second one is obtained by considering the feature matrix minus the orthogonal projection of the feature matrix onto p_1:

$$p_2 = \max_{w_2} w_2^T(\mathbf{X} - p_1 p_1^T\mathbf{X})(\mathbf{X} - p_1 p_1^T\mathbf{X})^T w_2 \quad \text{s.t} \quad w_2^T w_2 = 1.$$

A recursive proof shows that $\{p_1, \cdots, p_n\}$ are the eigenvectors of the symmetric positive definite matrix $\mathbf{X}\mathbf{X}^T$. Since it is symmetric positive definite, one can retrieve the PCAs with the spectral theorem. Given that it is a costly computing operation one rather use the Singular-value-decomposition on $\mathbf{X}$:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T \implies \mathbf{X}\mathbf{X}^T = \mathbf{U}\Sigma\Sigma^T\mathbf{U}^T \quad \mathbf{U} \in R^{n \times n}, \mathbf{V} \in \mathbb{R}^{N \times N}.$$

Where Σ is the diagonal matrix with diagonal elements corresponding to the singular values $\{\sigma\}_{i=1}^n$ ordered in descending order. Such a procedure result in the same eigendecomposition of $\mathbf{X}\mathbf{X}^T$. It is unique up to permutations when the columns of $\mathbf{U}$ are normalized and the singular values are ordered by descending order. Given that we only have to perform a SVD decomposition and interact with $\mathbf{U} \in \mathbb{R}^{n \times n}$, this procedure is preferred when $n \ll N$. Since the columns of $\mathbf{U}$ are orthogonal, a projection on the space spanned by the first m eigenvectors corresponding to the greatest variance of the feature matrix is:

$$\mathbf{U_m}(\mathbf{U_m}^T\mathbf{U_m})^{-1}\mathbf{U_m}^T\mathbf{X} = \mathbf{U_m}\mathbf{U_m}^T\mathbf{X} \quad \text{with} \quad \mathbf{U_m} = [u_1, ..., u_m].$$

It can be shown by Ekart-Young theorem[1] that U_m is a solution to:

$$\min_{\mathbf{W} \in \mathbb{R}^{n \times m}} ||\mathbf{X} - \mathbf{W}\mathbf{W}^T\mathbf{X}||_F^2 \quad \text{s.t} \quad \mathbf{W}^T\mathbf{W} = \mathbf{I}_{m \times m} \quad \text{F is for Frobenius norm.} \tag{1}$$

2 Auto-encoders

Autoencoders, a type of artificial neural networks, aim to reconstruct its own inputs for representing informatively the data in lower dimensions [2]. The process first involves encoding the input data into a representation through some lower-dimensional central hidden layers. Subsequently, it targets to recover the input from such a representation, thereby preserving the essential information.

Autoencoders are first introduced as a nonlinear generalization of the PCA method [2]. This non-linearity can be leveraged for the learning of more powerful generalizations, and the reconstruction of the input data with significantly lower information loss [3].

2.1 Mathematical formulation

Let $\mathbf{x}_i \in \{\mathbf{x}_j\}_{j=1}^N$ be an input vector in the space $\mathbb{R}^n$ and $\mathbf{y}_i \in \mathbb{R}^n$ be the output vector of the autoencoder. A single hidden layer autoencoder with an encoding function and decoding function is defined as:

$$\mathbf{a}_i = f\left(\mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{b}^{(1)}\right) \quad \mathbf{y}_i = g\left(\mathbf{W}^{(2)}\mathbf{a}_i + \mathbf{b}^{(2)}\right)$$

where $\mathbf{a}_i \in \mathbb{R}^m$ is the activation of the hidden layer, $\mathbf{W}^{(1)} \in \mathbb{R}^{m\times n}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{n\times m}$ are the weight matrices, $\mathbf{b}^{(1)} \in \mathbb{R}^m$, $\mathbf{b}^{(2)} \in \mathbb{R}^n$ are the bias vectors, and $f : \mathbb{R}^m \to \mathbb{R}^m$, $g : \mathbb{R}^n \to \mathbb{R}^n$ are the activation functions. We note that, to effectively reconstruct the input data, it is designed that $m < n$, that is, the central hidden layer has lower dimensions.

2.2 The training of an autoencoder task

The goal of the auto-encoder task is to learn the parameters $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}$, the weights matrices and bias vectors, such that $\mathbf{y}_i = g(\mathbf{W}^{(2)} f\left(\mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}) \approx \mathbf{x}_i$, i.e., the output vector $\mathbf{y}_i$ is a good approximation of the input vector $\mathbf{x}_i$. We can reformulate such a goal as an optimization problem in the least square framework:

$$\underset{\mathbf{W}^{(1)},\mathbf{W}^{(2)},\mathbf{b}^{(1)},\mathbf{b}^{(2)}}{\text{minimize}} \frac{1}{N}\sum_{i=1}^{N}(||\mathbf{x}_i - g(\mathbf{W}^{(2)} f\left(\mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)})||^2) \tag{2}$$

To solve the above optimization problem in practice, one can leverage existing optimization methods. For the sake of completeness, the presentation of the batch gradient descent method and back-propagation regarding our optimization problem is done in Appendix. A.1 and Appendix. A.2.

2.3 Auto-encoders to PCA

We address the equivalence of PCA and linear autoencoders by showing that the minimization problem (2) shares the same solution with that of the PCA. Without loss of generality, we can choose the activation function f, g to be the identity map. Therefore, (2) can be reduced to:

$$\underset{\mathbf{W}^{(1)},\mathbf{W}^{(2)},\mathbf{b}^{(1)},\mathbf{b}^{(2)}}{\text{minimize}} \frac{1}{N}\sum_{i=1}^{N}(||\mathbf{x}_i - \mathbf{W}^{(2)}\left(\mathbf{W}^{(1)}\mathbf{x}_i + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}||^2) \tag{3}$$

Designing $\mathbf{X} = [x_1, \cdots, x_N] \in \mathbf{R}^{n\times N}$ to be the feature matrix, and $\mathbf{A} \in \mathbb{R}^{(m+1)\times n}, \mathbf{B} \in \mathbb{R}^{n\times(m+1)}$, two matrices such that $\mathbf{A}\mathbf{x}_i = \begin{bmatrix}\mathbf{W}^{(1)}\mathbf{x}_i \\ \mathbf{b}^{(1)}\end{bmatrix}, \mathbf{B}\mathbf{a}_i = \begin{bmatrix}\mathbf{W}^{(2)}\mathbf{a}_i \\ \mathbf{b}^{(2)}\end{bmatrix}$[1], we further reduce the problem to:

$$\underset{\mathbf{A}\in\mathbb{R}^{(m+1)\times n},\mathbf{B}\in\mathbb{R}^{n\times(m+1)}}{\text{minimize}} \frac{1}{N}||\mathbf{X} - \mathbf{A}^\top\mathbf{B}^\top\mathbf{X}||_F^2 \tag{4}$$

Where $\mathbf{A}^\top\mathbf{B}^\top \in \mathbb{R}^{n\times n}$ has rank smaller or equal to m.

With few additional steps, as detailed in Appendix. A.3, one can readily prove the equivalence of the minimization problem (1) and (4).

3 Experiments

3.1 Methodology

The data set considered is MNIST[2]. It has a training set of size 60'000 and a test set of 10'000 images. Each images are of size 28 times 28 and are considered as features of size 1 times 784. For

[1]This is allowed as we simply reformulate the question from an affine model setting to a linear model setting
[2]https://pytorch.org/vision/main/generated/torchvision.datasets.MNIST.html

easier computation we normalized the feature matrix and center it to 0. For each method we will first present its reduction propriety on the data and then on a noisy data to highlight its denoising propriety. Let X be our feature matrix and X' its noised version with an i.d.d normal noise of variance 0.4 added to each component of X.

3.2 Result: PCA

The Mean Squared Error (MSE) achieved by the estimator $\mathbf{U_m U_m^T X}$ of $\mathbf{X}$ diminishes non-linearly towards zero with an increasing number of Principal Components (PCs), as illustrated in the left plot of Figure 1. This trend arises from the distribution of the data's variance in a high-dimensional space. The intrinsic geometry of the data suggests its embedding in a lower-dimensional space than the ambient one[3], explaining the non-linear decline with the inclusion of more PCs.

In the middle plot of Figure 1, the Mean Squared Error is computed by the estimator $\mathbf{U_m U_m^T X'}$ of $\mathbf{X'}$. The observable convergence rate indicates that introduced noise disrupts the intrinsic geometry of the data while maintaining a low dimension.

The right plot of Figure 1 presents the MSE obtained by the estimator $\mathbf{U_m U_m^T X'}$ of $\mathbf{X}$. Despite the presence of noise, the PCA procedure prioritizes axes that explain most of the variance, initially aligning with the same axes as the original data due to its stronger dispersion compared to the artificial one.

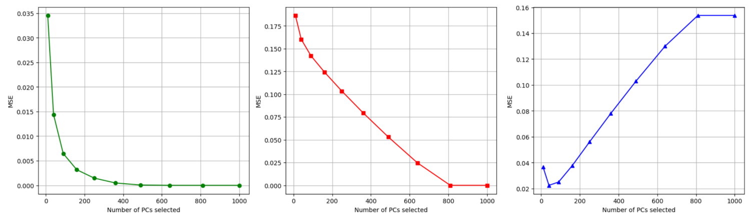

Figure 1: PCA Result

3.3 Result: single hidden-layer linear auto-encoder

Initially, a single hidden-layer linear auto-encoder is examined, demonstrating comparable performance to PCA, offering valuable insights for their theoretical comparison. Section 3.4 introduces more intricate (non-linear) architectures.

The auto-encoder employs Stochastic Gradient Descent (SGD) with a batch size of 64 and a learning rate η of 1, trained for 10 epochs using Mean Squared Error (MSE) as the loss function. Various hidden layer sizes are tested to investigate their impact on auto-encoder performance. Test set errors are not provided, given the small ratio between the training set size and the number of weights, resulting in low risks of overfitting and test errors mirroring training errors.

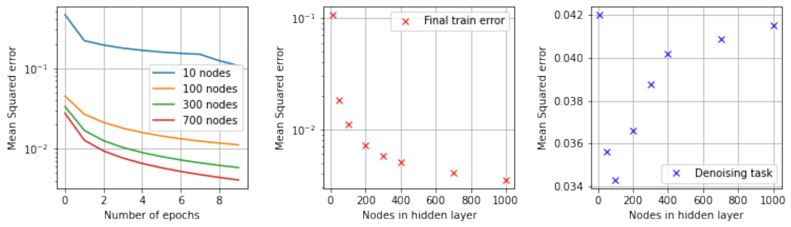

Figure 2: Linear auto encoder results

The middle plot in Figure 2 indicates that the auto-encoder's performance varies notably when adjusting the hidden layer size up to around 200 nodes, after which additional nodes have diminishing impact on the final mean squared error.

In the right plot of Figure 2, a sweet spot for the number of hidden nodes emerges, optimizing the auto-encoder's denoising performance. Striking a balance is crucial, as encoding too little infor-

[3]Reference: https://arxiv.org/pdf/2107.03903.pdf

mation results in poor image recovery, while encoding too much leads to effective noise recovery, unintended for denoising. Our results agree with theory since this sweet spot aligns with the range of 100 to 200 hidden nodes, resembling the number of principal components minimizing the mean square error in the denoising task for PCA (Figure 1).

Figure 3 visually demonstrates denoising performance for two training scenarios: one over the training set (as in Figure 2) and the other over the training set with added noise. Both scenarios utilize mean squared error between the output of the AE and the unnoised data as the loss function. The latter exhibiting enhanced denoising due to fine-tuning during training, achieving a superior denoising outcome.

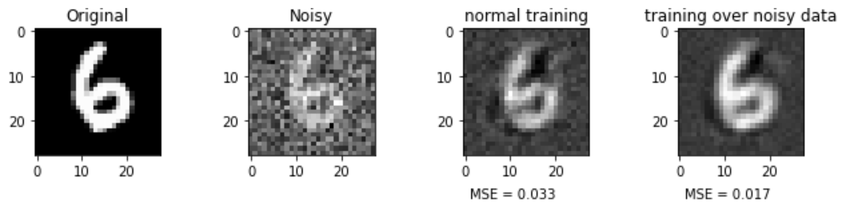

Figure 3: Results of the linear auto encoder on the denoising task for different training approaches

3.4 Result: high architecture auto-encoder

For denoising, the primary advantage of auto-encoders over PCA lies in their ability to employ more intricate, non-linear architectures. This section explores two different auto-encoders with the same hyperparameters as in Section 3.3.

The first auto-encoder features three hidden layers with non-linear activation functions: a fully connected neural network with an input layer of size 784, hidden layers of sizes 264 and 64, each followed by a Relu activation function, and a final output layer of size 784 with a sigmoid activation function.

The second auto-encoder employs a convolutional architecture, with in total 4 convolutions followed by relu activation functions and lastly a sigmoid activation function. Both auto-encoders were trained using the same hyperparameters as in Section 3.3.

Figure 4 can be directly compared to Figure 3. Notably, border pixels are recovered as solid black due to non-linearities, particularly in the ReLU and sigmoid activation functions, which set outputs to zero for negative inputs. The convolutional auto-encoder outperforms the other two, leveraging spatial analysis for improved denoising. Unlike fully connected neural networks, the convolutional approach considers spatial correlations between pixels, enhancing denoising outcomes by being able to analysis the coorelations inbetween the different pixel within the image.

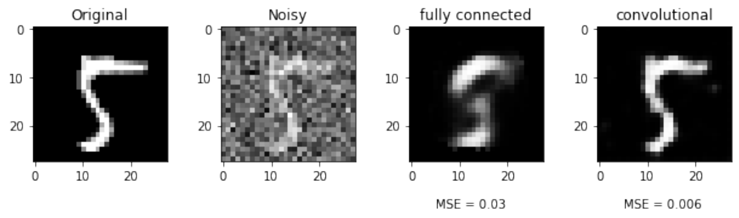

Figure 4: Non linear auto encoders results on the denoising task

4 Conclusion

In conclusion, PCA and a linear auto-encoder with a single fully connected activation function perform equally well on a denoising task, as suggested by theory. Auto-encoders with a narrower bottleneck address a similar problem to PCA, focusing on dimension reduction, and both can be utilized for noise reduction. However, auto-encoders can achieve enhanced performance by leveraging a sophisticated neural network architecture, surpassing the results of PCA methods significantly.

5 References

[1] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, pp. 211–218, Sep 1936.

[2] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE Journal*, vol. 37, no. 2, pp. 233–243, 1991.

[3] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[4] AndrewNg, "Cs294a lecture notes," 2011.

[5] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE Journal*, 1991.

[6] E. Plaut, "From principal subspaces to principal components with linear autoencoders," *ArXiv*, vol. abs/1804.10253, 2018.

A Appendix

A.1 Gradient descent

Gradient descent is a widely used optimization algorithm in machine learning for finding the minimum of a function, often with the purpose of training neural networks. The central idea relies on iteratively moving towards the minimum of the loss function by updating the model's parameters in the opposite direction of the gradient.

The steps invovle first initializing our parameters $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}$. We remark that it is essential to randomize the initiation following a zero-mean normal distribution with low-variance, for the purpose of symmetry breaking.

Then, we follow the gradient descent update:

$$\mathbf{W}_{l,j}^{(k)} := \mathbf{W}_{l,j}^{(k)} - \alpha \frac{\partial}{\partial \mathbf{W}_{l,j}^{(k)}} \mathbf{F}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}),$$

$$\mathbf{b}_{l}^{(k)} := \mathbf{b}_{l}^{(k)} - \alpha \frac{\partial}{\partial \mathbf{b}_{l}^{(k)}} \mathbf{F}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)})$$

where k denotes the numbering of the layer, α the learning rate, and $\mathbf{F}$ the objective function of the minimization problem.(1)

By iterating the steps, we are able to retrieve the solutions of the minimization problem, hence the goal of the autoencoder task[4].

A.2 Backpropagation

Calculating the aforementioned partial derivatives is a key task in autoencoders. This is where backpropogation, a common method applied to iteratively train neural network models, comes in handy for efficient computation.

The backpropogation algorithm can be summarized as follows:

Given a fixed training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i \in \{1,\ldots,N\}}$ of N training examples, we fix on one sample $(\mathbf{x}, \mathbf{y})$ for simplicity. We start by performing a feedforward pass, where we compute the activation of the central hidden layer $\mathbf{a}$ and the the activation of the output layer $\mathbf{y}'$.

For each output unit of the output vector $\mathbf{y}_l'$, we compute the error term defined as:

$$\epsilon_l^{(k=2)} = -(y_l - y_l')[\mathbf{g}'(\mathbf{W}^{(2)}\mathbf{a} + \mathbf{b}^{(2)})]_l$$

Subsequently, we compute for each node of the hidden layer:

$$\epsilon_j^{(k=1)} = (\Sigma_{l=1}^{m} \mathbf{W}_{l,j}^{(1)} \epsilon_l^{(k=2)})[\mathbf{f}'(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})]_j]$$

Given the error terms, we are able to compute the partial derivatives:

$$\frac{\partial}{\partial \mathbf{W}_{j,l}^{(2)}} \mathbf{F}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}) = y_l' \epsilon_l^{(k=2)}$$

$$\frac{\partial}{\partial \mathbf{W}_{l,j}^{(1)}} \mathbf{F}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}) = a_j \epsilon_j^{(k=1)}$$

$$\frac{\partial}{\partial \mathbf{b}_{l}^{(2)}} \mathbf{F}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}) = \epsilon_l^{(k=2)}$$

$$\frac{\partial}{\partial \mathbf{b}_{j}^{(1)}} \mathbf{F}(\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(1)}, \mathbf{b}^{(2)}) = \epsilon_j^{(k=1)}$$

With iteration, we are able to train the samples and obtain the results efficiently.

[4]Since the objective function is non-convex, gradient descent is susceptible to local optima. However, in practice, gradient descent methods work well in general

A.3 Further steps of the proof on the equivalence

We then start to proceed with the remaining proof such that the result from PCA is equivalent to the optimization problem (2):

We first set $\mathbf{Z} = \mathbf{A}^\top \mathbf{B}^\top \mathbf{X}$. By definition, $\text{Im}(\mathbf{Z}) \subseteq \text{Im}(\mathbf{X})$. Additionally, we can check that $\text{rank}(\mathbf{Z}) \leq \min(\text{rank}(\mathbf{X}), \text{rank}(\mathbf{A}^\top \mathbf{B}^\top)) \leq \text{rank}(\mathbf{A}^\top \mathbf{B}^\top)$. We first observe that by the Eckart-Young theorem [1], the matrix $\mathbf{Z} = U_m^\top X$ containing the m principal components from the truncated transformation of PCA satisfies the following optimization:

$$\min_{\mathbf{Z}} \quad \|\mathbf{X} - \mathbf{Z}\|_F^2 \quad \text{s.t.} \quad \text{rank}(\mathbf{Z}) \leq m. \tag{5}$$

and its solution is of the form $\mathbf{Z} = \mathbf{X}_{[m]} = \mathbf{U}_{[m]} \mathbf{S}_{[m]} \mathbf{V}_{[m]}$ with $\mathbf{U} \in \mathbb{R}^{N \times N}$ an orthogonal basis of $\mathbb{R}^N$, $\mathbf{S} \in \mathbb{R}^{N \times n}$ a diagonal matrix, and $\mathbf{V} \in \mathbb{R}^{n \times n}$ an orthogonal basis of $\mathbb{R}^n$ from the single value decomposition.

We then state that the solution to (5) is equivalent to the one to (2) noting that (5) misses only the image inclusion constraint comparing to (2), and that $\mathbf{U}_{[m]}$ is by definition in $\text{Im}(\mathbf{X})$.

Therefore, we conclude that the solution to both problems are equivalent.