MATH-412 SML Project Report
PCA and auto-encoders comparison

Boris Kuznetsov Mattia Mariantoni

December 2020

Contents

Abstract

This paper sets the goal to compare the performance of the principle component analysis (PCA) and the auto-encoder (AE) techniques applied to the financial data compression task. Multiple stocks undergo the compression up to the factor $\alpha = 10$. Then, having passed the compression bottleneck, the data is being decoded back and compared to the initial validation set.

1 Introduction

In order to compare the performance of the principle component analysis (PCA) and auto-encoder (AE) techniques applied to the financial data compression task we first have to set the framework in which both methods can be compared in an efficient way. The subsection 1.1 introduces data sets that we use for the encoding-decoding tasks. Subsections 4.2 and 4.3 explain how both techniques are applied to data.

1.1 Data choice and transformation

We have chosen the following 10 stocks from the hotel industry: 'BYD', 'CHH', 'CVEO', 'PLNT', 'RLH', 'HLT', 'H', 'IHG', 'LVS', 'MGM'. We take stocks from the same industry since we hope to find significant linear correlation (the stronger the linear correlation between the components of a random vector, the more efficiently PCA works) and non-linear dependence (since AE is capable of capturing both linear and non-linear dependencies) between them. Stocks are chosen arbitrarily from the list of hotel industry stocks trading on the NY stock exchange.

The training set covers the period from 07/08/2015 to 05/11/2020. The ensuing two months (from 06/11/20 to 05/01/2021) are taken as a validation (test) set. We convert each stock price time-series to the log-return time-series. Then, we scale the log-return series using the MinMaxScaler so that the largest return is scaled to 1 and the smallest is scaled to 0.

1.2 Encoding and decoding data

We apply both methods (PCA and AE) to the aforementioned 10-dimensional training set of 10 stock scaled log-returns in the period from 07/08/2015 to 05/11/2020 (see the example of the 1-dimensional component of the set on the figure 1). The encoding dimension, denoted d_{encode}, (i.e. the bottleneck dimension) varies from 9 to 1. The compression factor α varies from 10/9 to 10 respectively.

PCA applies the principle component transform choosing d_{encode} new basis vectors explaining most of the data variance. Then, inverse principle component transform is applied.

Auto-encoder forces data to go through the d_{encode}-dimensional bottleneck returning the decoded data onto the output layer. The subsection 4.5 explains in detail what types of auto-encoders are used.

1.3 Activation function

Activation functions, applied to Neural Network can successfully approximate functions (up-to a certain arbitrary error e decided by the user) which does not follow linearity or it can successfully predict the class of a function which is divided by a decision boundary which is not linear. We used the:

$$\text{Rectified Linear Unit (ReLU): } \sigma(z) = (z)_+ := \max(z, 0)$$

The rectified linear unit (ReLU) activation function was proposed by Nair and Hinton in 2010, and ever since, has been the most widely used activation function for deep learning applications with state-of-the-art results to date. It offers the better performance and generalization in deep learning compared to the Sigmoid and Tanh activation functions. The ReLU represents a nearly linear function and therefore preserves the properties of linear models that made them easy to optimize, with gradient-descent method. The main advantage of using the rectified linear units in computation is that, they guarantee faster computation since it does not compute exponentials and divisions, with overall speed of computation enhanced. Another property of the ReLU is that it introduces sparsity in the hidden units as it squishes the values between zero to maximum. However, the ReLU has a limitation that it easily overfits compared to the sigmoid function although the dropout technique has been adopted to reduce the effect of overfitting of ReLUs and the rectified networks improved performances of the deep neural networks.[1] Also one of the reason why we chose ReLU is because empirically it gave us better results.

2 Results

2.1 Qualitative comparison

2.1.1 Explaining 10 stocks by building 5 virtual stocks

We will first compare the two techniques in a most straightforward way:

1. We train both models on the training set. 10-dimensional vector X of scaled log-returns of 10 aforesaid stocks is given to PCA and AEs as an input. There are 1323 realisations of X in the period from 07/08/2015 to 05/11/2020. PCA leaves out only 5 new basis vectors explaining most of the data variance. Similarly, AE's bottleneck has the dimension 5.

   Speaking from the financial perspective, the compressed version of data in both cases is the 5-dimensional time-series of log-returns corresponding to 5 "virtual" stocks capturing as much information about initial 10 stocks as possible.

2. We examine the performance of pre-trained models on the test set from 06/11/20 to 05/01/2021. The results are shown on figures 2 and 3.

2.1.2 "Missing data" task

Now suppose that the fragment (test set) of one stock on our choice is missing. Instead, the missing fragment is set to the average 0.5 value. The figure 4 shows how both techniques recover the missing data having captured correlations between stocks while compressing the training set.

2.1.3 Conclusions

Figures 2, 3, 4 show that both PCA and AE techniques successfully manage to capture the dependence structure between stocks. As we see, if the compression factor $\alpha = 2$ PCA and AE perform in a similar way. However, we have managed to discover the fact that depending on the value of α one approach outruns another one and vice versa.

2.2 Quantitative comparison

2.2.1 Mean squared error

We compare PCA, feed-forward Auto Encoders and deep Auto Encoders using the Mean Square Error metric:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2$$

where the y_i represents the true value of the stock and the x_i represents the value obtained from the low dimensional representation. Figure 5 presents in a grid shape which of the technique performed better while varying the encoding dimensions. Purple corresponds to PCA, green to simple (one layer) feed-forward AE, yellow to deep AE. In the y-axis we loop over the possible bottleneck dimensions for the AE (which also represents the dimensions for the PCA) ranging from 1 to 8. In a deep AE before reducing our signal to the final bottleneck dimension we take an intermediate step represented in the x-axis as intermediate AE dimension which ranges from 2 to 9.
From the way we set up the grid we can deduce the following:

- In any row of the grid the MSE of PCA and simple AE is the same becuase the intermediate AE dimension affects only deep AE.
- For clarity we showed the entire grid, although only some of them are "proper" deep AE those in the upper triangular part of the grid. That is because the intermediate AE dimension should be strictly lower than the bottleneck dimension in a Auto Encoder architecture.

Figure 5 seems to indicate that for lower bottleneck dimension (1 to 4) AE perform better than PCA, with higher dimensions (from 5 to 8) PCA dominates.

2.3 Residual correlation matrix norm

We call the residual correlation matrix the difference between correlation matrices of original stocks time-series and its decoded versions. We have calculated the residual correlation matrices between the original data and approximations and then, to gauge its value, we used two metrics.

The nuclear metric is defined for a $m \times n$ matrix A as:

$$\|A\|_2 = \sup\{\|Ax\| : x \in K^n \text{with } \|x\| = 1\} = \sup\{\tfrac{\|Ax\|}{\|x\|} : x \in K^n \text{ with } x \neq 0\}$$

The Frobenius norm is defined for a $m \times n$ matrix A as:

$$\|A\|_{\mathrm{F}} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\operatorname{trace}(A^*A)} = \sqrt{\sum_{i=1}^{\min\{m,n\}}\sigma_i^2(A)} \text{ with } \sigma_i \text{ singular values of } A.$$

We have seen that for relatively large compression factors $\alpha > 2$ auto-encoders tend to perform better than PCA. Let's choose $\alpha > 2$ and compare the Frobenius norms of PCA, FF AE and Deep AE residual correlation matrices. The results are shown below:

	PCA	FF AE (simple)	Deep AE
$d_{encode} = 4$	2.44	2.26	2.31
$d_{encode} = 3$	3.99	3.68	3.82
$d_{encode} = 2$	4.98	7.51	4.64
$d_{encode} = 1$	7.63	7.62	7.61

For the deep auto-encoder the appropriate intermediate layer dimension is chosen for every d_{encode}.

We observe that for $\alpha > 2$ auto-encoders indeed prove to work better. The deep auto-encoder outruns other compression methods in terms of Frobenius norms of residual correlation matrices for an appropriate choice of an intermediate layer dimension. Note also that $\alpha = 5$ is already a breakdown point of a simple feed-forward auto-encoder: the residual error for $d_{encode} = 2$ doubles compared to $d_{encode} = 1$.

3 Conclusion

With the appropriate choice of an intermediate layer dimension, deep auto-encoders perform best compared to simple feed-forward auto-encoders and PCA techniques for compression factors $\alpha > 2$.

Simple feed-forward auto-encoders proves to work slightly better than PCA for $\alpha > 2$ but break down for large compression factors ($\alpha > 3.3$). It is thus recommended to use deep AEs when it comes to using AEs in financial data compressing.

The PCA technique proves better on average for small compression factors. Also, PCA is way faster, which often plays an important role in industry.

The conclusion is each technique has its own strengths and drawbacks. It is important to evaluate what method is best to apply to the given task every time before using the aforementioned techniques.

4 Appendix

4.1 Figures

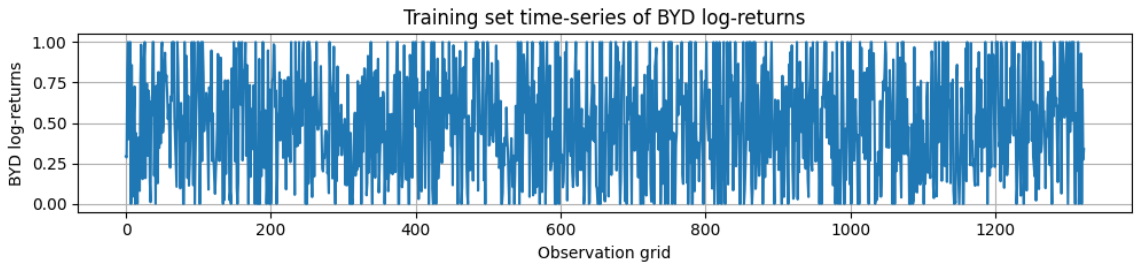


Figure 1: Series of scaled BYD log-returns

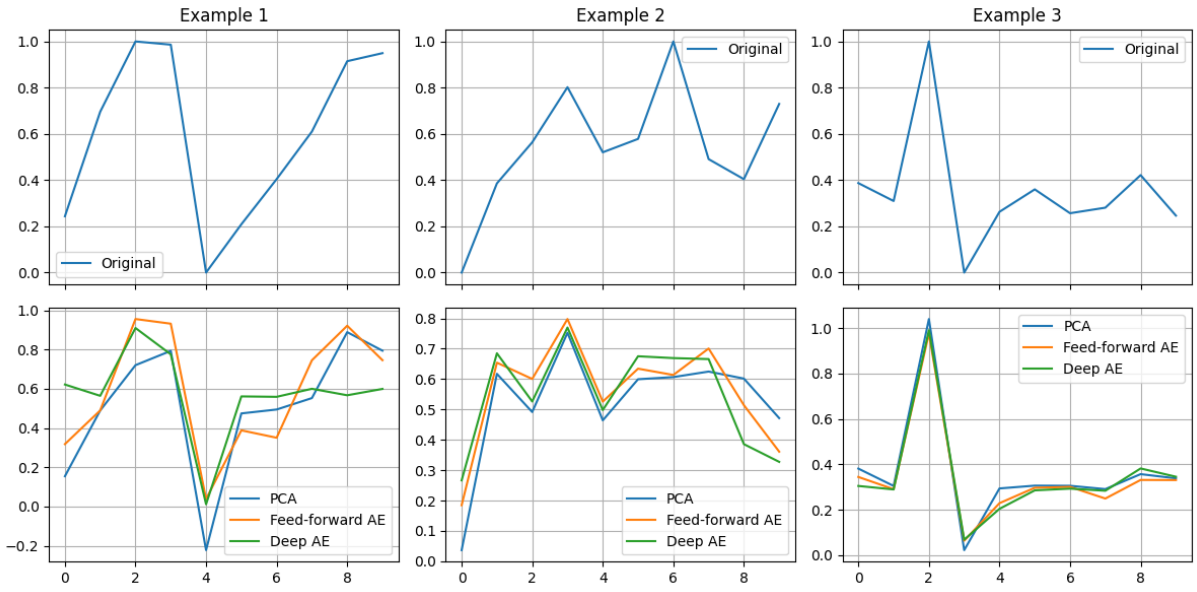


Figure 2: Example of how PCA and AEs recover the given series. Each subplot has 10 points corresponding to 10 stocks at one moment of time.

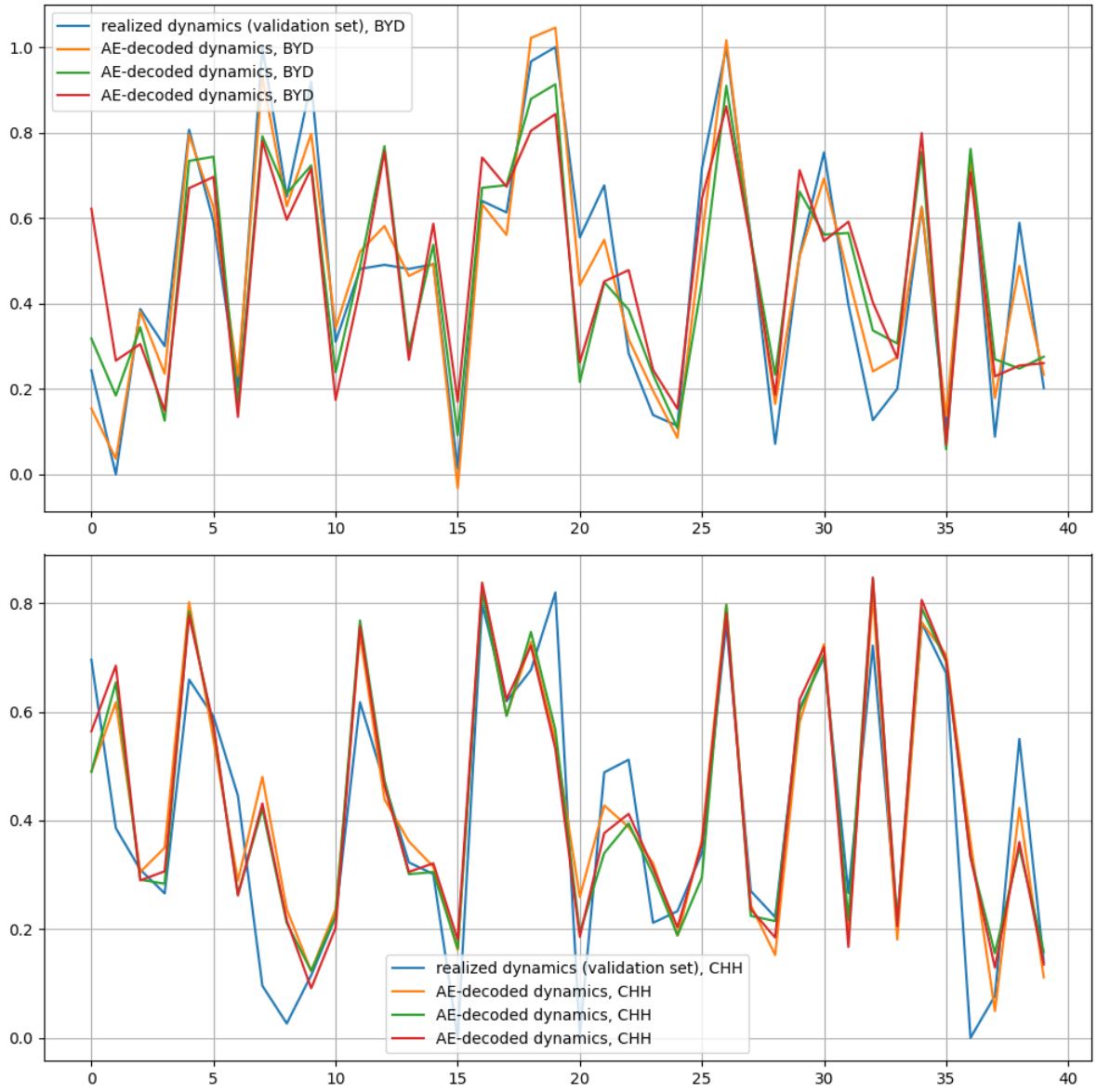


Figure 3: How PCA and AEs recover BYD and CHH stocks time-series, 40 consecutive time points taken, i.e. around 2 calendar months

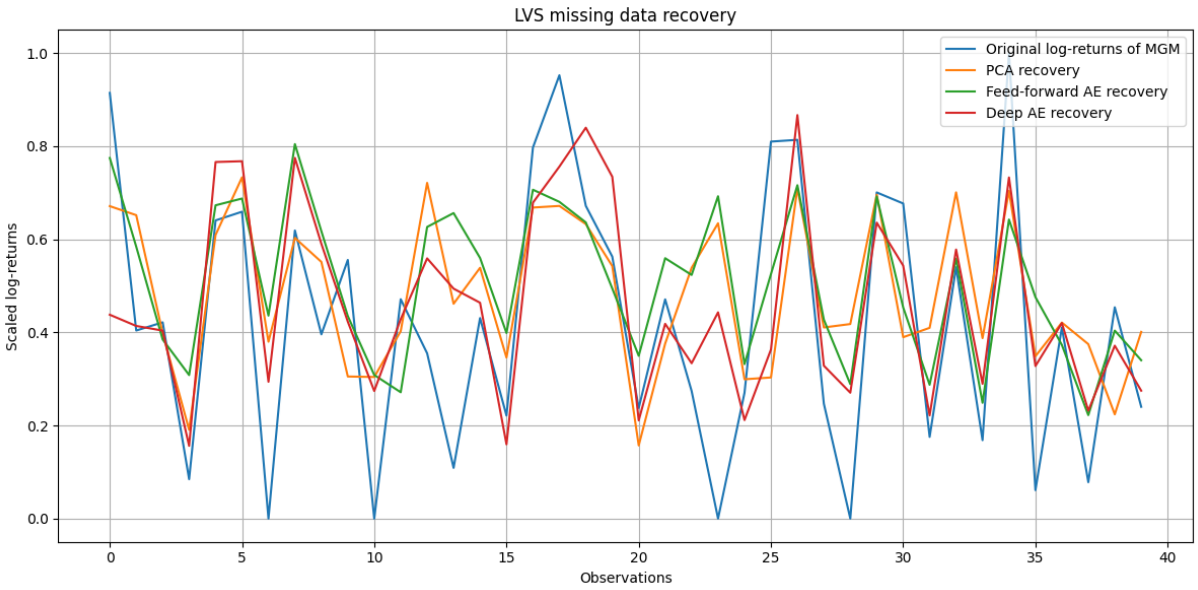


Figure 4: LVS has missing data throughout 2 calendar months. The attempt to recover the missing fragment using 9 other stocks using PCA and AE.

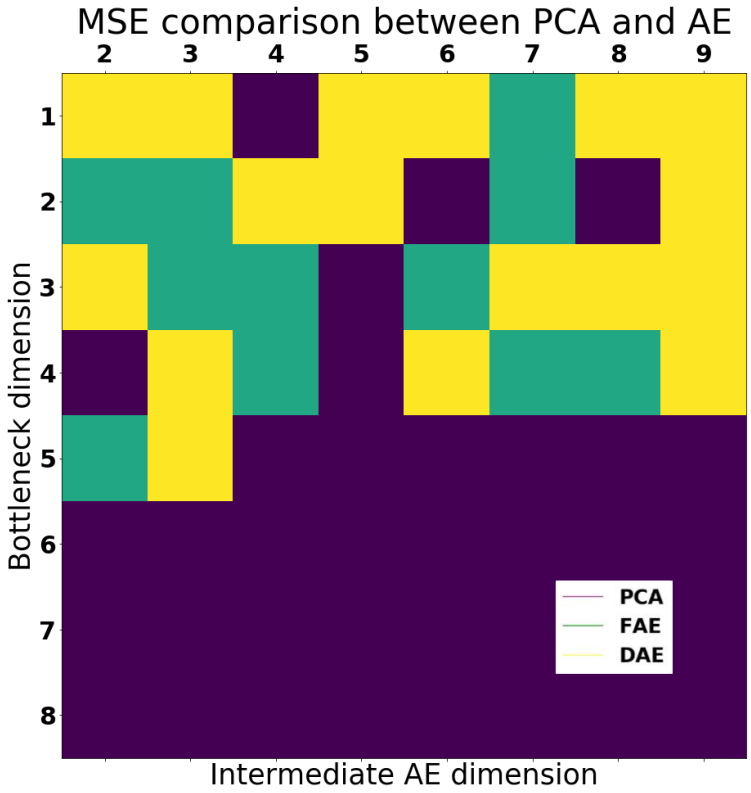


Figure 5: PCA vs AE with MSE metric

4.2 PCA

The central notion of this technique is the principle component transform. Let X be a d-dimensional random vector with mean μ and covariance matrix $\Sigma^{(d\times d)}$. As $\Sigma^{(d\times d)}$ is symmetric, one diagonalizes it applying the spectral decomposition theorem:

$$\Sigma = \Gamma\Lambda\Gamma^T,$$

where Λ is diagonal and Γ is orthogonal. Λ d eigenvalues of Σ and Γ contains d normalized γ_j eigenvectors of Σ. We interchange eigenvalue-eigenvector pairs so that

$$\Lambda_{1,1} \geq \Lambda_{2,2} \geq ... \geq \Lambda_{d,d}$$

The principal component transform of X is then defined as:

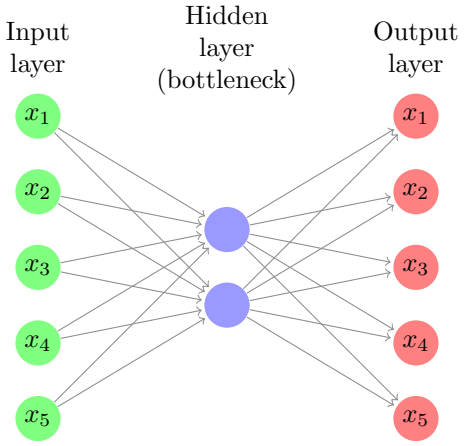
$$Y = \Gamma^T(X - \mu)$$

Now that Σ has undergone the rotation such that all the components of Σ are uncorrelated and new basis vectors have been ordered, one drops eigenvectors starting from the last (i.e. smallest) eigenvalue moving backwards until the desired accuracy-compression trade-off is achieved.

The compressed vector is then being decompressed by applying the inverse principle component transform to recover (decode) the original vector.

4.3 AE

An auto-encoder is a variant of an artificial neural network. Let X_i be a i-th realization of a d-dimensional random vector. The observation goes directly to the input layer of the auto-encoder with shape d. The structure of hidden layers presumes to have a so-called bottleneck. The bottleneck is a narrowest intermediate layer with dimension $d_{encode} < d$. The fact that its dimension is strictly smaller than that of the input layer leads to the compression of the data which comes in.



Having gone through the bottleneck, data is being decoded back by setting the dimension of the output layer to d and then fit directly to input by minimising the error function on our choice.

4.4 Autoencoders and PCA equivalence

Auto Encoder are composed of: an encoding layer which first applies a linear transformation **A** on the input x followed by non-linear activation σ_a, and a decoding layer, which applies another linear transformation **B**

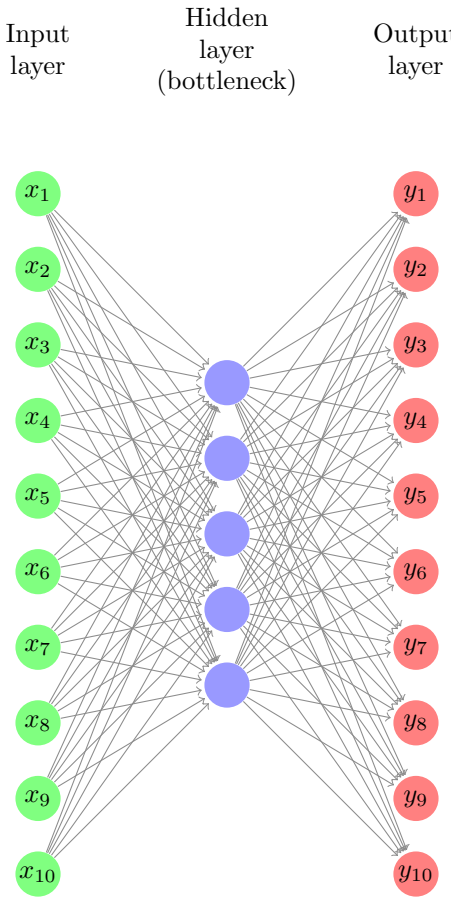
followed by non-linear activation σ_b. Training an auto-encoder consists of minimizing,

$$L(\mathbf{A}, \mathbf{B}) = \frac{1}{n} \sum_{j=1}^{n} \left\| \mathbf{x}_j - \sigma_b \left(\mathbf{B} \left(\sigma_a \left(\mathbf{A} \mathbf{x}_j \right) \right) \right) \right\|^2$$

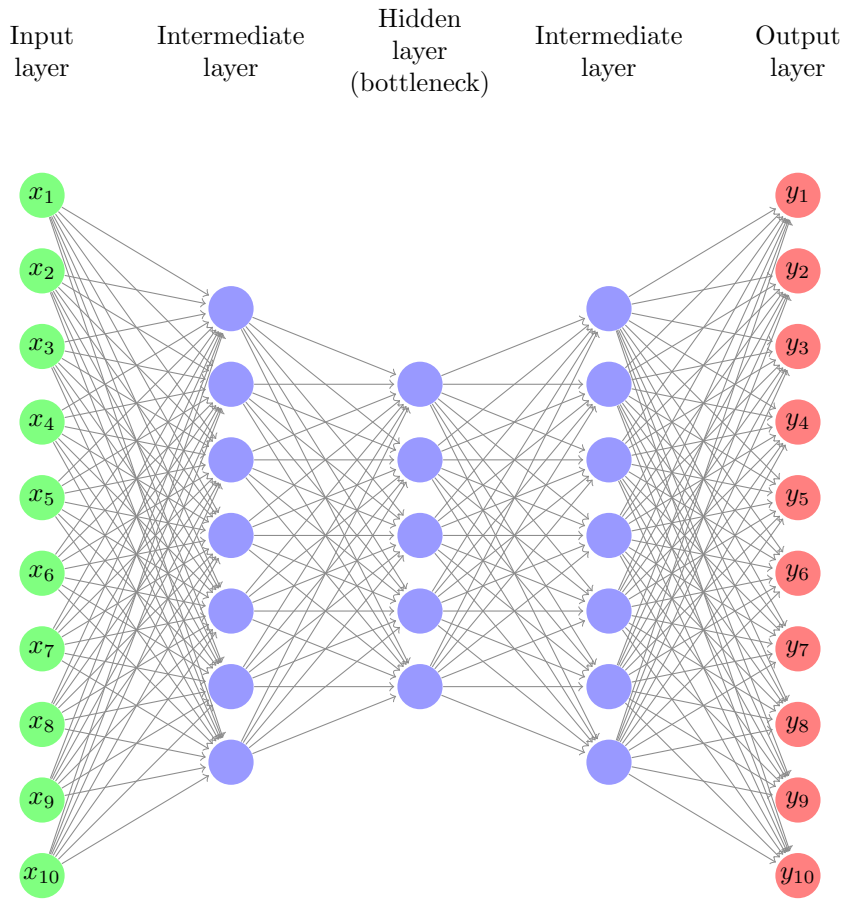
It's possible to show that an auto-encoder as above with linear activation functions is equivalent to PCA.[3]

4.5 On auto-encoders used

We used two different types of auto-encoders in our analysis: simple AE and deep AE. The simple AE, having 10 as input and output dimension and with variable bottleneck dimension, is characterized by the following structure:



The deep AE, having 10 as input and output dimension and with variable bottleneck dimension and variable intermediate layer dimension, is characterized by the following structure:



Using deep encoders and decoders offers many advantages such as:

- Depth can exponentially reduce the computational cost of representing some functions.[2]
- Depth can exponentially decrease the amount of training data needed to learn some functions.[2]
- Experimentally, deep autoencoders yield better compression compared to shallow or linear autoencoders.[2]

References

[1] Activation Functions: Comparison of Trends in Practice and Research for Deep Learning *Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall.* 2018

[2] Deep Learning *Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron* 2016

[3] Statistical machine learning *Guillaume Obozinski; Jonathan Koh; Kartik Waghmare* 2020