

# Rank Reduction Autoencoders - Enhancing interpolation on nonlinear manifolds.

Jad Mounayer<sup>1,2,\*</sup>, Sebastian Rodriguez<sup>1,3</sup>, Chady Ghnatios<sup>1,2</sup>,  
Charbel Farhat<sup>4</sup>, Francisco Chinesta<sup>1,5</sup>

<sup>1</sup>PIMM, ENSAM Institute of Technology, 151 Boulevard de l'Hôpital, 75013, Paris, France.

<sup>2</sup> PIMM Lab, SKF Research Chair, Arts et Metiers Institute of Technology, Paris, France

<sup>3</sup>ESI Group Chair, PIMM, ENSAM Institute of Technology, 151 Boulevard de l'Hôpital, 75013, Paris, France.

<sup>4</sup> Stanford University, Department of Aeronautics and Astronautics  
Department of Mechanical Engineering and Industrial for Computational and Mathematical Engineering,  
496 Lomita Mall, Stanford, 94305, CA, USA.

<sup>5</sup>RTE Research Chair, PIMM, ENSAM Institute of Technology,  
151 Boulevard de l'Hôpital, 75013, Paris, France.

April 13, 2024

## Abstract

Is it possible to reproduce the solution of a physical problem in a parametric space if we only know the solution for a few parameters? In other words, can we interpolate between available solutions to find new ones? This problem was the main reason behind the development of many methodologies (e.g. PODi, optimal transport). Yet, the effectiveness of most currently available techniques is reduced significantly when the problem has many features (i.e. high-rank solution matrices, with many large singular values). In general, High-rank matrices are hard to deal with. For instance, the efficiency of most reduction techniques (e.g. POD, PGD, PCA), or any other low-rank methods decreases when the number of dominant singular values increases. But, is it possible to reduce the dimension of high-rank matrices by finding an approximation of a lower rank?

Linearly, the truncated SVD is our best alternative. However, its reduction capabilities are limited, since we almost need all the features to represent the matrix when many singular values are large. Accordingly, research is directed to find low-rank approximations using nonlinear functions, specifically, Autoencoders with reduced latent spaces. However, a smaller latent space makes it difficult, even impossible in practice, to reconstruct the original matrix. Therefore, some architectures were proposed to expand the data dimensionality in the latent space while enforcing the resulting matrix to

have a low rank (e.g. IRMAE, LoRAE). Their results show that Autoencoders with longer, but low-rank, latent spaces lead to better Autoencoders, specifically for interpolation. In this paper, we present Rank Reduction Autoencoders (RRAE), designed to strongly enforce a low rank on long latent spaces by finding the basis that represents it. We present two formulations, strong/weak ones to achieve what's proposed. Further, we propose two ways (linear/nonlinear ones) to interpolate between the curves in the latent space. We show that both formulations can interpolate between curves with one/two-dimensional parametric spaces even when a linear interpolation is used. Finally, we showcase the effectiveness of RRAEs by using them to interpolate between physical solutions generated by the Avrami crystallization model. We show that RRAEs preceded by a POD can filter noisy data, their interpolating abilities are not reduced if a wrong dimension of the parametric space is chosen, and both the linear/nonlinear interpolation techniques proposed can lead to effective nonlinear interpolation.

*Keywords:* Rank Reduction Autoencoder, Non-linear interpolation, Low-rank latent space, Non-linear model-order reduction, Singular Value Decomposition, Dominant singular values, High-rank matrices.

## 1 Introduction

Numerical methods, such as the Finite Element Method (FEM) [1], or the Finite Difference Method [2] are popular due to their ability to reproduce real physical systems. However, an increase in the complexity of the system directly affects the computational overhead, hence the speed, of these methods. Thus, multiple modalities, such as Model Order Reduction (MOR) techniques, have been presented in the literature to circumvent this problem.

Amongst the most popular MOR techniques are the Proper Orthogonal Decomposition (POD) [3, 4], the Proper Generalized Decomposition (PGD) [5, 6, 7], and the Principal Component Analysis (PCA) [8, 9]. The efficiency of these techniques made them lay the basic principles of many others, specifically those that tackled interpolation between different solutions. For instance, the POD with interpolation (PODI) [10, 11, 12], or the sparse PGD (sPGD) [13, 14], were developed to create high dimensional parametric surrogates, with the ability to reproduce the solution in an entire parametric space with only a few samples.

The methods presented above, as well as every low-rank technique (e.g. [15, 16, 17, 18]), all depend on the ability to represent the space by a finite set of features, which is usually of much smaller dimension than the original feature set. Mathematically speaking, if multiple solutions are stacked into a matrix, only a few singular values of the matrix are dominant compared to the others (i.e. a low-rank matrix). When this assumption is not valid, the presented techniques would provide little to no improvement at all, meaning that the dimensionality of the original problem may not be reduced efficiently. The aforementioned methods, and many other reduction techniques, lose most of their efficiency when dealing with high-rank matrices, making it challenging to create efficient surrogates for the correct prediction of physical phenomena.

On the other hand, machine learning algorithms are being used in almost every field due to their speed and their ability to approximate nonlinear behavior. Countless

model architectures include Multiple Layer Perceptrons [19, 20, 21, 22], Recurrent Neural Networks [23, 24, 25], and Convolutional Neural Networks [26, 27, 28].

The nonlinearity in Neural Networks encouraged researchers to use them for interpolation [29, 30]. One of the techniques that showed a great ability to interpolate is Autoencoders. The idea of an autoencoder is to learn two functions. First, an encoding function moves the data into what we call the latent space, which is usually of a different dimension than the original space. This is followed by the decoding function that brings us back to the original values. Autoencoders have been introduced in the 1990s for dimensionality reduction and feature learning [31]. Their ability to learn generative models of data made them a compelling match with machine learning algorithms. Accordingly, autoencoders with Neural Networks as their encoding/decoding functions have been successfully used in many applications such as speech recognition [32, 33], medical applications [34, 35], robotics [36, 37], and others [38]. In practice, models enforce some characteristics of the latent space to avoid learning the identity function. For instance, larger latent spaces could be used to find Koopman embeddings [39]. On the other hand, a better understanding of the data can be established when the latent space is smaller [40]. Even though Autoencoders appeared to be very promising, their efficiency is limited, especially for data reduction/interpolation (Section 4), since it is usually very hard to reconstruct the original data from a small dimension in the latent space. Accordingly, multiple enhancements, such as Sparse, Denoising, and Variational autoencoders [Citations](#), have been proposed to alleviate these issues. Specifically, the Implicit Rank-Minimizing Autoencoder (IRMAE) [41], and the Low-Rank Autoencoder (LoRAE) [42] showcased how increasing the latent space dimension while enforcing a low-rank achieves better results, including interpolation.

In the following article, we present Rank Reduction Autoencoders (RRAE), which consist of autoencoders that find a reduced basis of the latent space that can be later used for interpolation. This is achieved by the use of two architectures: strong/weak formulations. Furthermore, we propose two ways (linear/nonlinear) to perform interpolation in the latent space. Since the reduced basis is supposed to represent the entire latent space, interpolating between curves becomes more efficient.

The present paper is structured as follows: section 2 presents the architecture and both formulations proposed. Next, we explain our interpolation strategy in section 3. Afterward, we demonstrate, based on a simple example, why an enlarged latent space with a low rank is more efficient than a smaller latent space (regular autoencoders) in section 4. Then, both formulations are tested on a variety of curves in section 5. Subsection 5.1 covers one-parameter examples where we show that RRAEs with both formulations have better interpolation abilities than the regular Autoencoder on challenging examples. The generalization to more parameters is discussed and tested in section ??, where we show that by finding a reduced basis of the latent space, a bilinear interpolation is sufficient! RRAEs are then tested on physical data in subsection 5.2, where we test their abilities to filter noise and learn interpolation when a wrong dimension of the parametric space is given. Finally, we conclude and discuss the future aspects and the limitations of the proposed model in section 6. Our results show that RRAEs with both formulations, and both interpolation techniques, can interpolate on nonlinear manifolds, even when the solution matrix is high-rank.

## 2 Rank Reduction Autoencoders (RRAEs)

The idea of Rank Reduction Autoencoders (RRAEs) is to find a reduced basis that can represent the latent space. To be consistent, we start by defining autoencoder notations.

Let  $\{X_i\}_{i \in [1, D]} \in \mathbb{R}^T$  be a set of  $D$  series of observations, each of length  $T$ . We define our input  $X \in \mathbb{R}^{T \times D}$  with  $X_i$  as its  $i$ th column. Let,  $Y \in \mathbb{R}^{L \times D}$  with  $L$ , the chosen dimension of the latent space. We also define the encoding map  $e : \mathbb{R}^{T \times D} \rightarrow \mathbb{R}^{L \times D}$  and the decoding map  $d : \mathbb{R}^{L \times D} \rightarrow \mathbb{R}^{T \times D}$ . The Vanilla autoencoder can be written as the following two operations,

$$Y = e(X), \quad \tilde{X} = d(Y). \quad (1)$$

In practice, we usually demand that the output of the autoencoder gives us back the original data, hence the loss usually reads,

$$\mathcal{L}(X, \tilde{X}) = \|X - \tilde{X}\|_2, \quad \text{where, } \|\cdot\|_2 \text{ is the L2-norm.} \quad (2)$$

The idea behind RRAEs is to enforce the latent matrix to have a low rank. In other words, let  $Y = U^y \Sigma^y V^y$  be the Singular Value Decomposition (SVD) [43, 44] of  $Y$ , and let  $\{\sigma_i^y\}_{i \in [1, r]}$  be the sorted diagonal values of  $\Sigma^y$ ,  $r$  being the rank of  $Y$ . We want the following,

$$\sigma_k^y \gg \sigma_j^y, \quad \forall j \in [k+1, r], \quad k \ll r. \quad (3)$$

Another way to define the requirement is by the truncated SVD,

$$Y = \sum_{i=1}^r \sigma_i^y U_i^y (V^y)_i^T \quad \Rightarrow \quad Y \approx \sum_{i=1}^k \sigma_i^y U_i^y (V^y)_i^T, \quad (4)$$

where  $U_i^y$  is the  $i$ th column of  $U^y$  and  $(V^y)_i^T$  is the  $i$ th row of  $V^y$ . In other words, we can write  $Y_d$ , the  $d$ th column of  $Y$  as,

$$Y_d = \sum_{j=1}^k \alpha_j^d W_j, \quad \text{with } \alpha_j^d \in \mathbb{R}, \quad W_j \in \mathbb{R}^L, \quad \forall d \in [1, D]. \quad (5)$$

In other words, for a specified number of nodes  $k$ , each column of  $Y$  is defined by  $k$  constants and  $k$  vectors. In other words, the vectors  $W_j$  form a basis for the latent space. By stacking all the vectors  $W_j$  as columns of a matrix  $W$ , we write (5) in matrix form as follows,

$$Y = (A \cdot W^T)^T, \quad \text{with: } A_{i,j} = \alpha_j^i, \quad A \in \mathbb{R}^{D \times k}, \quad W \in \mathbb{R}^{L \times k}, \quad (6)$$

with  $(\cdot)$  denoting the dot product. Based on (5), and (6), we propose two formulations to enforce the latent space to have a reduced basis. The architecture is sketched in figure 1.

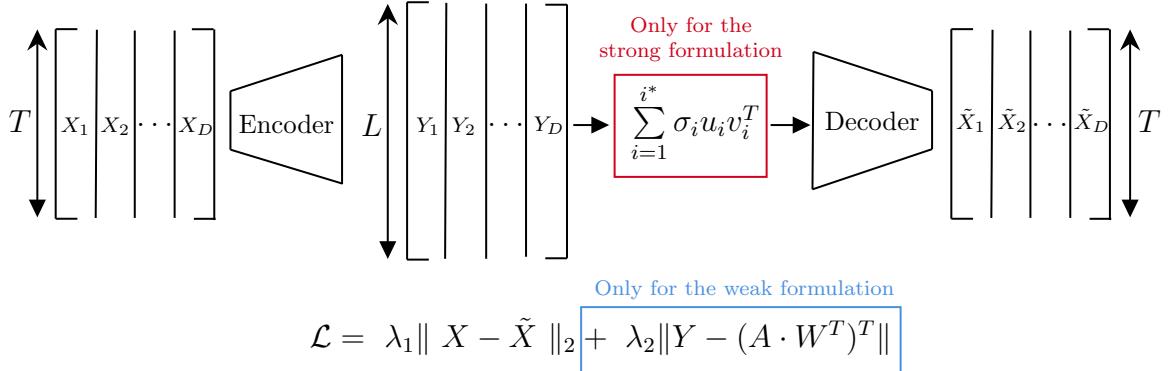


Figure 1: Schematic showing the autoencoder in use as well as both methodologies. There are two terms in the loss function for the [Weak formulation](#). On the other hand, there's an additional step before the decoder for the [Strong formulation](#).

1. The Weak formulation: This formulation is based on (6). After choosing the number of nodes  $k$ , we generate two trainable matrices of sizes  $A \in \mathbb{R}^{D \times k}$ , and  $W \in \mathbb{R}^{L \times k}$ . Afterward, we reduce the number of dominant singular values of the latent space by adding a term to the loss as seen in blue in figure 1. By doing so, we are implicitly asking the Neural Network to find the values of constants  $\alpha_j^d$  and vectors  $W_j$  in (5), and hence imposing the latent space to be represented by the reduced basis of the chosen size (i.e. low-rank latent space). We can say that the method finds PGD-like vectors that span the space, since we ask the Neural Network to find a basis, without enforcing the orthogonality of the vectors. Accordingly, the formulation gains some of the benefits and limitations of a PGD. On the one hand, we are finding a basis, so generalization over the test should be better since the basis should span the space of the parameters already seen in training. On the other hand, a larger number of modes could be required to represent the entire solution.

We will refer to this method as the Weak formulation since at the end of the training, the second term of the loss could still be large, meaning we might need more modes ( $k^* > k$ ) to represent the latent space  $Y$  correctly.

Note for training: Since in (5), the constants  $\alpha$  can be anything, but the vectors  $W_j$  are normalized, we normalize each column of matrix  $W$  after every gradient descent. In addition, our initial guess of  $A$  and  $W$  is normalized, so we use a bigger learning rate for those matrices compared to the rest of the Neural Network. The main reason is to allow the Neural Network to change significantly the values of the coefficients  $\alpha_j^d$  even if the rest of the Neural Network doesn't need/can't sustain larger learning rates (more details in Section [AAA](#)).

2. The Strong formulation: Unlike the weak formulation, this architecture enforces, in a strong manner, the dimension of the reduced basis of the latent space. Similarly to the first formulation, we choose the number of modes  $k$ . Then, as seen in red in figure 1, a truncated SVD (up to the  $k$ th singular value) of the latent space is given to the decoder, instead of the latent space itself. Accordingly, the input of the decoder will have exactly  $k$  dominant singular values. In contrast to weak formulation, we can say that the strong formulation computes a POD-like basis since the vectors are by construction orthogonal. The orthogonality of the basis vectors, as well as limiting the loss

function to one term should both make training and interpolating easier. On the other hand, backpropagation through the singular value decomposition is not common in practice, so it could lead to unexpected behavior in some frameworks where it is not implemented correctly.

In both formulations, the main idea is to find a reduced basis of the latent space and enforce that the basis has a low dimension  $k$ . In the remainder of the paper, we show that both formulations lead to much better interpolation between curves with one/two parameters. We also show how RRAEs can reduce noise and computational overhead when combined with a POD, and they can perform training even if  $k$  isn't chosen to be the original dimension of the parametric space.

### 3 Interpolation in the latent space

Since both formulations previously proposed find a reduced basis of a small dimension for the latent space, we propose two ways to perform interpolation in the latent space. Our work in this paper will be limited to interpolation but future research will include other tasks such as data generation and downstream classification tasks.

What motivated us to propose this application is the limitations of linear interpolation between solutions when the solution matrix is high-rank. Take, for instance, sinusoidal curves shifted by a scalar  $p$  (i.e.  $\sin(x + p)$ ). As can be seen in figure 2, interpolating linearly between the curves corresponding to parameters  $p_0 = 0$  and  $p_1 = \pi$  to find the middle curve at  $p^* = \pi/2$  (i.e. simply the sum divided by two) leads to the horizontal line at zero, instead of finding the correct curve shifted towards the middle. By sending the data into a longer latent space with a low rank, we believe we will be able to find a space where we can interpolate linearly before

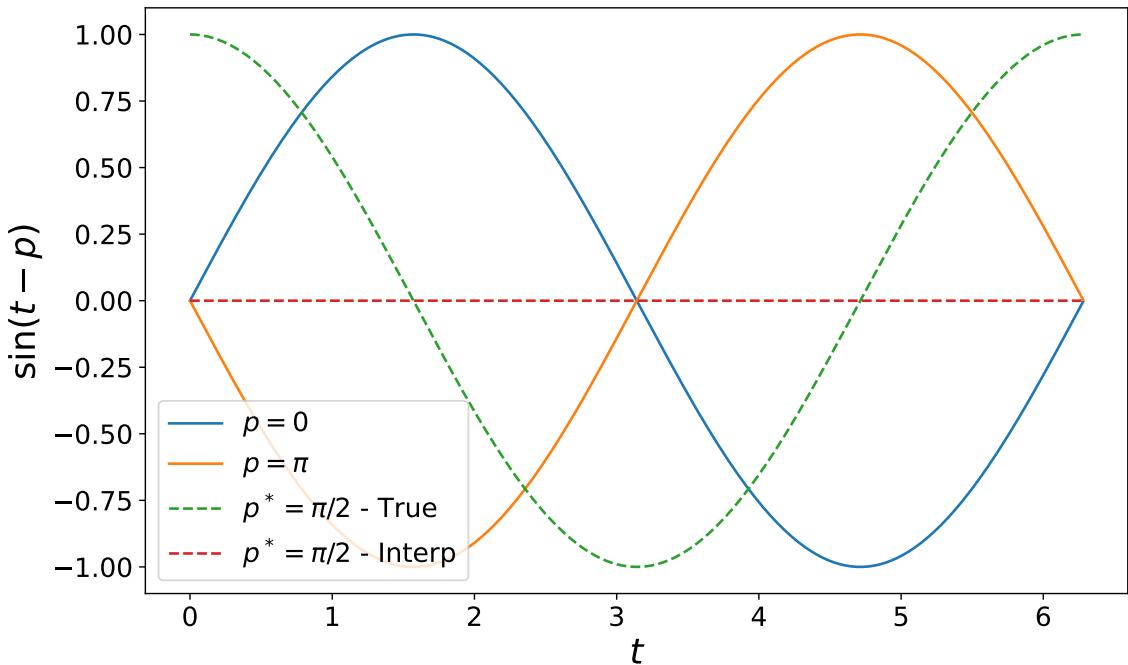


Figure 2: Figure showing the result when interpolating linearly between two shifted sin curves, showing why linear interpolation is not a good idea for problems with multiple dominant singular values.

sending the data back to the original space with the decoder.

Both proposed formulations allow us to approximate the latent space by (5) or (6). Now say, every series of observations  $X_i$  is tied to a vector of parameters  $\mathbf{p}_i \in \mathbb{R}^P$ . Accordingly, and since each column  $W_j$  is the same for every column of  $Y$ , we can only interpolate between the coefficients  $\alpha_j^d$ , as we can say,

$$Y_d = \sum_{j=1}^k \alpha_j^d W_j, \quad \Rightarrow \quad Y(\mathbf{p}_i) = \sum_{j=1}^k \gamma_j(\mathbf{p}_i) w, \quad (7)$$

where each  $\gamma_j : \mathbb{R}^P \rightarrow \mathbb{R}$ , could be any mapping, that maps all the training parameters to the corresponding  $\alpha_j^d$ , and allows us to interpolate when used on new values of the parameter  $p$ .

Note: In other words, we interpolate between the coefficients in matrix  $A$ !

Throughout the paper, we show that for every example presented a linear/bilinear interpolation is enough. For instance, for a parameter space of dimension one, and each  $j$ , we can sort the values of  $\alpha_j^d$  based on their corresponding parameter values  $p_i$  and write,

$$\gamma_j(\mathbf{p}^*) = \gamma_j(p^*) = \alpha_m + \frac{\alpha_{m+1} - \alpha_m}{p_{m+1} - p_m}(p^* - p_m). \quad (8)$$

where we distinguish between the **bold** notation for vectors and non-bold for scalars,  $m$  is the index of the largest value of  $p$  that is smaller than  $p^*$ , and both  $p_0 = \alpha_0 = 0$  for the equation to be valid for any  $i \in [1, d]$ .

Similarly, bilinear interpolation is performed between the four closest values of  $\alpha_j^d$  from the top right, top left, bottom right, and bottom left respectively (for details, refer to Appendix AAA).

Finally, for higher dimensional cases, more elaborated techniques could be used for interpolation (e.g. the sPGD), or even a Neural Network. Yet, the drawback of using a Neural Network is that we would need a Neural Network for every mapping  $\gamma_j$ . In other words, as many Neural Networks as modes! Even though we show an example later in the paper where a Neural Network works very well, interpolating linearly between the coefficients (as shown in multiple examples in what follows) would work well in practice since the RRAE already did the linearisation job.

## 4 Insights behind RRAEs

Autoencoders were originally introduced with smaller latent dimensions for feature recognition and data reduction. Overall, Vanilla autoencoders try to find only a few features that represent the data before giving these back to the decoder. The papers that presented the IRMAE [41], and LoRAE [42] provided many examples showing that longer low-rank latent spaces lead to better results in interpolation, data generation, and downstream classification. In this section, we explain by two examples why and when longer low-rank latent spaces can overcome Vanilla autoencoders. The idea is to show why and when a long latent space of rank  $k$  leads to better results than simply a latent space of dimension  $k$ .

1- Longer latent spaces lead to more flexibility in encoding: As has been shown in many applications (e.g. Koopman Theory), a larger dimension leads to more linear

behavior. Accordingly, by enlarging the latent space and asking the Neural Network to only use a reduced version of the matrix (i.e. both our Weak and Strong formulations), we allow the encoder to find more linear embeddings in the latent space, to then choose a few of them. Accordingly, for the same training parameters, while a regular autoencoder might find highly nonlinear coefficients to fit the training data, RRAEs find more “linear” coefficients. Even though both could lead to great results on the training data, when interpolating (especially linearly in the latent space), the Vanilla autoencoder wouldn’t have the right interpolations over the test set. Note that we don’t claim to have done the impossible. Any Neural Network can overfit the data without learning the actual dynamics.

As we show later in this section for curves parameterized with one parameter, Vanilla Autoencoders can be as efficient as RRAEs if the curves are highly nonlinear but easily separable. However, as can be seen in Section 5, RRAEs can interpolate between harder one-parameter curves and other two-parameter ones where the Vanilla AE fails. This is expected since both our Weak and Strong formulations have the advantage of building a basis in the latent space instead of finding random parameters. Once we have more than one parameter (i.e. more than one basis vector in the latent space), the strong formulation is finding an orthogonal basis (similar to a POD), while the weak formulation is finding a non-orthogonal one (similar to a PGD). Accordingly, if the test data have the same dynamics as the training ones, they should be representable by the basis already created in the latent space. The Vanilla Autoencoder on the other hand finds any values that help the decoder retrieve the solution. Accordingly, interpolating linearly, especially with multiple parameters, can lead to terrible interpolation results.

2- Longer latent spaces are easier to decode: A long latent matrix of rank  $k$  includes exactly as much information as a matrix of length  $k$  with full rank. In other words, the decoder does not receive any more information when longer, but low-rank latent spaces are used. However, decoding with much longer latent spaces is easier since duplicated data is given to the network. For instance when  $k = 1$ , a Vanilla autoencoder’s latent space will contain only a constant per solution (say  $c_i$ ). However, a longer latent space would include linear combinations as  $\alpha_l c_i$  where  $l \in [1, L]$ . So the bigger the latent space, the more duplicated data is given to the decoder which can help it in untangling the relationships between parameters. Since the amount of data given is the same, we believe that a powerful decoder (e.g. many layers) should be able to reconstruct the data in a regular autoencoder anyway. In other words, smaller decoders can be used with RRAEs, and in complicated examples (e.g. pictures with multiple parameters, or curves that intersect multiple times), this could lead to better results over the training data.

To explain further the arguments presented above, we test Vanilla Autoencoders and both our formulations on two examples characterized by one parameter. The first example we propose is shifted sin curves, since curves have a simple nonlinearity they are hard to separate (nonmonotonic and cross each other multiple times). The second example we propose is stair-like curves. In this example, we create highly nonlinear curves (different supports, different numbers of jumps, etc.) that are easier to separate (since they are monotonic and rarely cross each other). The equations

used to define the columns of our input matrix  $X$  in each case are,

$$\begin{cases} X_d(t_v, p_d) = f_{shift}(t_v, p_d) = \sin(t_v - p_d\pi), & p_d \in [0, 1.5], \quad t_v \in \mathbb{R}^T, \\ X_d(t_v, p_d) = f_{stair}(t_v, p_d, \text{args}) \end{cases} \quad (9)$$

where  $t_v \in \mathbb{R}^T$  is a vector of time at which observations are done, and  $f_{stair}$  is defined by the following algorithm,

---

**Algorithm 1:** Algorithm to find  $f_{stair}$  for a list of parameters  $\mathbf{p}$ .

---

**Input:**  $\mathbf{p} \in \mathbb{R}^P$ ,  $t_v \in \mathbb{R}^T$ ,  $(\text{Ph}_0, \text{Amp}_0, \kappa, y_0, w) \in \mathbb{R}$   
**for**  $p$  **in**  $\mathbf{p}$  **do**  
   $\text{Amp} = p$   
   $\text{Ph} = \text{Ph}_0 + \kappa(\text{Amp} - \text{Amp}_0)$   
   $g(t) = \text{Amp}\sqrt{t} \sin(w(t - \text{Ph})) - y_0$   
   $h(t) = \left( \frac{|g(t)| + g(t)}{2} \right)^5$   
   $f_p(t_v) = \text{cumsum}(g(t_v))$   
**end**  
**Output:**  $f_p(t_v)$  for every parameter  $p$ .

---

In this paper, we choose the initial parameters of the stair function to be,

$$\begin{cases} \text{Ph}_0 = 0.875, & \text{Amp}_0 = 1 \\ \kappa = 2.286, & y_0 = 2.3, \quad w = 2\pi. \end{cases} \quad (10)$$

Training is performed over 14 values of  $p_d$  that are equidistant. We then test on 20 values randomly chosen inside the training domain. The large number of tests is to guarantee that the models are learning the dynamics and not just the training curves and some tests nearby. Since the curves depend on one parameter, we use a Vanilla Autoencoder with a scalar latent space, and an RRAE with a latent space of length 240 and rank one for both the weak and strong formulations, we fix every other training parameter for the comparison to be fair. The details of the hyperparameters and architecture choices can be found in Appendix AAA. The relative error over all  $p$  values for both the train and test sets is documented in the following table,

	Error on Train (in %)	Error on Test (in %)
Vanilla AE	2.46	31.26
RRAE (Strong)	1.35	2.4
RRAE (Weak)	3.71	7.3

Table 1: Table showing the relative error (in %) for all three architectures on both the train and test set for shifted sin curves.

It can be seen from the high relative error of the Vanialla Autoencoder that it is overfitting the data, even though the same training hyperparameters are used for all three architectures. In addition, and as will be seen in multiple examples in the paper later on, the strong formulation has the lowest error on the train set. This

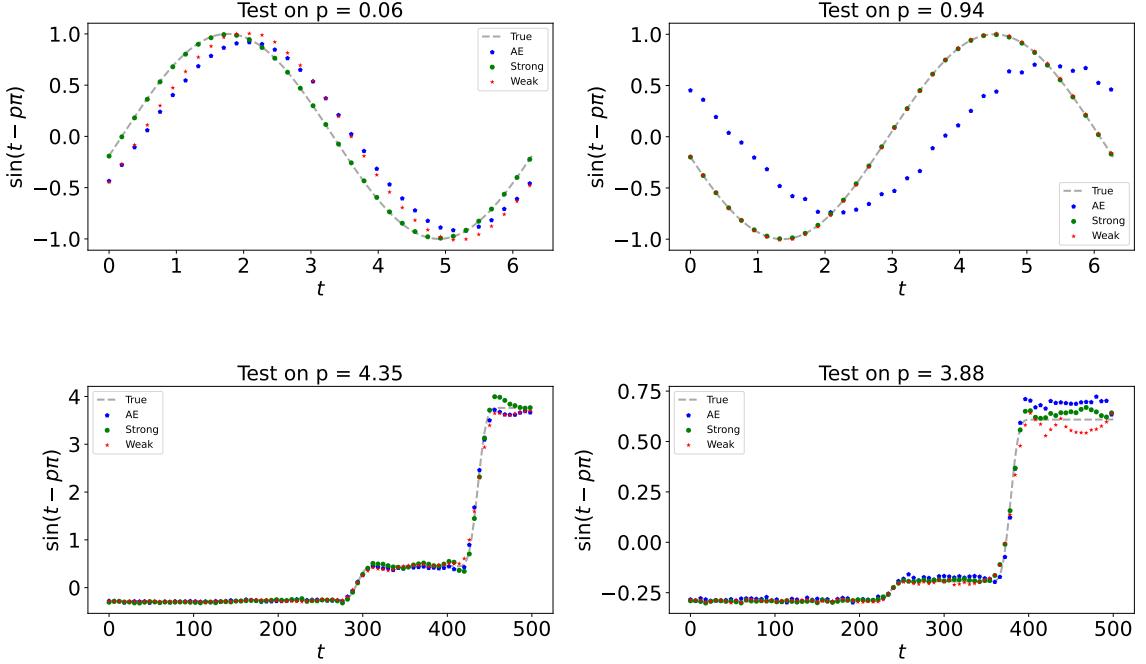


Figure 3: Figure showing the predictions of Vanilla Autoencoders and RRAEs with both formulations of four particular values of  $p$  in the test set.

shows that even if an encoder/decoder of the same size was used for the RRAE and a Vanilla AE, it is easier for the decoder in the RRAE to find better results because of the numerous duplicates in the latent space. To further understand the significance of the results, we depict the predictions of all three architectures in figure 3. We chose on purpose two values of  $p$  next to the boundaries of the parametric domain (i.e.  $p = 0.06$  and  $p = 1.31$ ), as well as two other values randomly picked. As can be seen in the figure, The Vanilla Autoencoder’s predictions (in blue) are close to the true curves (in grey) for some values of  $p$  (to the left), but are completely different for other  $p$ -values to the right. On the other hand, the Strong formulation’s predictions (in green), fit closely the curves for all the values of  $p$ . The weak formulation though is doing as well as the strong formulation over all the values of  $p$  except for the smallest one where it is as bad as the regular Autoencoder. According to the claims we established before, it is expected that the RRAE can find more “linear” coefficients that facilitate interpolation, but is this what’s happening here?

To validate our hypothesis, we draw the interpolated coefficients in figure 4. It is important to note that the coefficients are defined differently between the RRAE and the Vanilla AE. On the one hand, we remind the reader that the latent space for RRAEs can be written as in (6). However, since we only use one mode in this example,  $A \in \mathbb{R}^D$  is simply a vector of coefficients (one for each curve, between which we interpolate). Figure 4 shows the values inside vector  $A$  plotted against the corresponding parameter for every curve for both the weak and strong formulations. On the other hand, since the latent space of the Vanilla autoencoder is of length one, its latent space is only a constant per curve (again, between which we interpolate) and hence we draw the latent space against the corresponding parameters for the Vanilla AE.

The coefficients show why both the weak and the strong formulations have better in-

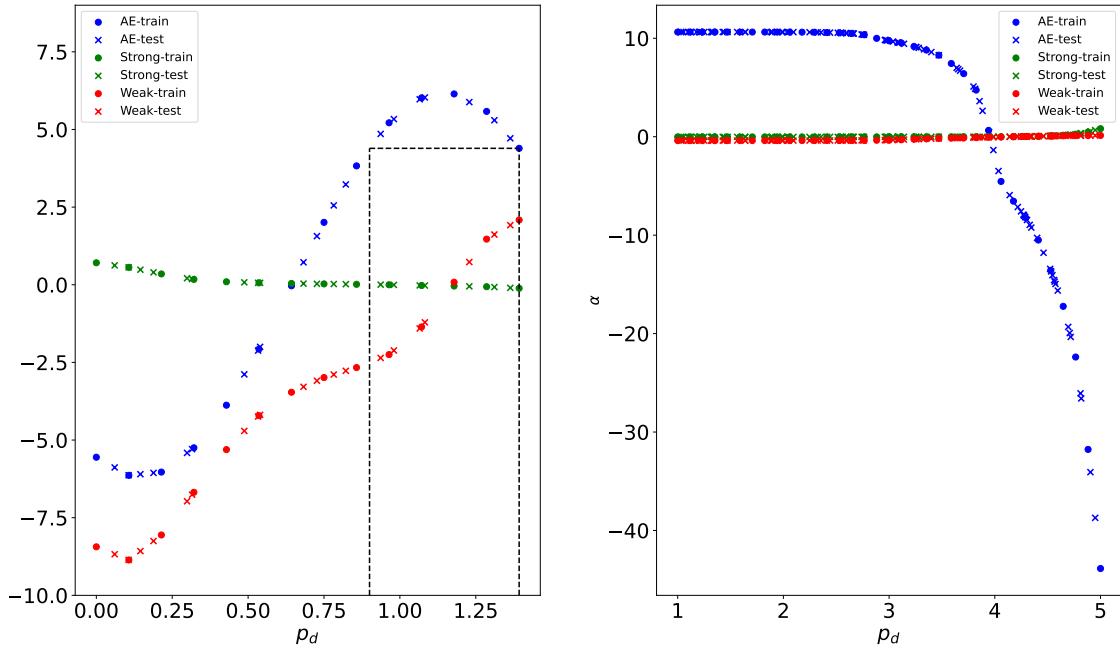


Figure 4: Figure showing the coefficients to be interpolated (dots) for all three architectures, and the interpolated values for the test set (crosses).

terpolation capabilities. The main problem with the coefficients found by the Vanilla AE (the blue crosses and dots) is that the resulting curve from linear interpolation is not an injection, for most of the domain! Specifically, looking between the black dashed line, any value of  $p$  (say  $p_1 = 1.1$ ) will have the same coefficient as another  $p$  nearby ( $p_2$  around 1.25). Accordingly, the decoder will find the same curve for two different parameters, which is wrong since  $p$  defines a shift. The same thing can be said about another interval from  $p = 0$  to around  $p = 0.3$ . This reasoning explains why the Vanilla AE isn't capable of interpolating between the curves, but why are both the weak/strong formulations doing better?

Based on the first argument proposed earlier in this section, a longer latent space allows RRAEs to find more “linear” coefficients. This is clearly shown by the coefficients of the strong method (in green) which have a linear monotonic behavior. On the other hand, the weak formulation finds monotonic coefficients in the range  $p \in [0.2, 1.4]$  but has the same problem as the Vanilla AE when  $p \in [0, 0.2]$ . Accordingly, the RRAE with the weak formulation can interpolate on most of the domain except a small part in the beginning. This explains why the weak formulation is doing as bad as the Vanilla AE on  $p = 0.06$ , and why its relative error over all the test curves is much lower than Vanilla Autoencoder.

Overall, we showed in a simple example how the decoder can be more powerful in RRAEs, and both formulations can find more interpolable coefficients. However, this might not always be true in practice! On simple curves, for instance, the Vanilla Autoencoder might match or even have better prediction abilities than any Autoencoder with a longer latent space. However, as we show in all of the examples in the paper, and as already shown for pictures with multiple parameters in [41] and [42], it is easier to perform learning and interpolation with longer low-rank latent spaces compared to Vanilla Autoencoders.

## 5 Testing on Numerical Data

In this section, we use RRAEs to interpolate between multiple examples with high-rank solution matrices. Our results show that RRAEs with both formulations can interpolate between curves with one/two parameters even if a linear/bilinear interpolation is used in the latent space. We also show how using a POD before the RRAE can filter noisy data and limit the computational overhead for long time series (i.e. when  $T$  in figure 1 is large).

### 5.1 Examples with one/two parameters

### 5.2 Extensions of RRAES

This section is empty since we're waiting for the data...

## 6 Summary and Conclusions

In this article, we presented Rank Reduction autoencoders (RRAE), autoencoders that can reduce any set of solutions (even if they are linearly independent) into a space with only one (or a few) dominant singular values. We proposed two formulations, a Weak, and a Strong one, to impose the dominance of one singular value in the latent space. The first consisted of adding a term with two trainable vectors to the loss, while the other allowed the latent matrix to be anything before taking its first-order truncated SVD as the new latent matrix.

We shared some insights into why one dominant singular value in the latent space should always be feasible. Afterward, we explained how the choice of the parameters of the autoencoder was done. This was followed by both formulations being tested to perform interpolation between curves that are not aligned.

We first tested the model on equation-generated examples. We proposed two simple functions and a more complicated one to show the robustness of the model and the methods. The results we shared showed small errors (< 2%), on both the train and the test sets, along with latent spaces with one dominant singular value.

Afterwards, we tested the model on a set of real-world data, to be continued...

Our results and discussion showed that even though both formulations work, the Strong formulation appears to be more robust and persistent. We decided to share the Weak formulation as well since it could either be useful for extensions later on or be the better choice for readers who prefer not to backpropagate through singular values and singular vectors.

The purpose of this article is to introduce the methodology. Our goal is to propose the idea and show that it works for tasks in the latent space. We hope that the extension of our work for multiple parameters (e.g. by using a mapping  $\alpha$  as previously mentioned) and multiple singular values (e.g. recurring autoencoders with one singular value each) will follow soon. We hope that this will motivate the use of Rank Reduction Autoencoders for many other applications that require a space with only a few dominant features, whether they're low-rank techniques, Model Order Reduction techniques, or others.

All in all, our results claim that Rank Reduction autoencoders are effective and easy

to train. For any readers who would like to test the autoencoders for themselves, our code was done in `Jax` and can be found on our [GitHub repository](#). For any questions and concerns, feel free to reach us there as well.

## References

- [1] Olek C Zienkiewicz and Robert Leroy Taylor. *The finite element method for solid and structural mechanics*. Elsevier, Amsterdam, 2005.
- [2] Gordon D Smith, Gordon D Smith, and Gordon Dennis Smith Smith. *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, Oxford, 1985.
- [3] Clarence W Rowley, Tim Colonius, and Richard M Murray. Model reduction for compressible flows using pod and galerkin projection. *Physica D: Nonlinear Phenomena*, 189(1-2):115–129, 2004.
- [4] Gaetan Kerschen, Jean-claude Golinval, Alexander F Vakakis, and Lawrence A Bergman. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: an overview. *Nonlinear dynamics*, 41:147–169, 2005.
- [5] Pierre Ladevèze. Sur une famille d’algorithmes en mécanique des structures. *Comptes-rendus des séances de l’Académie des sciences. Série 2, Mécanique-physique, chimie, sciences de l’univers, sciences de la terre*, 300(2):41–44, 1985.
- [6] S Rodriguez, David Néron, P-E Charbonnel, Pierre Ladevèze, and G Nahas. Non incremental latin-pgd solver for non-linear vibratory dynamics problems. In *14ème Colloque National en Calcul des Structures, CSMA 2019*, 2019.
- [7] Francisco Chinesta and Elias Cueto. *PGD-based modeling of materials, structures and processes*. Springer, Switzerland, 2014.
- [8] Caterina Labrín and Francisco Urdinez. Principal component analysis. In *R for political data science*, pages 375–393. Chapman and Hall/CRC, 2020.
- [9] David González, José Vicente Aguado, E Cueto, E Abisset-Chavanne, and F Chinesta. kPCA-based parametric solutions within the pgd framework. *Archives of Computational Methods in Engineering*, 25:69–86, 2018.
- [10] Marco Tezzele, Nicola Demo, and Gianluigi Rozza. Shape optimization through proper orthogonal decomposition with interpolation and dynamic mode decomposition enhanced by active subspaces, 2019.
- [11] Minh-Nhan Nguyen and Hyun-Gyu Kim. An efficient podi method for real-time simulation of indenter contact problems using rbf interpolation and contact domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 388:114215, 2022.
- [12] RR Rama and S Skatulla. Towards real-time modelling of passive and active behaviour of the human heart using podi-based model reduction. *Computers & Structures*, 232:105897, 2020.
- [13] Francisco Chinesta, Pierre Ladeveze, and Elias Cueto. A short review on model order reduction based on proper generalized decomposition. *Archives of Computational Methods in Engineering*, 18(4):395–404, 2011.
- [14] Abel Sancarlos, Victor Champaney, Elias Cueto, and Francisco Chinesta. Regularized regressions for parametric models based on separated representations. *Advanced Modeling and Simulation in Engineering Sciences*, 10(1):4, 2023.

- [15] Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 720–727, 2003.
- [16] Guangcan Liu, Zhouchen Lin, Shuicheng Yan, Ju Sun, Yong Yu, and Yi Ma. Robust recovery of subspace structures by low-rank representation. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):171–184, 2012.
- [17] Madeleine Udell, Corinne Horn, Reza Zadeh, Stephen Boyd, et al. Generalized low rank models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.
- [18] Mark A Davenport and Justin Romberg. An overview of low-rank matrix recovery from incomplete observations. *IEEE Journal of Selected Topics in Signal Processing*, 10(4):608–622, 2016.
- [19] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.
- [20] Udo Seiffert. Multiple layer perceptron training using genetic algorithms. In *ESANN*, pages 159–164. Citeseer, 2001.
- [21] Sushmita Mitra and Sankar K Pal. Fuzzy multi-layer perceptron, inferencing and rule generation. *IEEE Transactions on Neural Networks*, 6(1):51–63, 1995.
- [22] Jie Xiong Tang, Chenwei Deng, and Guang-Bin Huang. Extreme learning machine for multilayer perceptron. *IEEE transactions on neural networks and learning systems*, 27(4):809–821, 2015.
- [23] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.
- [24] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaei. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.
- [25] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.
- [26] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.
- [27] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Liyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.
- [28] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33(12):6999–7019, 2021.
- [29] Leonard Berrada, Andrew Zisserman, and M Pawan Kumar. Training neural networks for and by interpolation. In *International conference on machine learning*, pages 799–809. PMLR, 2020.

- [30] Juan P Rigol, Claire H Jarvis, and Neil Stuart. Artificial neural networks as a tool for spatial interpolation. *International Journal of Geographical Information Science*, 15(4):323–343, 2001.
- [31] Geoffrey E Hinton and Richard Zemel. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems*, 6, 1993.
- [32] Bhavik Vachhani, Chitralekha Bhat, Biswajit Das, and Sunil Kumar Koppaprapu. Deep autoencoder based speech features for improved dysarthric speech recognition. In *Interspeech*, pages 1854–1858, 2017.
- [33] Xue Feng, Yaodong Zhang, and James Glass. Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 1759–1763. IEEE, 2014.
- [34] Yihan Deng, André Sander, Lukas Faulstich, and Kerstin Denecke. Towards automatic encoding of medical procedures using convolutional neural networks and autoencoders. *Artificial intelligence in medicine*, 93:29–42, 2019.
- [35] Hoo-Chang Shin, Matthew R Orton, David J Collins, Simon J Doran, and Martin O Leach. Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4d patient data. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1930–1943, 2012.
- [36] Daehyung Park, Yuuna Hoshi, and Charles C Kemp. A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *IEEE Robotics and Automation Letters*, 3(3):1544–1551, 2018.
- [37] James Sergeant, Niko Sünderhauf, Michael Milford, and Ben Upcroft. Multi-modal deep autoencoders for control of a mobile robot. In *Proc. of Australasian Conf. for robotics and automation (ACRA)*, 2015.
- [38] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.
- [39] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.
- [40] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.
- [41] Li Jing, Jure Zbontar, et al. Implicit rank-minimizing autoencoder. *Advances in Neural Information Processing Systems*, 33:14736–14746, 2020.
- [42] Alokendu Mazumder, Tirthajit Baruah, Bhartendu Kumar, Rishab Sharma, Vishwajeet Pattanaik, and Punit Rathore. Learning low-rank latent spaces with simple deterministic autoencoder: Theoretical and empirical insights, 2023.
- [43] Gilbert W Stewart. On the early history of the singular value decomposition. *SIAM review*, 35(4):551–566, 1993.
- [44] Yuji Nakatsukasa. Accuracy of singular vectors obtained by projection-based svd methods. *BIT Numerical Mathematics*, 57(4):1137–1152, 2017.

## Appendix

Before introducing the testing examples, we begin by explaining our logic behind the choice of the autoencoder architecture, which we believe should be considered a logical choice for almost any input. These include the following:

1. The encoder: As previously mentioned, the encoder is chosen to be an MLP. We believe that the number of layers and Neurons for the encoder should be small. The main reason is that the decoder has to find the inverse function, which becomes much more complicated the larger the encoding network is. For this article, we choose an encoder of depth 1 (one hidden layer), and width 20 (20 Neurons).
2. The decoder: Since the decoder has to find inverse maps, we use deeper MLPs. For both proposed formulations, we find that a decoder with either 4 hidden layers and 64 Neurons for each layer is usually enough to be able to decode the latent space.
3. The latent space dimension  $L$ : Autoencoders give us the choice of the dimension  $L$ , which could be either bigger or smaller than the original dimension  $T$ . However, the shorter the latent space, the harder it is for the decoder to find the inverse map. On the other hand, the longer the latent space, the more the decoder's prediction is affected by the errors in the latent space. For instance, an error of 1% over 5 values has less effect on the decoder results than an error of 1% over 50 values. Since we chose the decoder to be a deeper Neural Network, we decided to reduce the dimension space, hoping that the decoder would be able to find the prediction even while using only a few values. For this article, we find  $L$  by multiplying the time dimension  $T$  by 0.2 and rounding to the nearest integer.
4. The loss weights: We chose all the weights in the loss to be equal to one. We believe that with the good choice of the architecture presented above, changing the constants won't be necessary.
5. Other training parameters: To show the flexibility and rigidity of our model, we choose almost fixed training parameters. We don't try to fine-tune these to get better results. The purpose of doing so is to show how powerful the RRAE is, and that with all the carefully selected parameters above, it can achieve small errors even for complicated examples. We perform 4 training loops, each of 2900 epochs, and a learning rate that's reduced from 1e-3 to 1e-6 by dividing by ten. Even though the number of epochs appears to be large, most epochs are not reached because we enforce a strong stagnation condition that stops the loops when the error stops decreasing. In addition, we used batches with sizes that range from 4 to 16, depending on the total number of curves and the formulation. In general, a larger batch size was required by the Weak formulation to converge.