Rank Reduction Autoencoders - Enhancing interpolation on nonlinear manifolds.

Jad Mounayer[1,2,*], Sebastian Rodriguez[1,3], Chady Ghnatios[1,2], Charbel Farhat[4], Francisco Chinesta[1,5]

[1]PIMM, ENSAM Institute of Technology, 151 Boulevard de l'Hôpital, 75013, Paris, France.
[2] PIMM Lab, SKF Research Chair, Arts et Metiers Institute of Technology, Paris, France
[3]ESI Group Chair, PIMM, ENSAM Institute of Technology, 151 Boulevard de l'Hôpital, 75013, Paris, France.
[4] 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.
[5]RTE Research Chair, PIMM, ENSAM Institute of Technology, 151 Boulevard de l'Hôpital, 75013, Paris, France.

April 14, 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 to interpolate between the coefficients of the basis in the latent space instead of the curves. We show that both formulations can interpolate between curves with one/two-dimensional parametric spaces even when a linear/bilinear interpolation is used. Finally, we showcase the effectiveness of RRAEs by using them to interpolate between physical solutions generated by the Avrami crystallization model. Our results show that RRAEs preceded by a POD can filter noisy data and have a reduced computational overhead and that ignorance of the dimension of the parametric space does not affect the interpolation capabilities of the model.

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 efficiency of the methods presented above, as well as every low-rank technique (e.g. [15, 16, 17, 18]), depends on our 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 fail to create an efficient surrogate for the correct prediction of physical phenomena. Other interpolation techniques based on different formulations such as the Optimal Transport (OT) [19, 20] are also unable to interpolate properly with high-rank matrices.

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 [21, 22, 23, 24], Recurrent Neural Networks [25, 26, 27], and Convolutional Neural Networks [28, 29, 30].

The nonlinearity in Neural Networks encouraged researchers to use them for interpolation [31, 32]. 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 [33]. 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 [34, 35], medical applications [36, 37], robotics [38, 39], and others [40]. 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 [41]. On the other hand, a better understanding of the data can be established when the latent space is smaller [42]. Even though Autoencoders appeared to be very promising, their efficiency is limited, especially for data reduction/interpolation, since short latent spaces make it hard for the decoder to retrieve the original values, and longer ones do not reduce the dimension (more details in Section 4). Accordingly, multiple enhancements, such as Sparse [43], Denoising [44, 45], and Variational autoencoders [46, 47], have been proposed to alleviate as enhancements for Vanilla AEs but still don't resolve the reduction problem.

Recently, the Implicit Rank-Minimizing Autoencoder (IRMAE) [48], and the Low-Rank Autoencoder (LoRAE) [49] showcased how increasing the latent space dimension while enforcing a low-rank achieves better results, including interpolation. Their models consist of adding one or more linear layers for the encoder to allow the Neural Network to find a latent space of a lower rank. Their results show that a long latent space with a low rank enhances Autoencoders. However, we believe that long latent spaces with low-rank can be exploited further.

In the following article, we present Rank Reduction Autoencoders (RRAE), which strongly enforce a long latent space of low rank by finding a reduced basis for it. Accordingly, if the training set is big enough, the basis we find can be used to interpolate since it spans the parametric space. To find the basis coefficients and enforce a low-rank latent space, we propose two formulations: a strong and a weak one. Throughout the paper, we show that the strong formulation finds orthogonal basis vectors and hence forms a POD-like basis, while the weak formulation is allowed to find non-orthogonal vectors so it mimics a PGD. We show that both formulations can interpolate efficiently between curves with high-rank solution matrices, where the Vanilla Autoencoder fails.

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 two examples, 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 three examples with one/two parameters where we show that RRAEs with both formulations have much better interpolation abilities than the regular Autoencoder, and that linear/bilinear interpolation is enough even on the hardest curves presented. Then, in Subsection 5.2, we show that combined with a POD, RRAEs can filter noise and their computational overhead is highly reduced.

Additionally, we show that RRAEs can find an approximation of the dimension of the latent space by themselves. In other words, prior knowledge of the dimension of the parametric space is not required. Finally, we conclude and discuss the future aspects of the proposed architecture in Section 6. Our results show that by finding a basis in the latent space, RRAEs are much better interpolators than Vanilla AEs, they can be used to filter noise, and they are capable of finding an approximation of the intrinsic dimension of the parametric space.

2 Rank Reduction Autoencoders (RRAEs)

To define the architecture of RRAEs, we begin 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 ith column. Let, $Y \in \mathbb{R}^{L\times D}$ with L, the chosen dimension of the latent space. We also define the encoding map $e : \mathbb{R}^{T\times D} \to \mathbb{R}^{L\times D}$ and the decoding map $d : \mathbb{R}^{L\times D} \to \mathbb{R}^{T\times D}$. The Vanilla autoencoder can be written as the following two operations,

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

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

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

The idea behind RRAEs is to enforce the latent matrix to have a low rank. In other words, let $Y = U^y\Sigma^yV^y$ be the Singular Value Decomposition (SVD) [50, 51] of Y, and let $\{\sigma_i^y\}_{i\in[1,r]}$ be the sorted diagonal values of Σ^y, r being the rank of Y. We want the following,

$$\sigma_k^y \gg \sigma_j^y, \qquad \forall j \in [k+1, r], \qquad k \ll r. \tag{3}$$

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

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

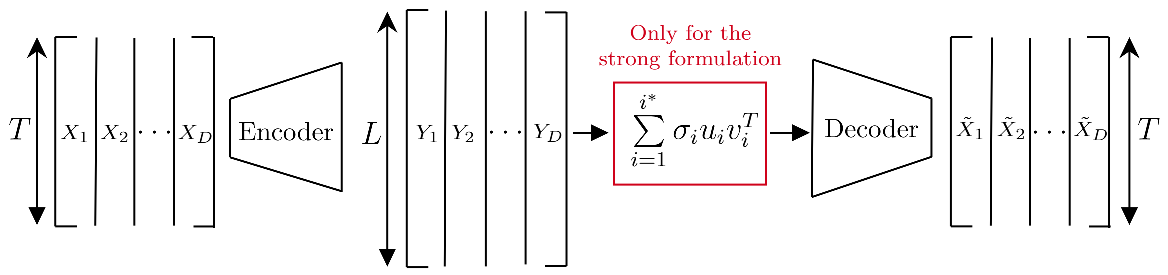
where U_i^y is the ith column of U^y and $(V^y)_i^T$ is the ith row of V^y. In other words, we can write Y_d, the dth column of Y as,

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

In other words, for a specified number of nodes k, each column of Y is defined by k constants and k vectors. 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,

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

with $(\cdot)$ denoting the dot product. Based on (5), and (6), we propose two formulations that enforce the low rank of the latent space and find its reduced basis. The architecture is sketched in figure 1.


T
X_1 X_2 $\cdots$ X_D
Encoder
L
Y_1 Y_2 $\cdots$ Y_D
Only for the strong formulation
$\sum_{i=1}^{i^*} \sigma_i u_i v_i^T$
Decoder
$\tilde{X}_1$ $\tilde{X}_2$ $\cdots$ $\tilde{X}_D$
T

Only for the weak formulation

$$\mathcal{L} = \lambda_1 \| X - \tilde{X} \|_2 \boxed{+ \lambda_2 \|Y - (A \cdot W^T)^T\|}$$

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 $A \in \mathbb{R}^{D\times k}$, 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 α_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). Accordingly, if training is performed properly, our trainable matrices are the basis vectors and the coefficients of the reduced basis of Y.

   The weak formulation 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 rank that's larger than the intrinsic parametric dimension 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.

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 rank k of the latent space. Then, as seen in red in figure 1, a truncated SVD (up to the kth 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 refraining from adding terms in the loss, should both make training and interpolating easier using this formulation.

   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. Additionally, we show in Section 5 that when the intrinsic parametric dimension is not given, the weak method has more flexibility to find the correct rank of the latent space, while the strong formulation only finds a good approximation of k.

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. Since by the end of the training, we have the basis vectors and the coefficients, we propose to interpolate directly between the coefficients as explained in the following section.

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 to interpolate between the coefficients instead of interpolating between the curves. We limit the objective of this paper to showcase the abilities of RRAEs for interpolation tasks. However, our future work will include multiple other applications such as data generation, downstream classification, and others.

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 can find a simpler space where a linear interpolation would be enough to find

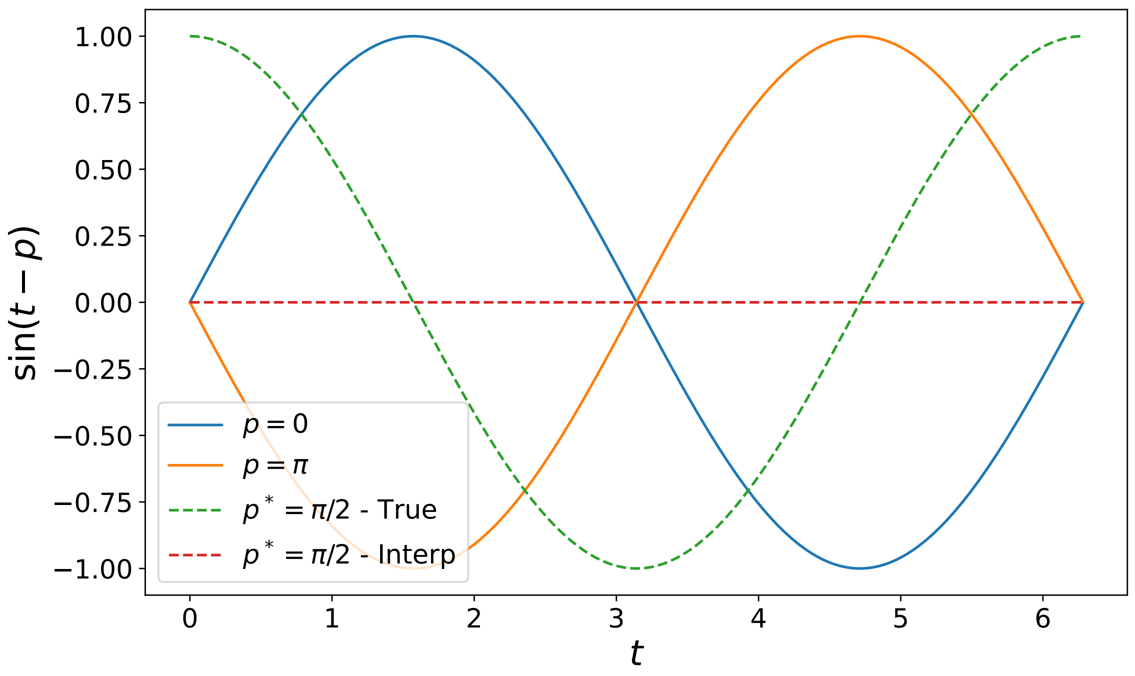


Figure 2: Figure showing the result when interpolating linearly between two shifted sin curves, showing why linear interpolation is not a good idea for problems with multiple dominant singular values.

the new curve, before going 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 α_j^d, as we can say,

$$Y_d = \sum_{j=1}^{k} \alpha_j^d W_j, \qquad \Longrightarrow \qquad Y(\mathbf{p}_i) = \sum_{j=1}^{k} \gamma_j(\mathbf{p}_i) W_j, \tag{7}$$

where each $\gamma_j : \mathbb{R}^{\mathbb{P}} \to \mathbb{R}$, could be any mapping, that maps all the training parameters to the corresponding α_j^d, and allows us to interpolate when used on new values of the parameter p. By that, we mean that interpolation is simply done between the coefficients inside matrix A in (6).

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 α_j^d based on their corresponding parameter values p_i and write,

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

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 α_j^d from the top right, top left, bottom right, and bottom left respectively (for details, refer to the Appendix).

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 γ_j. In other words, k Neural Networks to train! As we show throughout the paper, interpolating linearly/bilinearly between the coefficients is good enough since the Autoencoder already did the linearization. However, interested readers can find some of our results using k trainable Neural Networks in the Appendix as well.

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 [48], and LoRAE [49] provided many examples showing that longer low-rank latent spaces lead to better results in interpolation, data generation, and downstream classification. However, the reasons behind the success of longer low-rank latent spaces are not clarified. In this section, we provide two arguments that explain why a long latent space of rank k can lead to better training/interpolation than a latent space of length k. Specifically, we argue that finding a basis for the low-rank latent space leads to much better interpolations. We give our opinion on the validity of the arguments presented and provide two examples to support our claims.

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 tend to find more "linear" ones. Even though both could lead to great results on the training data, when interpolating (especially linearly in the latent space), Vanilla Autoencoders won't have the same generalization capabilities as RRAEs. Additionally, by finding a basis of the latent space, RRAEs should be able to generalize better over the test data, since the basis should span the space of the parameters already seen in training, whether with a POD-like basis (the strong formulation), or a PGD-like one (the weak formulation). The Vanilla Autoencoder on the other hand has holes in its interpolation [48], since it does not find a basis, but only coefficients that are helpful for the decoder to retrieve the solution.

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 information given is the same, we believe that a powerful decoder (e.g. many layers) should be able to reconstruct the data even in a Vanilla autoencoder. 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), RRAEs can have better results, even over the training data.

To explore further the arguments presented above, we test Vanilla Autoencoders and both our formulations on two examples characterized by one parameter. The first curves we propose are shifted sin curves since these have a simple nonlinearity, but they are hard to separate (nonmonotonic and cross each other multiple times). The second example we chose is curves with stair-like behavior. In this example, we create highly nonlinear curves (different supports, different numbers of jumps, etc.), but we define them to be monotonic and only cross each other occasionally (i.e. easier to separate). The equations used to define the columns of our input matrix X in each case are as follows,

$$\begin{cases} X_d(t_v, p_d) = f_{shift}(t_v, p_d) = \sin(t_v - p_d\pi), & p_d \in [0, 1.5], \\ X_d(t_v, p_d) = f_{stair}(t_v, p_d, \text{args}) & p_d \in [1, 5], \end{cases} \tag{9}$$

where $t_v \in \mathbb{R}^T$ is a vector of time at which observations are done, and f_{stair} takes some arguments "args" as detailed in the following algorithm,

```
Algorithm 1: Algorithm to find $f_{stair}$ for a list of parameters $\mathbf{p}$.
    Input: $p_d \in \mathbb{R}$, $t_v \in \mathbb{R}^T$, $(\mathrm{Ph}_0, \mathrm{Amp}_0, \kappa, y_0, w) \in \mathbb{R}$

    $\mathrm{Amp}_{p_d} = p_d$
    $\mathrm{Ph}_{p_d} = \mathrm{Ph}_0 + \kappa(\mathrm{Amp}_{p_d} - \mathrm{Amp}_0)$
    $g_{p_d}(t_v) = \mathrm{Amp}_{p_d}\sqrt{t_v}\sin(w(t_v - \mathrm{Ph}_{p_d})) - y_0$
    $h_{p_d}(t) = \left(\frac{|g_{p_d}(t)| + g_{p_d}(t)}{2}\right)^5$
    $X_d(t_v, p_d) = \mathrm{cumsum}(h_{p_d}(t_v))$

    Output: $X_d(t_v, p_d)$ for each parameter $p_d$.
```

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

$$\begin{cases} \mathrm{Ph}_0 = 0.875, \qquad \mathrm{Amp}_0 = 1 \\ \kappa = 2.286, \qquad y_0 = 2.3, \qquad w = 2\pi. \end{cases} \tag{10}$$

Training is performed over 14, and 35 equidistant values of p_d for the shifted sin curves and the stair-like curves respectively. We then test on 20 and 100 random values of p_d, respectively, 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 solution curves depend on one parameter, we use a Vanilla Autoencoder with a scalar latent space and an RRAE with a longer latent space of rank one. As detailed in Appendix A, we fix every other training parameter to ensure a fair comparison. The relative error over all p_d values for both the train and test sets is documented in the following table,

	Shifted sin		Stair-like	
	Train Error	Test Error	Train Error	Test Error
Vanilla AE	2.46	31.26	2.97	3.74
RRAE (Strong)	**1.35**	**2.4**	**1.87**	**3.2**
RRAE (Weak)	3.71	7.3	4.11	5.76

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

The results presented in the table depict the arguments presented above and some of their limitations. First of all, we can see that the Vanilla Autoencoder is overfitting the data when trained over the shifted *sin* functions. Since these are very hard to separate, and as mentioned in the first argument before, the Vanilla autoencoder is simply converging to coefficients that fit the training data, but are not generalizable. On the other hand, the RRAEs with both formulations are doing much better on the test set, with the strong formulation achieving better results than the weak one. When comparing the formulations to the POD and the PGD, we expect, that for the same rank in the latent space that is equal to the dimension of the parametric space, the PGD basis wouldn't be as efficient as the one found by the POD.

Additionally, the results on the train set show that the strong formulation is doing better than the Vanilla Autoencoder, even though the Vanilla AE is freeier to find more nonlinear coefficients. This illustrates what we proposed in the second argument, since for the same decoder size, the RRAE can fit better the training data. But is what's proposed true for every curve?

Opposingly, the results on the stair-like curves show that if the curves are highly nonlinear but easily separable (in this case, monotonic curves with little intersections between them), the Vanilla Autoencoder can fit both the train and test data well. Again, the strong formulation can fit the training data better, showing one more time that the duplicated values in the latent space help the training set. The weak formulation is as expected, doing worse than the strong one with the same rank in the latent space. However, the low error of the Vanilla AE on the test set shows that, when curves can easily be separated, the classical AE can do almost as well as the strong formulation, and even better than the weak one.

The results of these two examples explain why RRAEs can be much better than Vanilla Autoencoders for interpolation tasks. These also show that on very simple curves with one parameter, the effect of longer latent spaces is reduced. As clearly shown in all the other examples in Section 5, and as observed for pictures with multiple parameters in [48] and [49], choosing a longer latent space with a low rank provides much better results on larger parametric spaces and more complicated curves.

To further understand the significance of the results, we depict the predictions of all three architectures over both examples in figure 3. We chose on purpose two values of p_d that were the hardest to interpolate for all techniques. As can be seen in the figure, The Vanilla Autoencoder's predictions (in blue) are much worse than RRAEs with both formulations for the shifted sin curve with $p_d = 0.94$. On the other hand, it is interesting that the weak formulation is doing as bad as the Vanilla AE for a very small value of p_d (top left of the figure). A closer look at the interpolation error

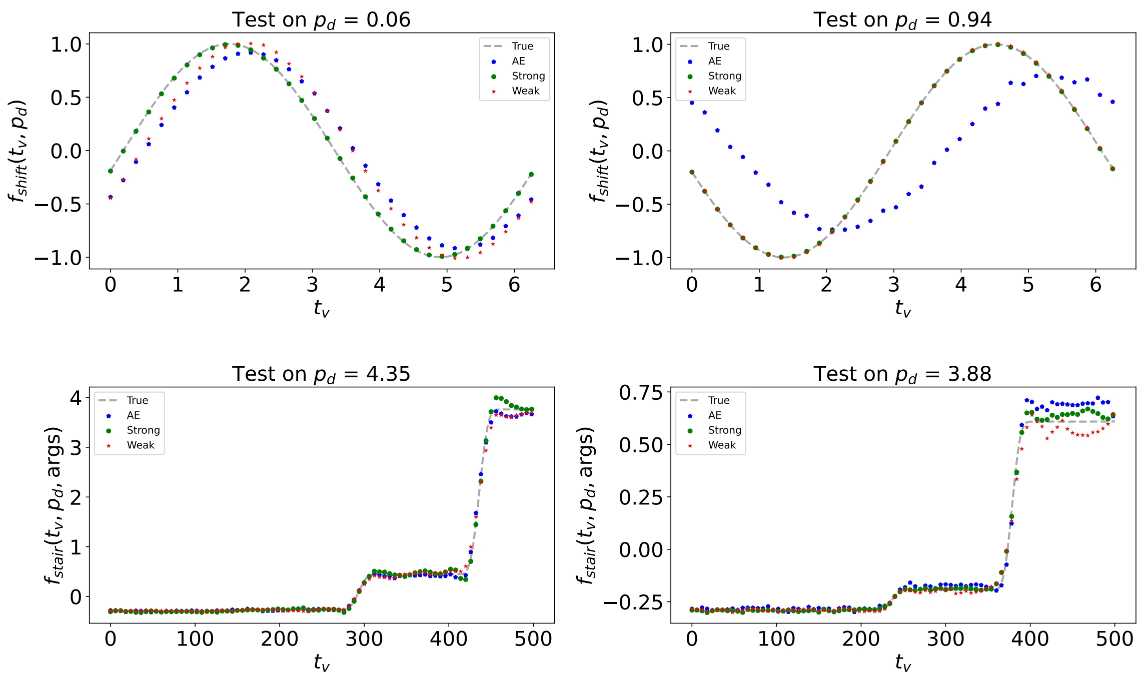


Figure 3: Figure showing the predictions of Vanilla Autoencoders and RRAEs with both formulations over two particular values of p_d for the shifted sin and stair-like examples.

of the weak formulation shows that it interpolates as well as the strong formulation over every value of p_d except for very small ones. On the stair-like function though, all three formulations have almost the same interpolation capabilities over most tests.

To explain this behavior and strengthen the arguments we provided at the beginning of the section, we plot the latent space 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 enforce a rank of 1 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 parameters 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 interpolation capabilities. The main problem with the coefficients found by the Vanilla AE for the shifted sin curves (the blue crosses and dots to the left) is that the resulting curve from linearly interpolating the coefficients is not an injection, over most of the domain. Specifically, looking between the vertical black dashed lines, 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 sin curves, but why are both the weak/strong formulations doing better?

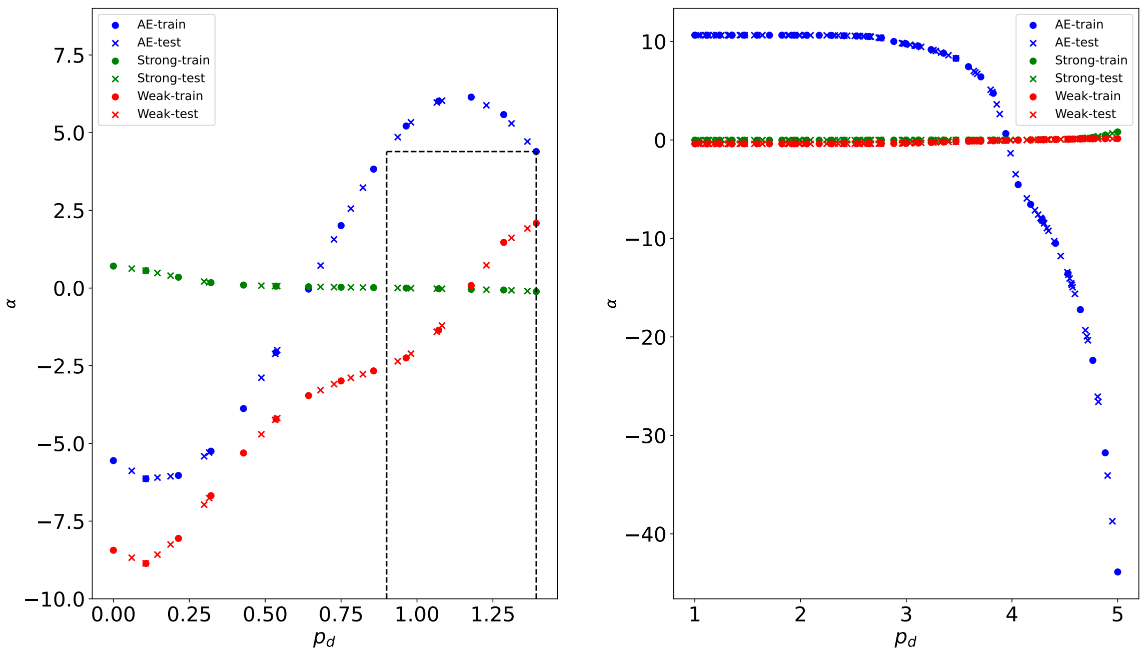


Figure 4: Figure showing the coefficients to be interpolated (dots) for all three architectures, and the interpolated values for the test set (crosses).

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, to the left) 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.

Finally, when looking at the coefficients for the stair-like problem (to the right of figure 4), we can see that since the curves are easily separable, the Vanilla AE can find monotonic coefficients, which explains why it can interpolate well in this example. It is important to note here that we talk about "monotonic" coefficients since we only have one parameter. Over parametric domains of higher dimensions, RRAEs are building a basis and will hence have a much bigger advantage over Vanilla AEs, as shown in many examples in section 5.

Overall, our results over those two simple examples show how duplicated data benefits RRAEs, and hence why we can use smaller decoders to achieve the same result (argument 2). In addition, we showed how increasing the dimension of the latent space helps the Autoencoder to find more linear coefficients, leading to better interpolation results (argument 1). Both our examples were with one parameter, where the advantage of building a basis is limited (since the basis in this case is just a vector). As we show in the following sections, over curves with parametric spaces of dimension two for instance, creating a basis for long low-rank latent spaces leads to much better interpolations than Vanilla Autoencoders. This is expected since if the training set is big enough, the basis found by the RRAEs will span the whole parametric space.

5 Testing on Numerical Data

In this section, we use RRAEs to interpolate between curves 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). Finally, we illustrate how RRAEs can find the dimension of the parametric space (or a close approximation) by themselves if a random larger rank of the latent space is chosen.

5.1 Examples with one/two parameters

In this subsection, we test the robustness of our proposed formulations over three problems. First, we choose accelerated sin curves, which are again parametrized by one parameter, but much harder to learn than shifted curves. We then propose two examples characterized by two parameters; first, the sum of two sin curves with different frequencies, as well as two gausses in two different locations. We show how in such examples, Vanilla Autoencoders are not able to interpolate, and that both our formulations have much better results for the same training parameters (again, training details can be found in Appendix AAA). We define the columns of our input matrix $X_d(t_v,\, p_d) = f_{prob}$ for each problem as follows,

$$\begin{cases} f_{acc}(t_v,\, p_d) = \sin(p_d t_v), & p_d \in [0, 1.5], \\ f_{freqs}(t_v,\, \mathbf{p}_d) = \sin(p_d^1 t_v) + \sin(p_d^2 t_v) & p_d^1 \in [1,5], \quad p_d^2 \in [1,5], \\ f_{gauss}(t_v,\, \mathbf{p}_d) = 1.3e^{-\dfrac{(t_v - p_d^1)^2}{0.08}} + 1.3e^{-\dfrac{(t_v - p_d^2)^2}{0.08}} & p_d^1 \in [1,5], \quad p_d^2 \in [1,5], \end{cases}$$

Where again, we distinguish between the **bold** notation for vectors and non-bold ones for scalars. In both the second and third expressions, our parametric space is of dimension 2 and so $\mathbf{p}_d = [p_d^1,\, p_d^2] \in \mathbb{R}^2$. Again, we only use linear/bilinear interpolation in the latent space, since the RRAEs should have dealt with the non-linearity already. The results on two interpolated curves over each example for all three formulations can be seen in figure 5.

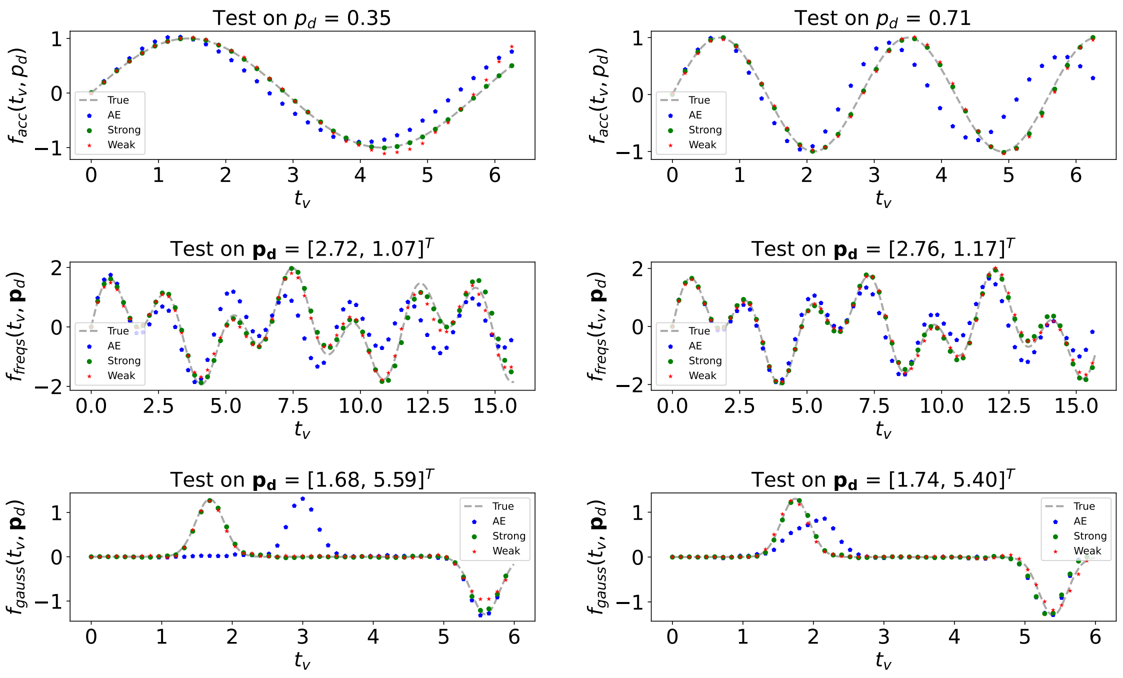


Figure 5: Figure showing the interpolated results of RRAEs with both formulations, as well as a Vanilla AE on the three examples presented with linear/bilinear interpolation in the latent space.

We also present the relative error over all the training/test sets in the following table, As can be seen in both the table and the figures, RRAEs with both formulations can

	Accelerated sin		Mult. Frequencies		Mult. Gausses	
	Train	Test	Train	Test	Train	Test
Vanilla AE	50.57	52.19	11.74	17.72	16.41	31.32
RRAE (Strong)	**3.2**	**3.13**	**4.7**	**9.68**	**8.67**	**14.67**
RRAE (Weak)	9.41	8.62	13.94	15.55	15.31	20.99

Table 2: Table showing the relative error (in %) for all three architectures on both the train and test set for the three examples with one/two parameters.

interpolate much better than Vanilla AEs when linear/bilinear interpolation is used.

Again, the weak formulation is not doing as well as the strong one for the same rank of the latent space which is expected. Most importantly, RRAEs can interpolate between curves with high-rank solutions, even those with different supports (i.e. the multiple gausses). We hope that from the previous section, we gave enough evidence to explain why RRAEs are much better at interpolation than regular AEs, especially with multiple parameters since a reduced basis is found.

5.2 Extensions of RRAES

From the results already presented, it is clear that RRAEs have a lot of potential when it comes to interpolation. However, readers might be wondering, what if I don't know the dimension of the parametric space? On the other hand, since Neural Networks do not discriminate between the dynamics and noise, won't noise be linearised with the data as well? And what if the time series is long (i.e. $T \gg 1$), won't the training take too much time? In the following section, we answer all of these questions by showing that RRAEs can conclude by themselves an approximate dimension of the parametric space, and combining a POD with the proposed model can filter noise and limit the computational overhead. To do so, we choose to interpolate between curves representing crystallization rates, governed by the Avrami equation as follows,

$$X_d(t_v, \mathbf{p}_d) = 1 - e^{-\frac{\pi p_d^1 (p_d^2)^3}{3} t_v^4}, \qquad p_d^1 \in [1.5, 3], \qquad p_d^2 \in [1.5, 3]. \tag{11}$$

Here our parameters p_d^1 and p_d^2 represent the nucleation rate per unit volume and the growth velocity respectively. Again, for details about the chosen parameters for train/test sets, readers are referred to Appendix A. To check whether RRAEs can determine the parametric dimension by themselves, we train the model with a

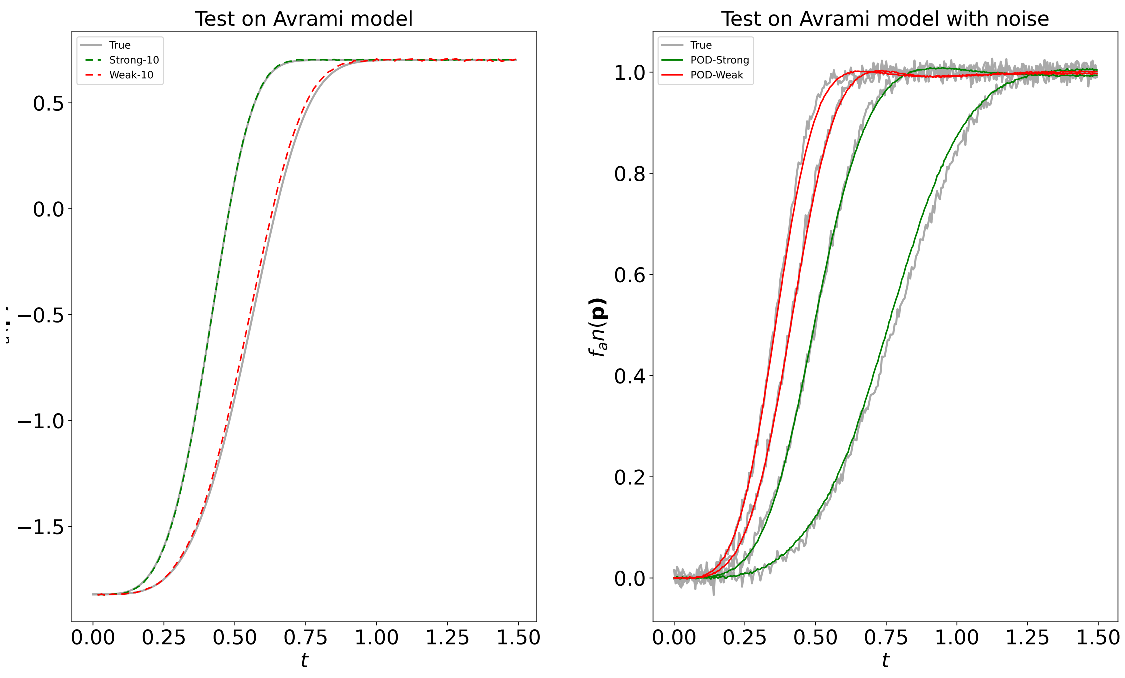


Figure 6: Figure showing the interpolated results of RRAEs with both formulations, as well as a Vanilla AE on the crystallization rates with different latent space dimensions and with/without noise.

latent space of rank 2 (the original dimension of the parametric space), 3, 5, and 10. On the other hand, to show that the model can filter noise, we add random Gaussian noise to the data and train the model combined with a POD. Mainly, the idea is to reduce the dimension of the input matrix X with a POD, and then perform training over the reduced data before going back to the original data by applying the inverse transformation to the output of the decoder. The results for some of the interpolated curves can be seen in figure 6.

To have clearer figures, we only plot the result for training with a latent space of rank 10 (i.e. “Strong-10”, and “Weak 10”) since these must struggle the most in finding the intrinsic dimension of the parametric space. Similarly, we only plot two random curves for each of the strong/weak formulations used with a POD (i.e. “POD-strong”, and “POD-weak”).

As can be clearly shown to the right of the figure, the combination of a POD with an RRAE can filter noise to find smoother curves. Since the POD reduces the dimension, this leads to significantly smaller computational overheads. For example, training was done over a dimension $T_{POD} = 4$ instead of the original $T = 300$.

To be able to properly compare between the different ranks of the latent spaces, we present the relative error for each training in the table below,

	Weak formulation		Strong Formulation	
Rank	Train	Test	Train	Test
2	0.65	3.38	0.36	1.81
3	0.81	1.97	0.35	1.81
5	0.59	1.85	0.41	1.84
10	0.62	1.99	0.68	1.9

Table 3: Table showing the relative error (in %) when different ranks are enforced for the latent space to learn interpolation over the crystallization rates.

The results documented in the table, as well as the two curves plotted to the left of figure 6, show that specifying a rank for the latent space that's different than the intrinsic dimension doesn't affect the results on the test set by much. What's also interesting is that the results for the weak formulation are better when the rank specified is bigger than 2 (the dimension of the parametric space). Again, by comparing the Weak formulation with a PGD, it is expected that the Neural Network can converge to basis vectors that are not orthogonal and hence might need a larger rank (i.e. a larger dimension of the basis) to represent the data best.

Finally, the remaining question is whether those good results are caused by the RRAEs finding the intrinsic dimension, or if they are capable of representing properly a space of dimension two with a basis of dimensions 3, 5, and 10. Accordingly, we plot the first five singular values (normalized) of the latent space in figure 7. In the figure, the blue curve has only two dominant modes, since we enforce the latent space to have a rank of 2. But what's interesting is that no matter how big we choose the rank of the latent space to be, both formulations learn automatically to only use a small rank! Since by using the weak formulation, we allow the Neural Network to directly modify the coefficients, we can see that in all cases, the RRAE finds exactly the intrinsic dimension 2, since the singular values of the latent space are much smaller from the third one onwards. On the other hand, the strong formulation is only able to find an approximation of the intrinsic dimension, since it

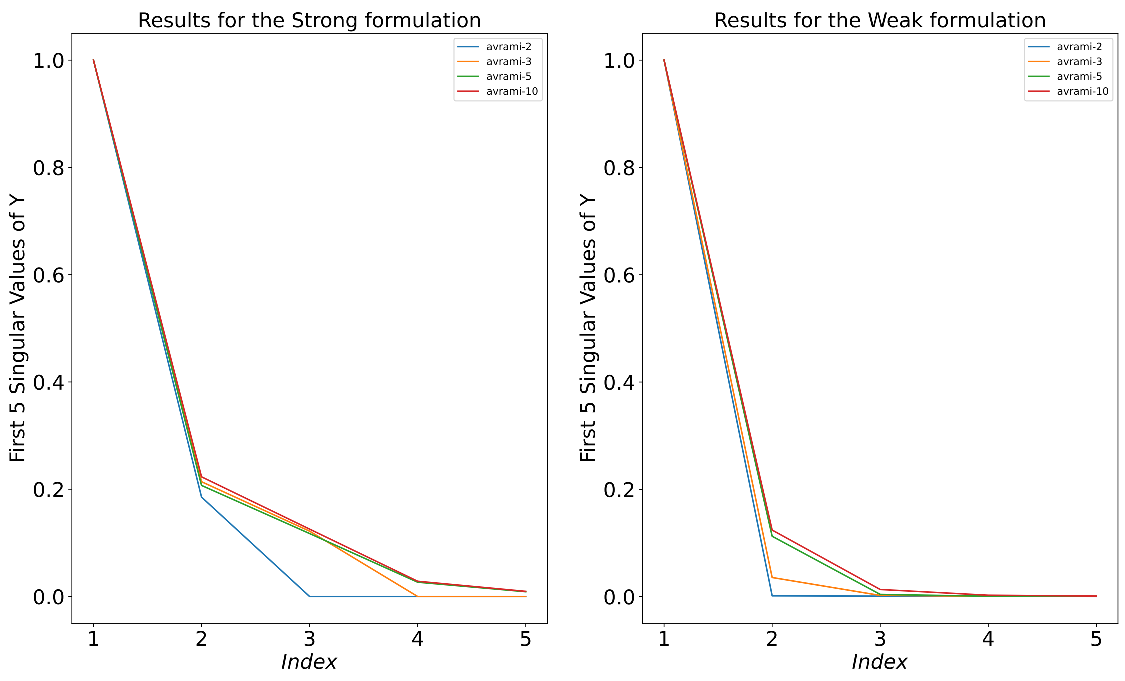


Figure 7: Figure showing the first five singular values of the latent spaced normalized when different ranks of the latent space are enforced with the weak (left) and the strong (right) formulations.

uses approximately a rank 3 in all the cases where $k > 2$. However, as can be seen in table 3, the strong formulation was able to find a basis that represents the whole data even with an additional dimension. In both cases, we show in this example that RRAEs can find an approximation of the dimension of the parametric space by themselves, and hence not knowing the dimension beforehand is not a problem when using these architectures.

6 Summary and Conclusions

In this article, we presented Rank Reduction Autoencoders (RRAEs), Autoencoders that enforce a reduced basis of long latent spaces. We proposed two formulations, the strong formulation which finds a reduced basis that's POD-like, and a weak formulation that forms a PGD-like basis. We also propose to interpolate linearly between the coefficients of the basis instead of the curves in the latent space.

We provided multiple reasons why longer latent spaces with low rank can be much more effective than reduced latent space, and we backed up our arguments with two simple examples.

We tested RRAEs on examples characterized by one/two parameters and showed that both formulations can interpolate well between curves that formed originally a high-rank matrix when the rank of the latent space is enforced to be equal to the dimension of the original parametric space.

Finally, we showed how RRAEs can filter noise and have much less computational overhead when combined with a POD. We also provided an example that showed that not knowing the dimension of the parametric space in advance won't significantly affect the interpolation of RRAEs. In other words, they can find an approximation of the intrinsic dimension by themselves.

All in all, our results show that Rank Reduction autoencoders have a huge potential when it comes to interpolation over nonlinear manifolds. The strong formulation provides the best results on train/test data, and the weak formulation can find a better approximation of the intrinsic dimension of the parametric space if it is unknown in advance.

Readers who are interested in testing the autoencoders for themselves can find our code on our GitHub repository, done in `JAX`. 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 vibratoric 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] Sergio Torregrosa, Victor Champaney, Amine Ammar, Vincent Herbert, and Francisco Chinesta. Hybrid twins based on optimal transport. *Computers & Mathematics with Applications*, 127:12–24, 2022.

[20] Sergio Torregrosa, Victor Champaney, Amine Ammar, Vincent Herbert, and Francisco Chinesta. Surrogate parametric metamodel based on optimal transport. *Mathematics and Computers in Simulation*, 194:36–63, 2022.

[21] 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.

[22] Udo Seiffert. Multiple layer perceptron training using genetic algorithms. In *ESANN*, pages 159–164. Citeseer, 2001.

[23] Sushmita Mitra and Sankar K Pal. Fuzzy multi-layer perceptron, inferencing and rule generation. *IEEE Transactions on Neural Networks*, 6(1):51–63, 1995.

[24] Jiexiong 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.

[25] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

[26] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

[27] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[28] 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.

[29] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang 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.

[30] 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.

[31] 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.

[32] 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.

[33] Geoffrey E Hinton and Richard Zemel. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems*, 6, 1993.

[34] Bhavik Vachhani, Chitralekha Bhat, Biswajit Das, and Sunil Kumar Kopparapu. Deep autoencoder based speech features for improved dysarthric speech recognition. In *Interspeech*, pages 1854–1858, 2017.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] James Sergeant, Niko Sünderhauf, Michael Milford, and Ben Upcroft. Multimodal deep autoencoders for control of a mobile robot. In *Proc. of Australasian Conf. for robotics and automation (ACRA)*, 2015.

[40] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[41] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

[42] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

[43] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[44] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[45] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[46] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[47] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[48] Li Jing, Jure Zbontar, et al. Implicit rank-minimizing autoencoder. *Advances in Neural Information Processing Systems*, 33:14736–14746, 2020.

[49] 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.

[50] Gilbert W Stewart. On the early history of the singular value decomposition. *SIAM review*, 35(4):551–566, 1993.

[51] 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.

Note for training: Since in (5), the constants α 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 α_j^d even if the rest of the Neural Network doesn't need/can't sustain larger learning rates (more details in Section AAA).