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[1,5,6,7]

[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.
[6] CNRS@CREATE, 1 Create Way, 04-05 Create Tower, Singapore, 138602, Singapore.

April 17, 2024

Abstract

Auteoncdoers encounter some limits as interpolators. If the latent space becomes too reduced, feature extraction can be done but their interpolation capabilities are limited to some parts of the parametric space. On the other hand, increasing the dimension of the latent space doesn't lead to any reduction, and hence doesn't help with interpolation. In this paper, we present Rank Reduction Autoencoders (RRAEs), Autoencoders that enforce large latent spaces to enable a linear reduction. We propose two formulations, a strong and a weak one, that enforce the latent space to have a low rank by constructing its reduced basis. We show that both formulations can interpolate efficiently between curves defining high-rank matrices, where the Vanilla Autoencoder fails. Further, we illustrate how preceeding RRAEs by a POD allows us to filter noisy data and reduce computational overhead. We finally show that both formulations can find an approximation of the intrinsic dimension of the parametric space if it is unknown in advance.

Keywords: Rank Reduction, Autoencoders, Non-linear interpolation, Low-rank latent space, Non-linear model-order reduction, Singular Value Decomposition.

1 Introduction

Interpolation over a parametric space is a widely considered topic in the literature since good interpolation allows us to reconstruct the solution of a physical problem in an entire parametric space from a few samples. Multiple techniques have been proposed to perform interpolation, mainly the Proper Orthogonal Decomposition with Interpolation (PODI) [1, 2, 3], or the sparse PGD (sPGD) [4, 5]. Most of these techniques are based on Model Order Reductions, such as the Proper Orthogonal Decomposition (POD) [6, 7], the Proper Generalized Decomposition (PGD) [8, 9, 10], and the Principal Component Analysis (PCA) [11, 12]. These techniques depend on the fact that the solution can be represented by only a few parameters. In other words, if we stack multiple solutions 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 does not apply, the just referred techniques fail to create an efficient surrogate for the correct prediction of physical phenomena. Other techniques based on different formulations such as the Optimal Transport (OT) [13, 14] are also unable to interpolate properly in those settings. The efficiency of the methods presented above, as well as every low-rank technique (e.g. [15, 16, 17, 18]), drops when the solution is of a high rank.

On the other hand, machine learning algorithms are being used in almost every field due to their fast evaluations and their ability to approximate nonlinear behavior. This encouraged researchers to use them for data reduction and interpolation [19, 20]. One of the main architectures that are successful in feature recognition is Autoencoders [21]. By using Neural Networks as their encoding and decoding functions, Autoencoders with reduced latent spaces have been successfully used in many applications such as speech recognition [22, 23], medical applications [24], robotics [25], and others [26]. However, Vanilla Autoencoders can only interpolate on a small part of the parametric domain and hence their use for interpolation is limited. This led to multiple enhancements such as Variational [27], or Sparse Autoencoders [28] which improved Atueoncoders overall but did not definitively solve the interpolation issues.

Recently, the Implicit Rank-Minimizing Autoencoder (IRMAE) [29], and the Low-Rank Autoencoder (LoRAE) [30] 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. The present paper proves that long latent spaces with low-rank can be exploited further.

Based on both the Koopman theory [31] and the kPCA [32], it is known that we can find a linear behavior in larger spaces. Inspired by these, we present Rank Reduction Autoencoders (RRAEs), which have large latent spaces that can be represented by a reduced basis. By enforcing the latent space to accept a linear reduction (hence a lower rank), we show that we end up with more powerful Autoencoders. Furthermore, since we find a reduced basis of the latent space, we can interpolate much more efficiently. In a certain sense, our proposal can be viewed as a kernel-free nonlinear PCA, overperforming usual kPCA techniques.

Our architecture includes two proposed formulations: a strong and a weak one. Throughout the paper, we show that the strong formulation finds orthogonal basis vectors and forms a POD-like basis, while the weak formulation is allowed to find

non-orthogonal ones. We show that both formulations can interpolate efficiently between high-rank solutions 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. Section 4 discusses why RRAEs can be more efficient than Vanilla AEs based on two examples. Then, we compare the interpolation capabilities of RRAEs with Vanilla AEs on a variety of problems in section 5. Finally, we summarize our work in Section 6.

Our results show that RRAEs perform much better with interpolation and that combining them with a POD can filter noise and reduce the computational overhead. Further, RRAEs can find an approximation of the intrinsic dimension of the parametric space if it is unknown in advance.

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 enforce 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 while finding a reduced basis. In other words, let $Y = U^y\Sigma^y V^y$ be the Singular Value Decomposition (SVD) [33, 34] 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,

$$Y = \sum_{i=1}^{r} \sigma_i^y U_i^y (V^y)_i^T \qquad \Rightarrow \qquad Y \approx \sum_{i=1}^{k} \sigma_i^y U_i^y (V^y)_i^T, \qquad k \ll r, \tag{3}$$

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,

$$Y_d = \sum_{i=1}^{k} \sigma_i^y U_i^y (V^y)_{i,d} = \sum_{i=1}^{k} \alpha_{i,d}^y U_i^y, \qquad \forall d \in [1, D], \tag{4}$$

with $(V^y)_{i,d}$ being entry d of vector $(V_i^y)^T$.

Accordingly, for k modes, each column of Y is defined by k coefficients and k vectors. Further, vectors U_i^y form a basis for the latent space. We can write (4) in matrix form as follows,

$$Y = UA, \qquad \text{with: } A_{i,j} = \alpha_{i,j}^y, \quad U \in \mathbb{R}^{L\times k}. \quad A \in \mathbb{R}^{k\times D}, \tag{5}$$

Based on (4), and (5), 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.

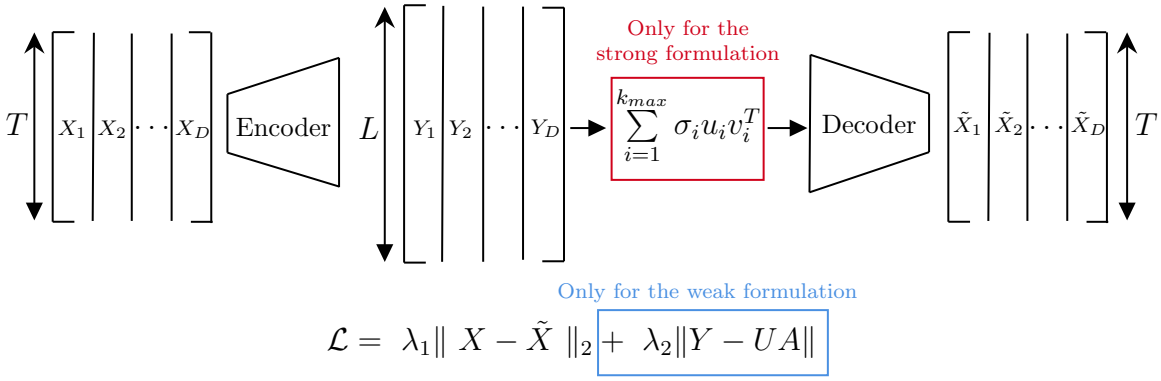

Figure 1: Schematic showing the autoencoder in use as well as both methodologies. There are two terms in the loss function for the Weak formulation. On the other hand, there's an additional step before the decoder for the Strong formulation.

1. The Weak formulation: This formulation is based on (5). After choosing the maximum allowed number of modes k_{max}, we generate two trainable matrices $A \in \mathbb{R}^{k_{max} \times D}$, and $U \in \mathbb{R}^{L \times k_{max}}$. Afterward, we add a term to the loss as seen in blue in figure 1. By doing so, minimizing the loss means that the latent space would have at most a rank of k_{max}. After convergence, we can say that the columns of our trainable matrix U form the reduced basis of the latent space. Additionally, the coefficients found in matrix A describe how to form each column Y_d as a linear combination of the basis vectors. 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 than the specified value of k_{max}.

2. The Strong formulation: Unlike the weak formulation, this architecture enforces, in a strong manner, the maximum dimension of the reduced basis of the latent space. Similarly to the first formulation, we begin by choosing the maximum rank k_{max} of the latent space. Then, as seen in red in figure 1, a truncated SVD (of order k_{max}) of the latent space is given to the decoder, instead of the latent space itself. Accordingly, the input of the decoder will have at most k_{max} dominant singular values. We refer to this method as the Strong formulation since we strictly enforce the latent space to have a rank that's lower or equal to k_{max}. In this case, the basis vectors and coefficients are simply the ones found by the truncated SVD.

When using the strong formulation, we compute 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 interpolation easier. On the other hand, backpropagation through the singular value decomposition is not common in practice. All the work presented in this paper was done using `equinox` in `JAX`, where gradients of the singular value decomposition are implemented and accessible.

In both formulations, k_{max} is a hyperparameter to be chosen. We propose to choose k_{max} to be equal to the dimension of the parametric space. If this dimension is unknown, and as shown later in Section 5, any choice of k_{max} that is small (to enforce a reduced basis), but larger than the dimension of the parametric space can be done. RRAEs will then find an approximation of the required rank by themselves.

3 Interpolation in the latent space

The fact that 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 mapping the data into a longer latent space while enforcing a low rank, we can find a simpler space where a linear interpolation would be enough to find the new curve, before going back to the original space from using the decoder.

Both proposed formulations allow us to approximate the latent space by (4) or (5). 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 the basis vectors (i.e. columns of U) are fixed for all solutions, we can interpolate between the coefficients $\alpha^y_{j,d}$,

$$Y_d = \sum_{j=1}^{k_{max}} \alpha^y_{j,d} U^y_j \qquad \Longrightarrow \qquad Y_d(\mathbf{p}_i) = \sum_{j=1}^{k_{max}} \gamma_j(\mathbf{p}_i) U^y_j, \tag{6}$$

where each $\gamma_j : \mathbb{R}^P \to \mathbb{R}$, could be any mapping, that maps all the training parameters to the corresponding $\alpha^y_{j,d}$, and allows us to interpolate when used on new values of $\mathbf{p}$. Since we interpolate between coefficients, the computational cost of RRAEs for interpolation is the same as the cost of using a Vanilla Autoencoder with a latent space of dimension k_{max}.

For instance, for a parameter space of dimension one and sorted coefficients, we can interpolate linearly by writing,

$$\gamma_j(p) = \alpha^y_{j,m} + \frac{\alpha^y_{j,m+1} - \alpha^y_{j,m}}{p_{m+1} - p_m}(p - p_m), \qquad \forall p \in [p_m, p_{m+1}]. \tag{7}$$

In this paper, we show that interpolating linearly for parametric spaces of dimension one, and bilinearly for those of dimension two is enough. In general, each γ_j can be approximated by any regression, including a Neural Network.

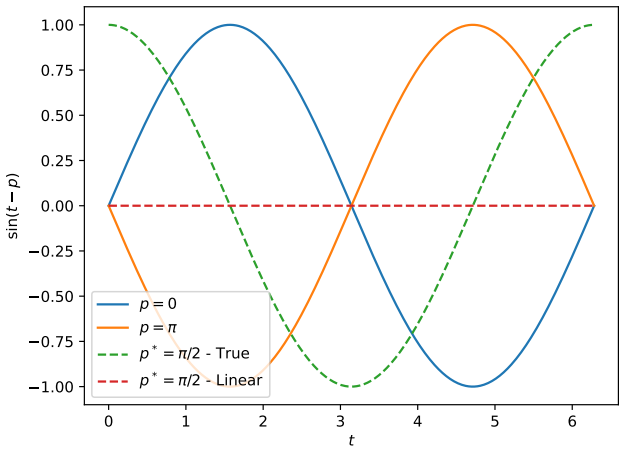

Figure 2: Figure showing the result when interpolating linearly between two shifted sine curves, showing why linear interpolation is not a good choice for problems with multiple dominant singular values.

4 Insights behind RRAEs

The RRAEs' architecture improves autoencoders' performances, whether on the train set or for interpolation in the latent space. A large latent space can allocate a linear behavior (as employed for instance in the Koopman theory, or the kPCA). Furthermore, by finding a reduced basis that spans the latent space, we can better interpolate to fit the test data. The Vanilla Autoencoder on the other hand has "holes" in its interpolation [29], since it does not find a basis, but only coefficients that are helpful for the decoder to retrieve the solution.

To illustrate 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 sine curves since these have a simple nonlinearity, but they are hard to separate (nonmonotonic and cross each other multiple times). For our second example, we chose curves with stair-like behavior. In that case, we create highly nonlinear curves (different supports, different numbers of jumps of different magnitudes), 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,

$$\begin{cases} X_d(t_v,\, p_d) = f_{shift}(t_v,\, p_d) = \sin(t_v - p_d\pi), & p_d \in [0, 1.5], \\ X_d(t_v,\, p_d) = f_{stair}(t_v,\, p_d,\, \text{args}) & p_d \in [1, 5], \end{cases} \tag{8}$$

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: $p_d \in \mathbb{R}$, $t_v \in \mathbb{R}^T$, $(\text{Ph}_0, \text{Amp}_0, \kappa, y_0, w) \in \mathbb{R}$

$$\text{Amp}_{p_d} = p_d$$

$$\text{Ph}_{p_d} = \text{Ph}_0 + \kappa(\text{Amp}_{p_d} - \text{Amp}_0)$$

$$g_{p_d}(t_v) = \text{Amp}_{p_d}\sqrt{t_v}\sin(w(t_v - \text{Ph}_{p_d})) - y_0$$

$$h_{p_d}(t) = \left(\frac{|g_{p_d}(t)| + g_{p_d}(t)}{2}\right)^5$$

$$X_d(t_v,\, p_d) = \text{cumsum}(h_{p_d}(t_v))$$

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,

$$\begin{cases} \text{Ph}_0 = 0.875, & \text{Amp}_0 = 1 \\ \kappa = 2.286, & y_0 = 2.3, \quad w = 2\pi. \end{cases} \tag{9}$$

Training is performed over 14, and 35 equidistant values of p_d for the shifted sine 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. We then linearly interpolate in the latent space to predict the test set. The training parameter, including the dimension of the latent space, can be found in Appendix AAA. The relative error over all p_d values for both the train and test sets is summarized in table 1,

	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.

For each example, two of the hardest curves to interpolate are plotted in figure 3. The results show that when curves are hard to separate, RRAEs are much better interpolators than Vanilla AEs. On the other hand, the effect of longer latent spaces is reduced for simple curves that can be highly nonlinear, but characterized by one parameter, and easily separable. Further, the weak method never does as well as the strong one since the required conditions are imposed weakly. What's interesting is that the weak formulation is doing as poorly as the Vanilla AE for a very small value of p_d (subplot at the top left), while it has a much lower error overall.

To further investigate the discrepancy, we plot the coefficients to be interpolated in the latent space as a function of the corresponding parameter in figure 4. It is important to note that the coefficients are defined differently between the RRAE and the Vanilla AE. For RRAEs, when $k_{max} = 1$, the coefficients are simply the

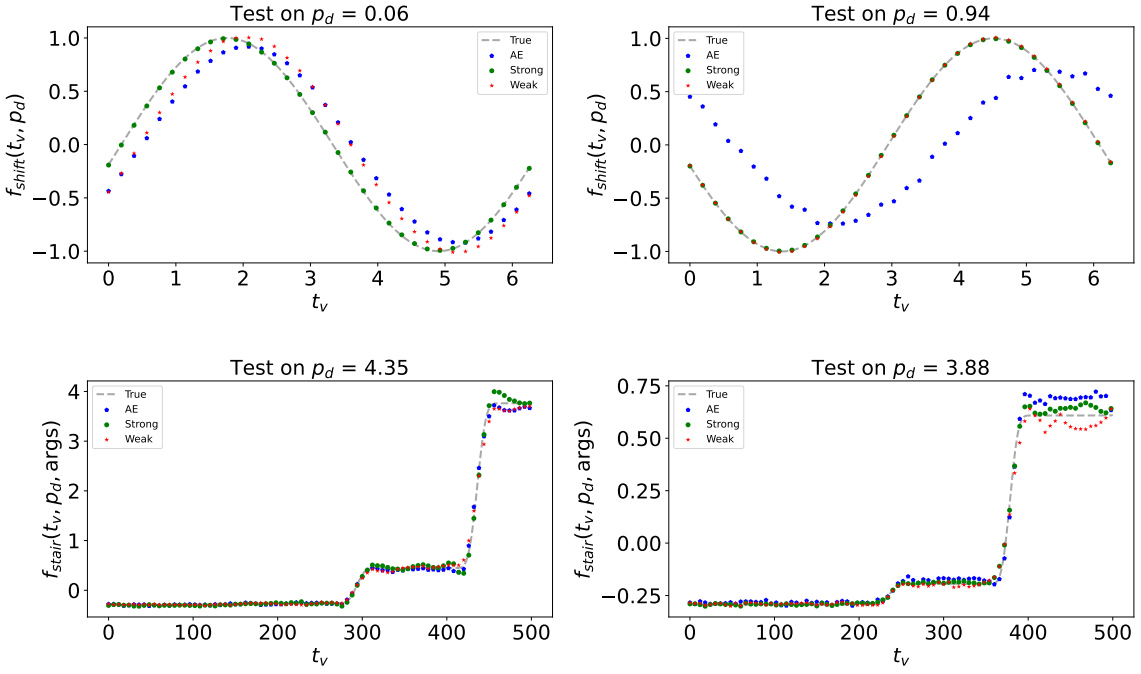

Figure 3: Figure showing the predictions of Vanilla Autoencoders and RRAEs with both formulations over two particular values of p_d for the shifted sine (above) and the stair-like examples (below).

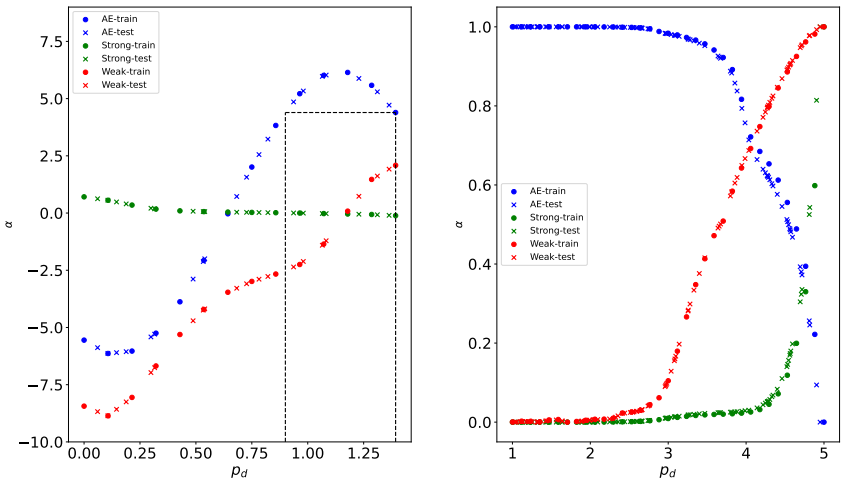

Figure 4: Figure showing the coefficients to be interpolated (dots) for all three architectures, and the interpolated values for the test set (crosses) for the shifted sine curves (left) and the stair-like curves after normalization (right).

entries of $A \in \mathbb{R}^D$ in (5). On the other hand, for a Vanilla Autoencoder with a scalar latent space, the values in the latent space themselves are the coefficients.

The main problem with the coefficients found by the Vanilla AE for the shifted sine curves (the blue crosses and dots in figure 4 (left)) is that the resulting curve from linearly interpolating the coefficients is not an injection, over most of the domain. Specifically, for any value of p_d in between the black vertical dashed lines, there exists another value with the same coefficient α. Accordingly, the decoder will find the same curve for two different parameters, which is wrong since p_d defines a shift. The same thing can be said about another interval from $p_d = 0$ to around $p_d = 0.3$. In other words, Vanilla AEs can't perform good predictions except on approximately the range $p_d \in [0.3, 0.85]$.

On the other hand, as proposed earlier, a longer latent space allows us to find better coefficients. This is clearly shown by the coefficients of the strong method in figure 4 (left), which have a monotonic behavior. On the other hand, the weak formulation finds monotonic coefficients in the range $p_d \in [0.2, 1.4]$ but has the same problem as the Vanilla AE when $p_d \in [0, 0.2]$. Accordingly, the RRAE with the weak formulation can interpolate on most of the domain except a small region with the lowest parameter values. This fact explains why the weak formulation is doing as bad as the Vanilla AE for $p_d = 0.06$, and why its relative error over all the test curves is much lower than the Vanilla Autoencoder's error.

Finally, the right part of the figure depicts how when the curves are simple to separate and are characterized by only one parameter, all formulations, including the Vanilla AE, can find monotonic coefficients that fit both the train and test sets.

5 Testing on Numerical Data

The solutions interpolated in the previous section were only characterized by one parameter. However, when the dimension of the parametric space is bigger than one, RRAEs are expected to outperform Vanilla AEs even more. In this section, we test RRAEs and compare them to Vanilla AEs on a harder case with one parameter, and several examples with a parametric space of dimension two.

5.1 Examples with one/two parameters

We generated several challenging tests for interpolation. First, we choose accelerated sine 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 sine curves with different frequencies, as well as two Gaussian bumps in two different locations. We show how in such examples, Vanilla Autoencoders are not able to interpolate, and that both our formulations provide much better results for the same training parameters (again, training details can be found in Appendix AAA). For all the examples, we train RRAEs with k_{max} that's equal to the dimension of the parametric space, and we compare it with a Vanilla AE with a latent space of dimension k_{max}. We define the columns of our input matrix $X_d(t_v, p_d) = f_{prob}$ for each problem as follows,

$$\begin{cases} f_{acc}(t_v, p_d) = \sin(p_d t_v), & p_d \in [0, 1.5], \\ f_{freqs}(t_v, \mathbf{p}_d) = \sin(p_d^1 \pi t_v) + \sin(p_d^2 \pi t_v), & p_d^1 \in [0.3, 0.5], \quad p_d^2 \in [0.8, 1], \\ f_{gauss}(t_v, \mathbf{p}_d) = 1.3e^{-\frac{(t_v - p_d^1)^2}{0.08}} + 1.3e^{-\frac{(t_v - p_d^2)^2}{0.08}}, & p_d^1 \in [1, 3], \quad p_d^2 \in [4, 6], \end{cases}$$

Where 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 $\mathbf{p}_d = [p_d^1, p_d^2] \in \mathbb{R}^2$. For each example and each of the three architectures, we present some interpolated predictions in figure 5.

We also present the relative error over all the training/test sets in table 2,

	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.

As can be seen in both the table and the figures, RRAEs with both formulations can interpolate much better than Vanilla AEs when linear/bilinear interpolation is used in the latent space.

5.2 Extensions of RRAES

In this section, we show why, if the dimension of the parametric space is unknown, the choice of the hyperparameter k_{max} is simple. We also show how combining a

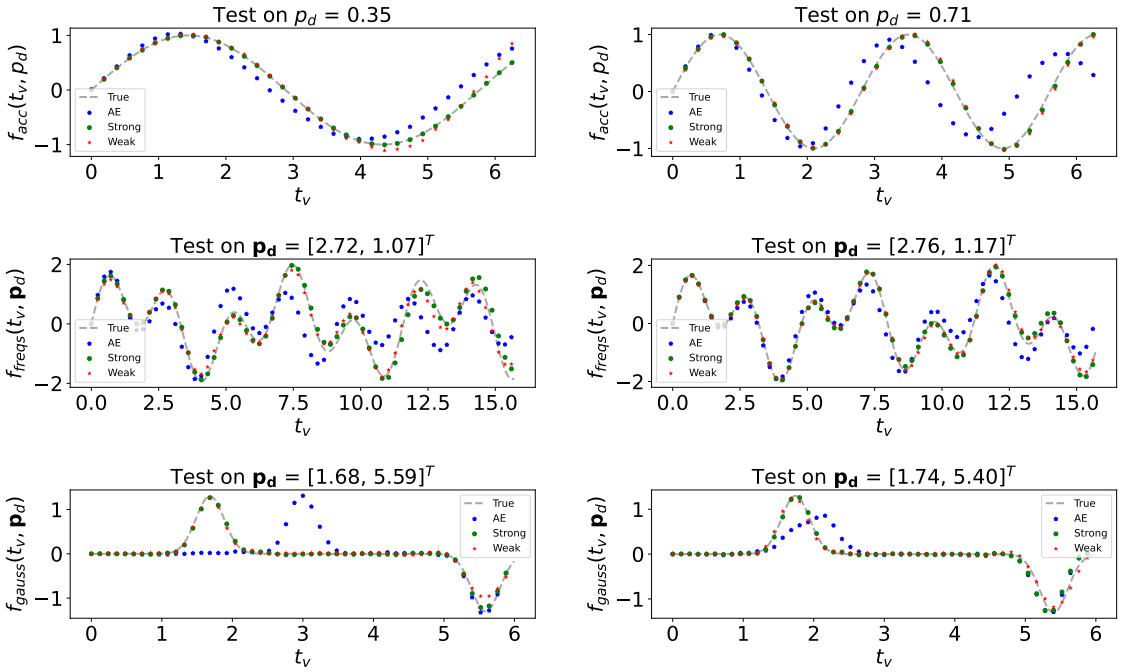

Figure 5: Figure showing the interpolated results of RRAEs with both formulations, as well as a Vanilla AE on the three examples presented with linear/bilinear interpolation in the latent space.

POD with RRAEs can filter noise and reduce computational time. To do so, we choose to interpolate between curves representing crystallization rates, governed by the Avrami model as follows,

$$X_d(t_v, \mathbf{p}_d) = 1 - e^{-\frac{\pi p_d^1 (p_d^2)^3}{3} t_v^4}, \qquad p_d^1 \in [1.5, 3], \qquad p_d^2 \in [1.5, 3]. \tag{10}$$

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. On this dataset, we train RRAEs with a latent space of maximum 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 use it to train RRAEs combined with a POD. Mainly, the objective 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").

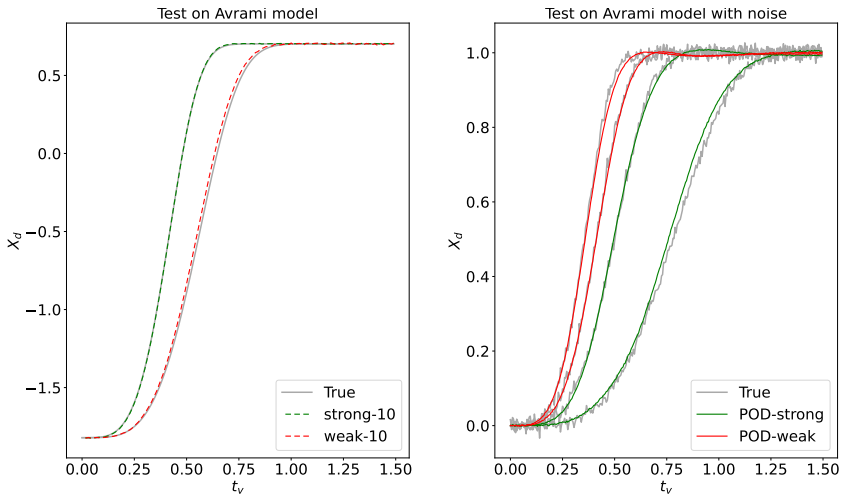

Figure 6: Figure showing the interpolated results of RRAEs with both formulations on the crystallization rates with a maximum rank of 10 (left) and with noise (right).

As can be clearly shown in figure 6 (right), 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 $T = 300$, T being the first dimension of the input matrix $X \in \mathbb{R}^{T \times D}$.

On the other hand, for the choice of k_{max}, one can usually find a good approximation of the required rank by starting with a low one and increasing it until training works. However, the results reported in table 3, as well as the plots in figure 6 show that the choice of k_{max} is not crucial. As long as k_{max} is chosen to be small (to enforce a reduced basis) and bigger than the intrinsic dimension, the results on the test set are not significantly impacted.

	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.

To further investigate the consistency of the good results on the test set for different imposed values of k_{max}, we plot the first five singular values (normalized) of the latent space in figure 7. In that figure, the blue curve has only two dominant modes, since we enforce the latent space to have a maximum rank of 2. However, it can be seen from the sharp decrease of all the other curves that RRAEs can find an approximation of the intrinsic dimension if k_{max} is set to a larger value. In these cases, the Neural Network converges to a latent space with a rank that's lower than the maximum enforced one k_{max}. It is important to note that k_{max} has to be small, otherwise, the Neural Network ends up finding high-rank solutions and we fall back to a Vanilla Autoencoder with an enlarged latent space.

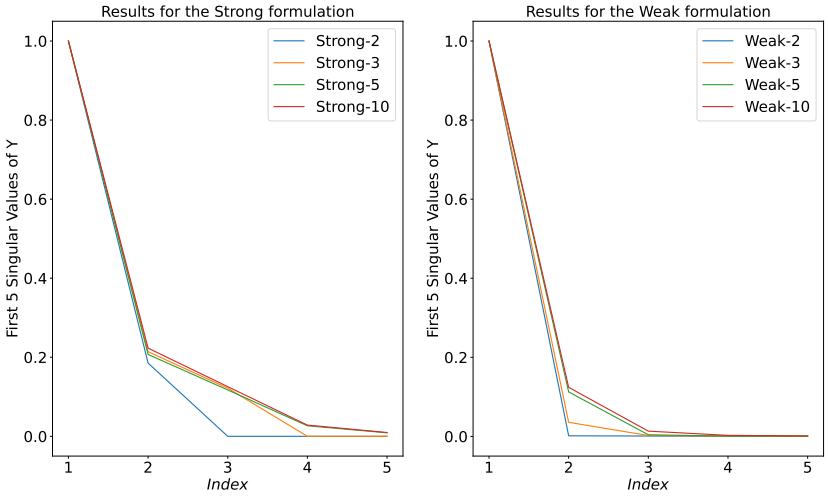

Figure 7: Figure showing the first five singular values of the latent spaced normalized when different ranks of the latent space are enforced with the strong (left) and the weak (right) formulations.

6 Summary and Conclusions

In this article, we presented Rank Reduction Autoencoders (RRAEs), Autoencoders with latent spaces that accept linear reduction. We proposed two formulations, a weak and a strong one to find the latent space while building its reduced basis. Even though the basis vectors in the strong formulation are orthogonal, and they need not be in the weak formulation, we showed that both formulations can interpolate well between curves. Overall, the strong formulation was better than the weak one over all the examples tested. However, the weak method is much simpler to implement, since it only consists of adding a term to the loss.

Our findings show that longer latent spaces with a reduced basis lead to more efficient Autoencoders. The enhancements brought by RRAEs are limited when interpolating between simple examples with one parameter. However, RRAEs are more performant interpolators between more complicated curves and parametric spaces. Further, they have the same computational cost as Vanilla AEs for interpolation.

Finally, we showed that, when combined with a POD, RRAEs can filter noise while reducing the computational overhead. They can also approximate the intrinsic dimension of the parametric space if a larger one (but smaller than the original dimension), is specified as the maximum rank of the latent space.

References

[1] Marco Tezzele, Nicola Demo, and Gianluigi Rozza. Shape optimization through proper orthogonal decomposition with interpolation and dynamic mode decomposition enhanced by active subspaces, 2019.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Francisco Chinesta and Elias Cueto. *PGD-based modeling of materials, structures and processes*. Springer, Switzerland, 2014.

[11] Caterina Labrín and Francisco Urdinez. Principal component analysis. In *R for political data science*, pages 375–393. Chapman and Hall/CRC, 2020.

[12] 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.

[13] 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.

[14] 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.

[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] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

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

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

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

[30] 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.

[31] Steven L Brunton, Marko Budišić, Eurika Kaiser, and J Nathan Kutz. Modern koopman theory for dynamical systems. *arXiv preprint arXiv:2102.12086*, 2021.

[32] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. *Kernel Principal Component Analysis*, volume 1327, pages 583–588. 10 2006.

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

[34] 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 (4), 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).