Supplementary for 'An Additive Autoencoder for Dimension Estimation'*

Kärkkäinen, Tommi and Hänninen, Jan
tommi.karkkainen@jyu.fi, jan.p.hanninen@jyu.fi
Faculty of Information Technology, University of Jyväskylä, Finland

September 17, 2022

All results from the computational experiments are given in this Supplementary Information (SI). Datasets and the final forms of the overall conclusions are described in the main body of the paper. Main facets of the autoencoding techniques are detailed in Section 2 of the main paper. The increment of the tested values of the squeezing dimension $n_{\tilde{L}}$ for the small-dimension datasets was one (starting from one) and for the large-dimension datasets ten (starting from ten). The maximum squeezing dimension for small-dimension datasets was $n-1$ (where n is the number of features) and for large-dimension datasets $\lfloor 0.6 \times n \rfloor$.

1 Preliminaries: deep learning

Deep learning has been an extremely active field of research and development in the twenty-first century (Alzubaidi et al. 2021). These techniques reveal repeatedly promising results in new application areas (Carletti et al. 2020). However, deep networks might be overly complex and equal performance could be reestablished after significant pruning (Chen & Zhao 2018). Deep neural networks are sensitive to small variations in training and architectural design parameters, thereby requiring careful calibration (Guo et al. 2017) and meticulousness in analyzing and comparing different models (Lathuilière et al. 2020). These factors have caused, e.g., the emergence of automated neural architecture search techniques (Elsken et al. 2019). Nevertheless, as indicated in Sejnowski (2020), it is important to improve our basic understanding of both the empirical and theoretical bases of deep learning. Therefore, systematic methods to analyze the behavior of deep neural networks are needed (Yu & Principe 2019).

Indeed, there exist some fundamental aspects of deep learning in which theory and practice are not fully aligned. The first issue is the univer-

*Supplement for manuscript submitted to review

sal approximation capability, whose main shallow results are reviewed, for example, by Pinkus (1999) and whose general importance was excellently summarized in (Hornik et al. 1989, p. 363): "We have thus established that such [feedforward] 'mapping' networks are universal approximators. This implies that any lack of success in applications must arise from inadequate learning, insufficient numbers of hidden units or the lack of a deterministic relationship between input and target." To put it succinctly, a sufficiently large width of one or at most two hidden layers is, according to the approximation theory, enough for an accurate approximation of a nonlinear function, which is implicitly represented by a discrete set of examples in the training data.

Another aspect is the dominant approach to training a deep network structure—that is, estimating unknown weights in different layers. This is realized by applying a certain form of the steepest gradient descent method with a rough approximation of the true gradient, using one observation (online or stochastic gradient descent) or a subset of observations (minibatch). In classical optimization, even the batch gradient descent using the complete data is considered a slow algorithm but still a convergent one provided that the gradient of the minimized function is Lipschitz continuous and the line search—that is, the determination of the step size (the learning rate in neural networks terminology) when moving to the search direction—satisfies specific decrease conditions (Dennis Jr. & Schnabel 1996, Theorems 6.3.2 and 6.3.3) and (Nocedal & Wright 2006, Theorem 3.2). In deep learning, step size may be fixed to a small positive constant (Goodfellow et al. 2016) or be based on monitoring the first- and second-order moments of the gradient during the search with direct updates (Kingma & Ba 2015). Often restricted to a fixed number of iterations and not assuring stopping criteria measuring fulfillment of the optimality conditions (Dennis Jr. & Schnabel 1996, Section 7.2), this implies that the actual optimization problem for determining the weights of a DNN might be solved inaccurately (Taylor et al. 2016).

In a genuine supervised learning for a regression model or a classifier, inexact optimization can be tolerated when seeking the best generalizing network. Then, the search of weights that provide better minimizers for the cost/loss function is stopped prematurely when the test or validation error of the model begins increasing as an indication of overlearning. In such a case, we are not seeking the most accurately optimized network but the best generalizing model based on another error criterion at the meta-optimization level. Theoretically, however, generalization and optimality can be linked in certain respects: An outer-layer locally optimal network, independently on the level of optimality of the hidden layers, provides an unbiased estimate of the prediction error over the training data in the sense of mean, median, or spatial median Kärkkäinen & Heikkola (2004).

In a typical use case, the goal of dimension reduction through autoencod-

ing is not generalization but more compact representation that encapsulates variation of data. Similar to linear dimension reduction techniques, when attempting to represent the given data accurately in lower dimensions, we aim for the best possible reconstruction accuracy. Then, the cost function that measures the autoencoding error (such as the least-squares error function) must be solved with sufficiently high optimization accuracy because of the direct correlation: The smaller the cost function, the better the autoencoder.

2 Preliminaries: optimization

We use one first- and one second-order memory-efficient optimizer to minimize the cost function in equation (4) of the main body: the Adam optimizer (Kingma & Ba 2015) and the limited memory quasi-Newton (L-BFGS) method (Nocedal 1980). The experiments are performed with Matlab, for which the Poblano toolbox[1] was chosen as the basis of the L-BFGS method. It includes the cyclic updates for the band-limited inverse Hessian approximation and utilizes the rigorous line search routines given in[2]. We implemented Adam by ourselves by following the pseudo-code as given in Algorithm 1 in (Kingma & Ba 2015).

A standard method to speed up the minimization of an error-over-training-data cost function is to incorporate only one or a minibatch of observations in the computation of the derivatives for the descent direction. However, a small or random choice of the subset of observations implies a highly inaccurate approximation of the original cost function and its gradient, while we should instead pursue highly accurate nonlinear optimization when performing dimension reduction through autoencoding. Computational efficiency and sufficient accuracy could, in principle, be integrated, if every minibatch approximated the original data density distribution, so that a reduced cost function and its derivatives would provide an accurate and, as constructed in formula (4) of the main paper, automatically scalable approximation of the entire data problem. This and the direct pretraining described in Section 2.2 of the main paper are the main differences between this work and (Le et al. 2011), where minibatching was proposed and tested using the L-BFGS method and the conjugate gradient method.

The density preserving or distribution optimal (DOP) folding algorithm, primarily used in cross-validation, was proposed in (Moreno-Torres et al. 2012), with a complete implementation given in (Kärkkäinen 2014). This method, when applied without additional stratification due to class labels, is given in Algorithm 1. In the following account, we refer to the approach where the DOP algorithm is used to generate subsets of training data as DOP minibatching. Another parameter to control the accuracy of the ap-

[1] `https://github.com/sandialabs/poblano_toolbox`
[2] `http://www.cs.umd.edu/users/oleary/software/`

Algorithm 1 Distribution optimal minibatching.

Input: Data $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ and the number of minibatches k
Output: Labels indicating k non-disjoint subsets $\mathbf{B}_i, i = 1, \ldots, k$, such that $\mathbf{X} = \cup_{i=1}^{k} \mathbf{B}_i$

1: **while** $|\mathbf{X}| \geq k$ **do**
2: Let $\mathbf{x}_1$ be a random observation from $\mathbf{X}$
3: Let $\mathbf{x}_2, \ldots, \mathbf{x}_k$ be $k-1$ closest neighbors of $\mathbf{x}_1$ from $\mathbf{X}$
4: $\mathbf{B}_i = \mathbf{B}_i \cup \{\mathbf{x}_i\}$ and $\mathbf{X} = \mathbf{X} \setminus \{\mathbf{x}_i\}, i = 1, \ldots, k$
5: **end while**
6: Place the remaining observations from $\mathbf{X}$ into different subsets $\mathbf{B}_i, i = 1, \ldots, |\mathbf{X}|$

proximation of the cost function in equation (4) of the main paper is the number of minibatches or their size compared to the amount of the entire data. Thus, to preserve the approximation capability of the original data density, we use only a small number of minibatches.

With the generated set of DOP minibatches, we modify the Adam optimizer in the usual manner by using an approximate gradient based on a randomly selected minibatch for moving averages and to perform the weight update. Note that online random sampling in stochastic gradient descent assures an unbiased gradient estimate Needell et al. (2016). This is exactly what we also obtain, but with better stepwise accuracy through creating and using a set of DOP minibatches.

We suggest a similar arrangement in the L-BFGS method as well. With this algorithm, it is not at all clear how the L-BFGS updates and the line search method react to the iteration-by-iteration changing approximate gradient and cost function value when ensuring sufficient decrease conditions. Theoretically, when the negative gradient multiplied by the L-BFGS update provides a descent direction, the complete optimization algorithm with a rigorous line search can be shown to converge (Dennis Jr. & Schnabel 1996, Theorems 6.3.2 and 6.3.3).

Let us describe the stopping criteria for the optimization algorithms, where typical choices with the classical optimization are summarized in (Dennis Jr. & Schnabel 1996, Section 7.2). We would like to ensure a sufficient decrease in the cost function value and/or a sufficiently small value of the norm of the gradient vector. They could be measured as is (for a selected vector norm), in a relative manner, or by monitoring the behavior of the most recent iterates. Fixed or absolute tolerances must be avoided here, because estimating the nonlinear residual of the autoencoder with a gradually enlarged size of the squeezing layer produces increasingly smaller residual scales for the cost function and the gradient. This search scale is measured with $\mathcal{J}^0$—that is, with the value of the cost function in the be-

ginning of the search. The Adam algorithm is stopped when the decrease in the cost function value (i.e., the difference between the last two iterates) is below $\varepsilon_a \cdot \mathcal{J}^0$, where ε_a is the tolerance set by the user. The L-BFGS method is stopped when the Euclidean norm of the gradient g is sufficiently small: $\|g\| \leq \varepsilon_l \cdot \mathcal{J}^0/Vars$ with the user-defined tolerance ε_l and $Vars$ denoting the number of optimized parameters. In addition, the upper limits for the maximum number of function evaluations (FunVals) for Adam and L-BFGS and for the number of iterations (MxIts) for the L-BFGS method are defined by the user.

1HID is used as the additive autoencoder's reference nonlinear residual approximation model. Its finetuning uses the complete dataset in the optimization and otherwise the default parameters of the L-BFGS optimizer from the Poblano toolbox, except setting the maximum number of iterations MxIts = 5000, thereby allowing at most 20 000 function calls, and using ε_l = 5e-4 in the gradient-based stopping criterion (See Section 2.2 of the main paper). Optimization of 1HID was initialized with the result $(\mathbf{W}_1^T, \mathbf{W}_1)$ of 1SYM, which always used two distributionally optimal folds (L-BFGS-DOP2), MxIts = 2000, and ε_l = 1.e-5. The same pretraining settings were also used in the layerwise construction of the deeper models from a sequence of 1SYMs as depicted in Section 2.3 of the main paper.

Because of the memory requirements, the distribution optimal DOP-algorithm 1, for MNIST and FashionMNIST, was executed using a Linux supermicro with an Intel Xeon E7-8837 CPU and 1 Tbyte of shared memory instead of a Laptop or a server with less memory which were used for other computations.

3 Identification of the intrinsic dimension

The identification of the intrinsic dimension is based on the behavior of MRSE of the 1HID model and the thresholding method depicted in Section 2.3 of the main paper.

Figures for all tested dataset are given with MRSE on the left for PCA, 1HID, and 1SYM, and improvement of the errors (backward difference) for 1HID and 1SYM on the right with the detection threshold illustrated in dashed red. The threshold values were iteratively searched using the visualizations given below. The threshold τ = 4e-3 was applied for the small-dimension datasets. For the larger datasets, two values: τ = 3e-3 or τ = 3e-2, were used and are specified in the corresponding figure captions. After small-dimension and large-dimension normal plots, zooms of thresholding to identify the intrinsic dimensions for a few datasets with gradual decrease of MRSE are given.

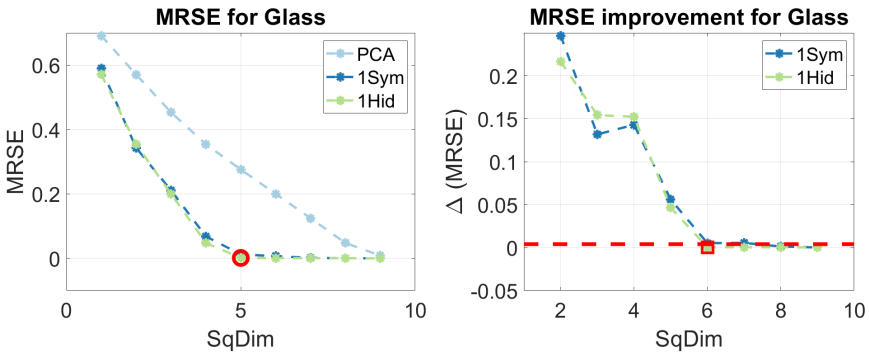

Figure 1: Identification of the intrinsic dimension for Glass.

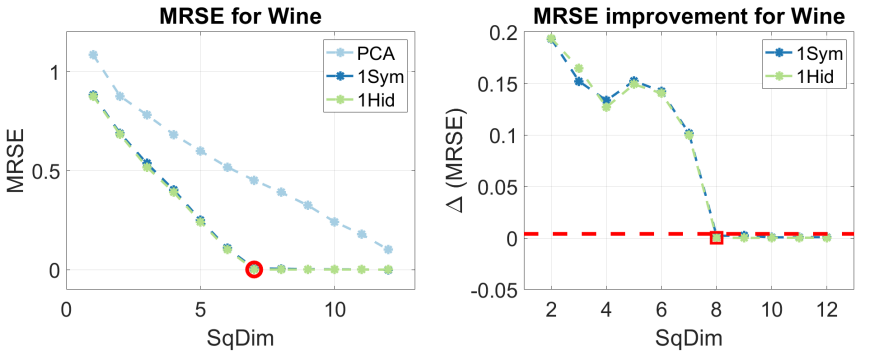

Figure 2: Identification of the intrinsic dimension for Wine.

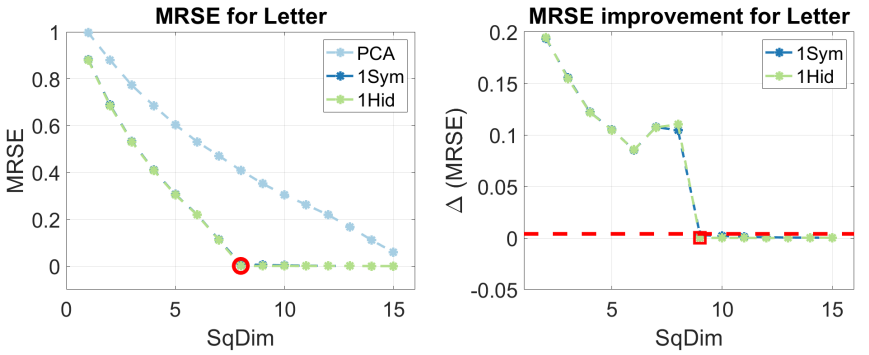

Figure 3: Identification of the intrinsic dimension for Letter.

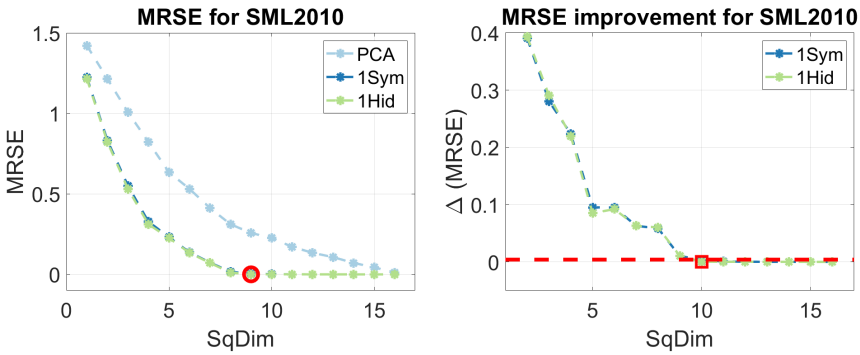

Figure 4: Identification of the intrinsic dimension for SML2010.

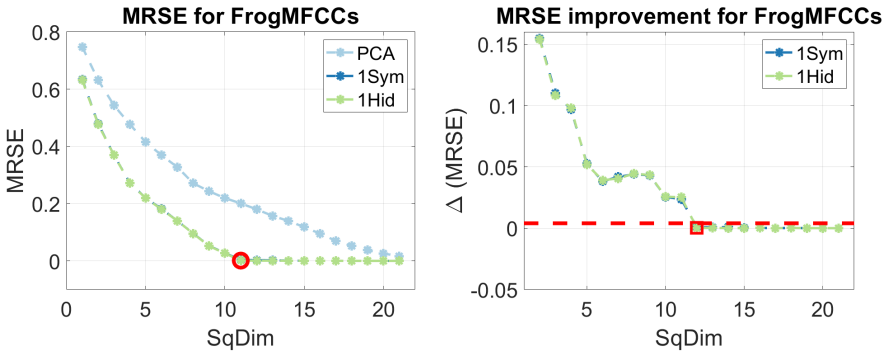

Figure 5: Identification of the intrinsic dimension for FrogMFCCs.

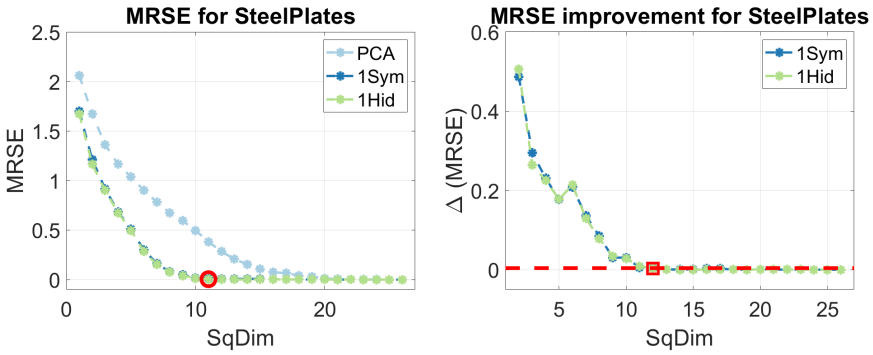

Figure 6: Identification of the intrinsic dimension for SteelPlates.

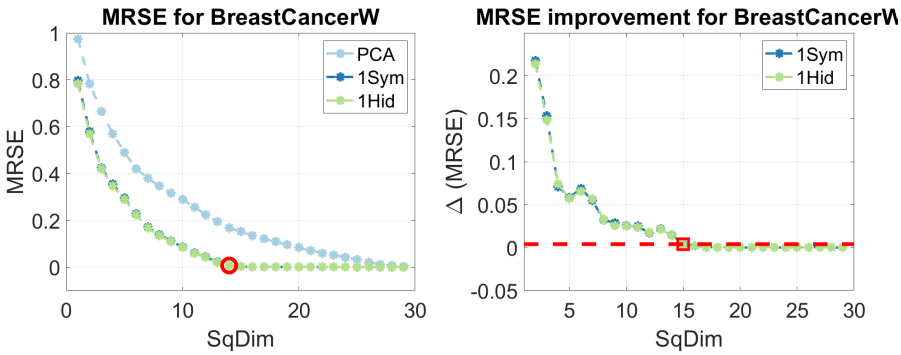

Figure 7: Identification of the intrinsic dimension for BreastCancerW.

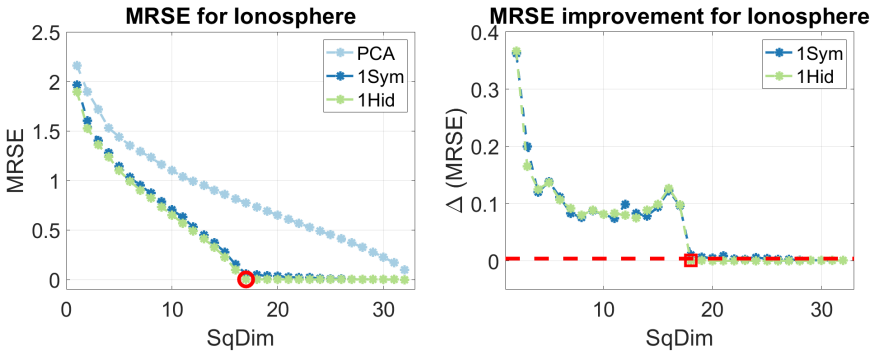

Figure 8: Identification of the intrinsic dimension for Ionosphere.

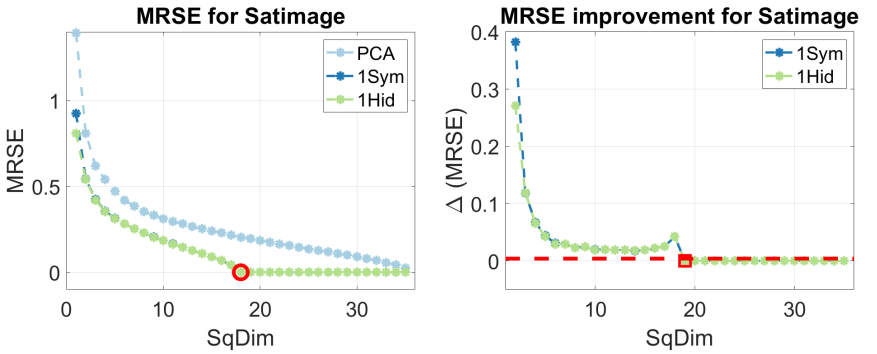

Figure 9: Identification of the intrinsic dimension for Satimage.

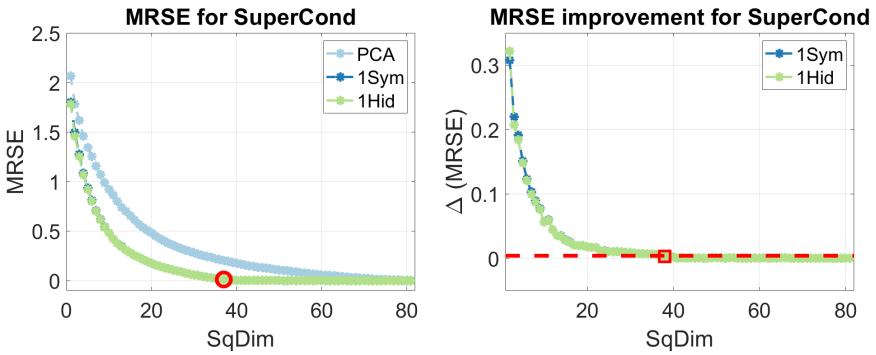

Figure 10: Identification of the intrinsic dimension for SuperCond.

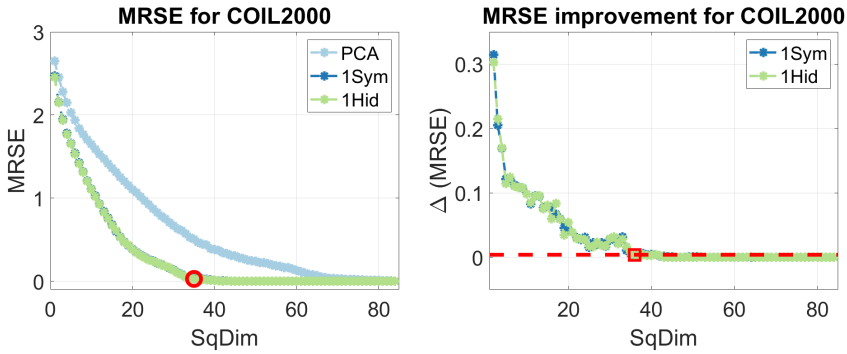

Figure 11: Identification of the intrinsic dimension for COIL2000.

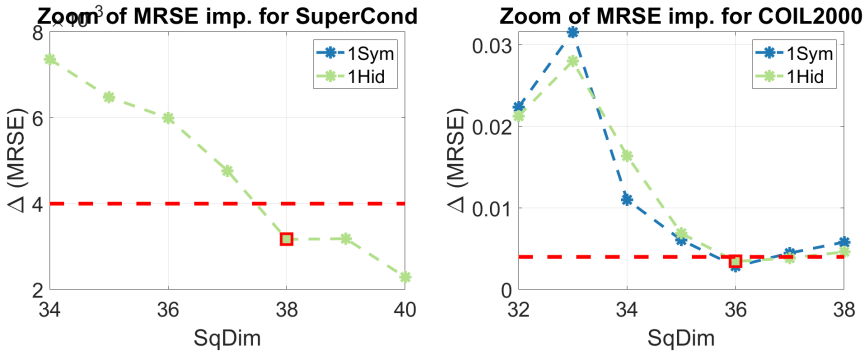

Figure 12: Zooms of the identification of the intrinsic dimension for SuperCond (left) and COIL2000 (right).

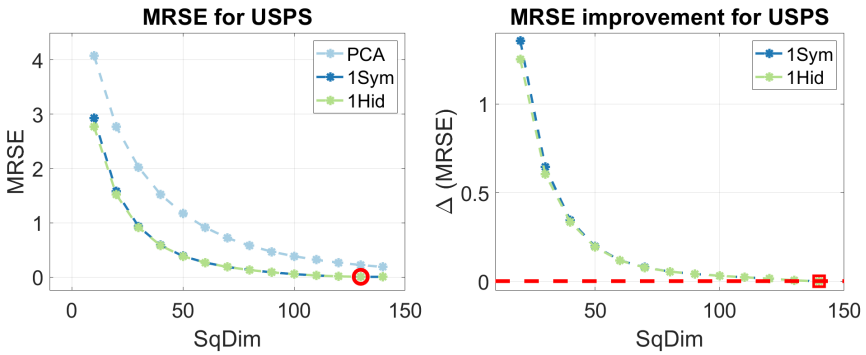

Figure 13: Identification of the intrinsic dimension for USPS (τ = 3e-3).

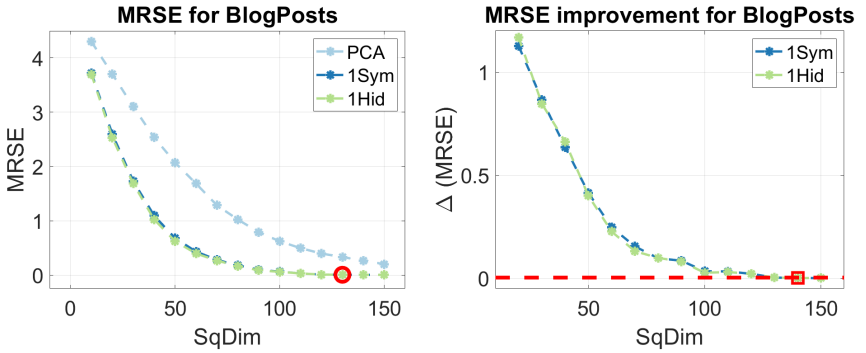

Figure 14: Identification of the intrinsic dimension for BlogPosts (τ = 3e-3).

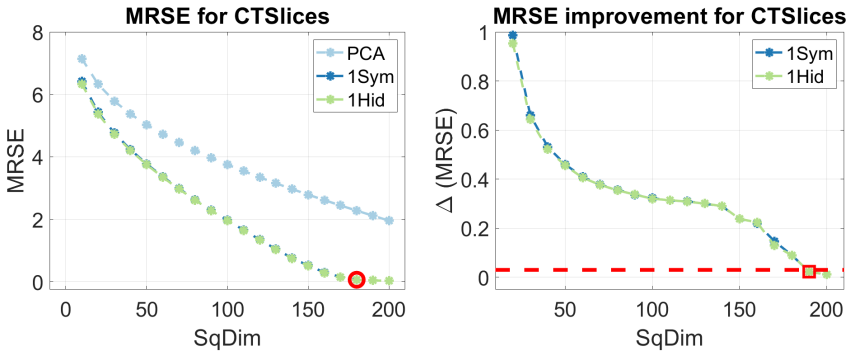

Figure 15: Identification of the intrinsic dimension for CTSlices (τ = 3e-2).

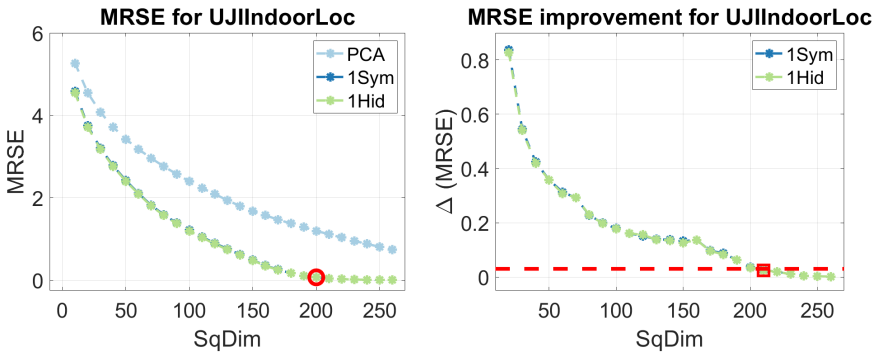

Figure 16: Identification of the intrinsic dimension for UJIIndoor (τ = 3e-2).

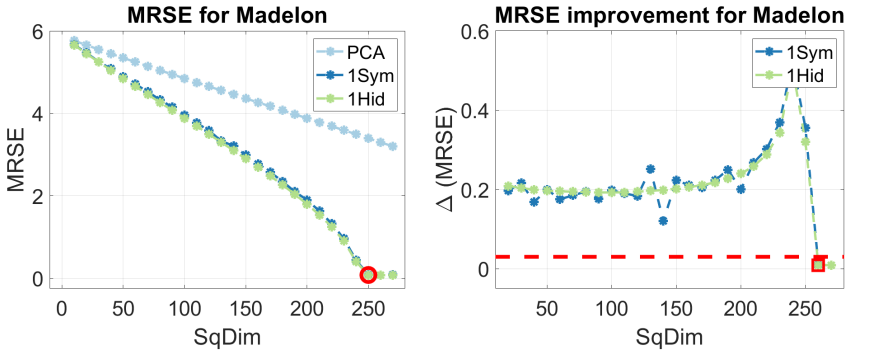

Figure 17: Identification of the intrinsic dimension for Madelon (τ = 3e-2).

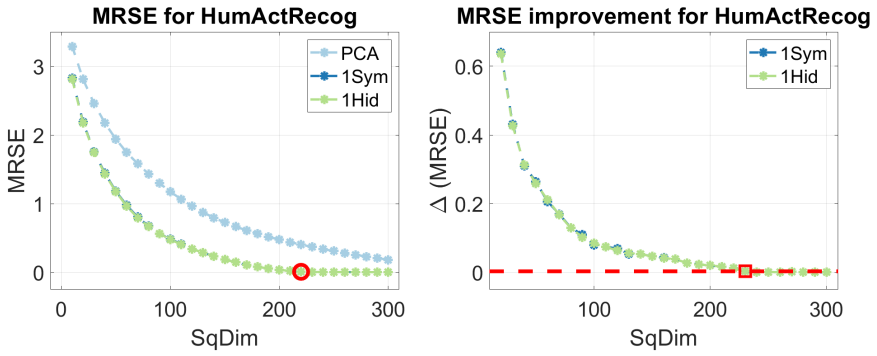

Figure 18: Identification of the intrinsic dimension for HumActRecog (τ = 3e-3).

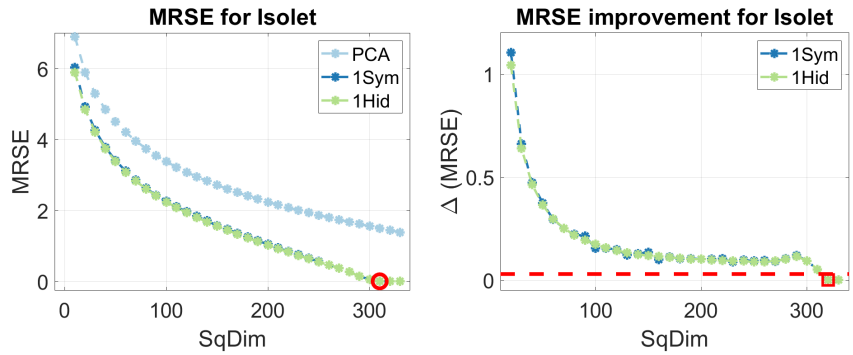

Figure 19: Identification of the intrinsic dimension for Isolet (τ = 3e-2).

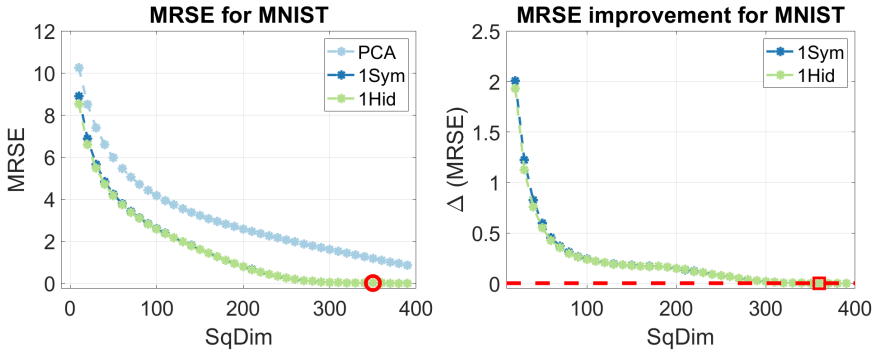

Figure 20: Identification of the intrinsic dimension for MNIST (τ = 3e-3).

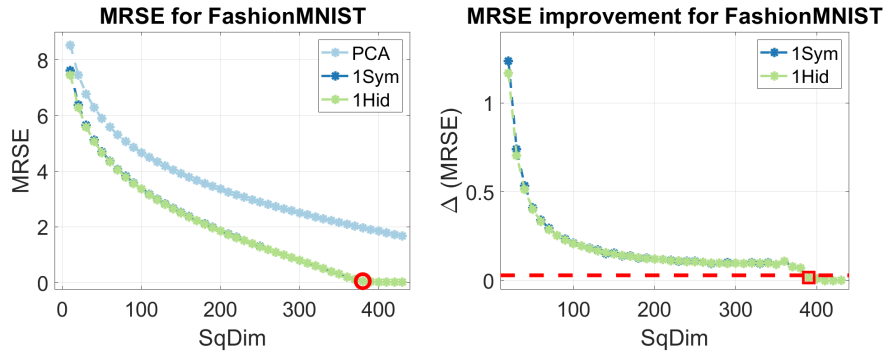

Figure 21: Identification of the intrinsic dimension for FashMNIST (τ = 3e-2).

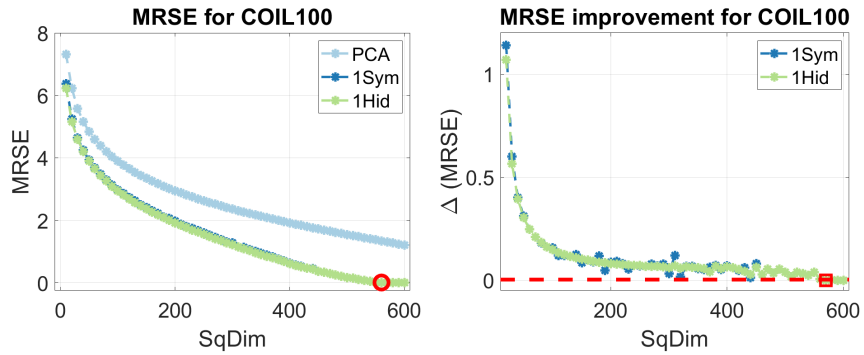

Figure 22: Identification of the intrinsic dimension for COIL100 (τ = 3e-3).

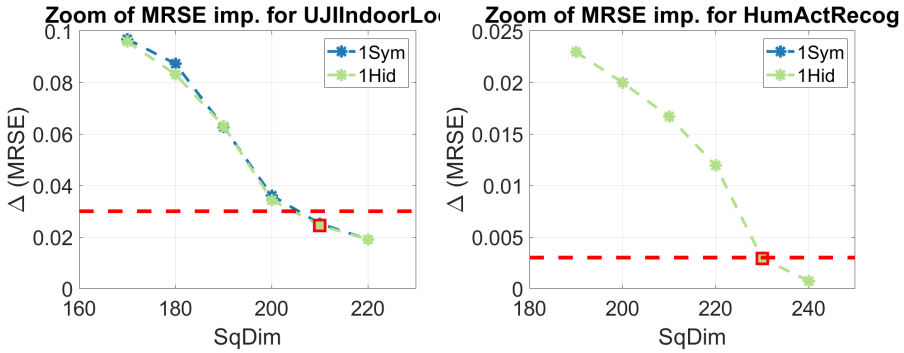

Figure 23: Zooms of the identification for UJIIndoor (left) and HumActRecog (right).

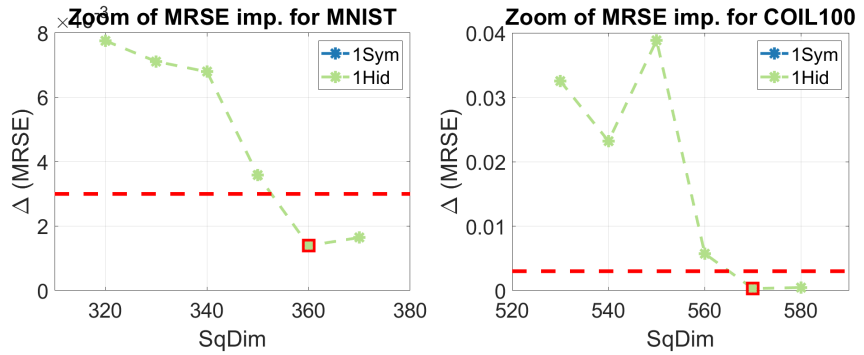

Figure 24: Zooms of the identification for MNIST (left) and COIL100 (right).

The identified dimensions and other conclusions based on this information are given in the main body of the paper. Below, we present a brief summary of the results above:

- The intrinsic dimensions were identified for all tested datasets. In most cases, there is a clear stabilization of the MRSE behavior, which is detected by the threshold technique. For some datasets (SuperCond, USPS, BlogPosts, HumActRecog, MNIST), the basic MRSE and its improvement plots are not self-evident by means of the intrinsic dimension, but the zooming confirms the successful detection. However, from Figs. 22 and 24 (right), it is evident that the selection of τ and search of ID for COIL100 should have been performed more rigorously so that for this largest dimensional data, the identification of the intrinsic dimension remained questionable.

- Use of a feedforward network to approximate the nonlinear residual clearly decreased the autoencoding error compared to the linear PCA. Usually, the 1HID's gain over the PCA continued to increase until the intrinsic dimension was found.

4 Comparison of Shallow and Deep Models

The following results compare the autoencoding errors between different nonlinear residual models as a function of the squeezing dimension $n_{\tilde{L}}$. Layerwise pretraining with the L-BFGS optimizer and complete training data for the deeper models 3SYM, 5SYM, and 7SYM used the same optimization settings (specified in the beginning of this document) as with 1SYM. In finetuning, we used MxIts = 2000 and ε_2 = 1.e-6, which in practice enforced MxIts iterations to be taken. The relative function value stopping criterion of Poblano was completely inactivated in fine-tuning.

In addition to visualizing the MRSEs for all residual models (left figures below), we compared relative performances of the deep and shallow models quantitatively. This was realized by dividing the MRSE values of all models with the corresponding value of the 1HID model's error for squeezing dimensions from the first until next to last of ID. These are illustrated in the figures on the righthand side below. For example, if the MRSE value of a model divided by the 1HID's value for a particular squeezing dimension would be 0.5, then such a model would have half the error level and, conversely, twice the efficiency compared to 1HID. Therefore, the model's efficiency is defined as the reciprocal of relative performance.

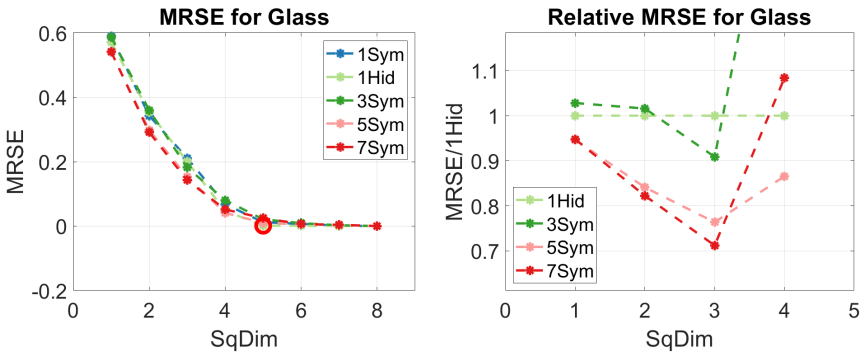

Figure 25: Relative performances for Glass.

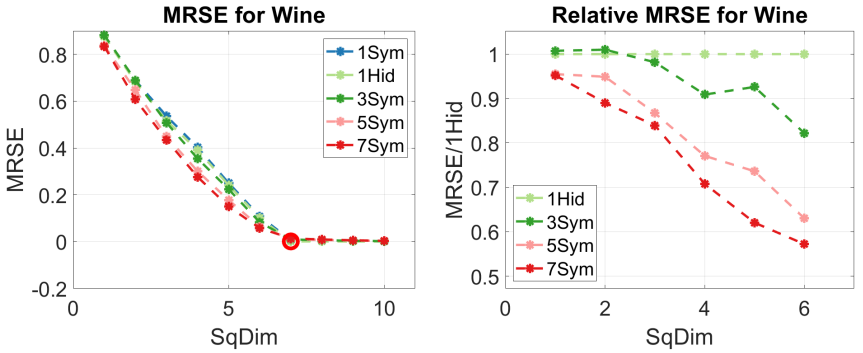

Figure 26: Relative performances for Wine.

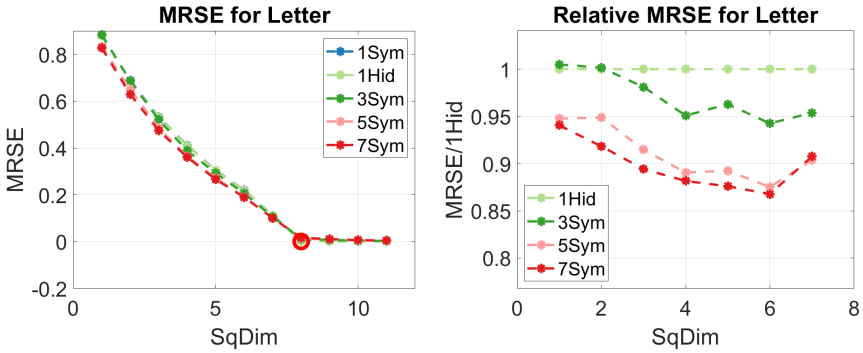

Figure 27: Relative performances for Letter.

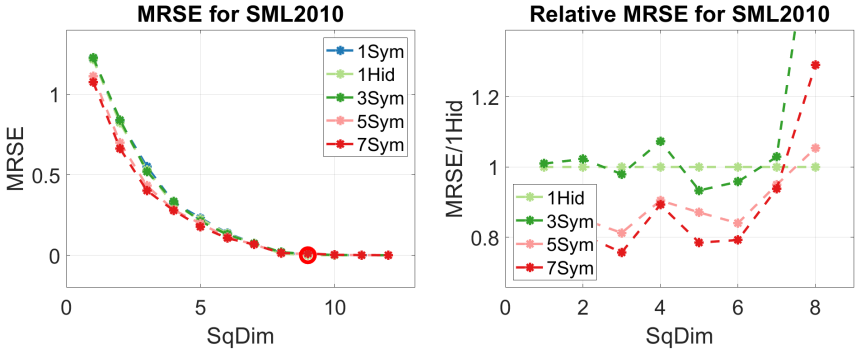

Figure 28: Relative performances for SML2010.

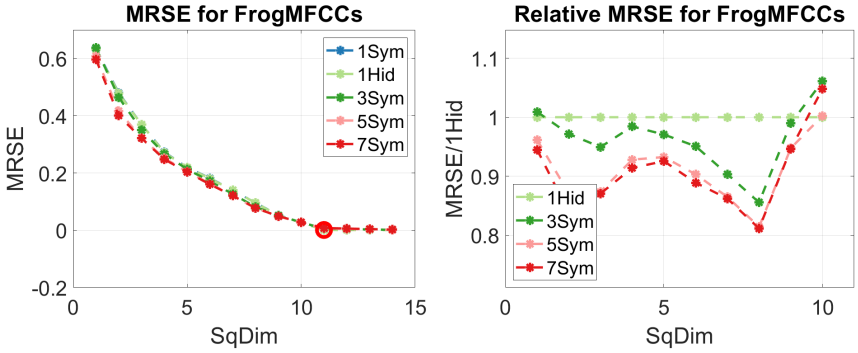

Figure 29: Relative performances for FrogMFCCs.

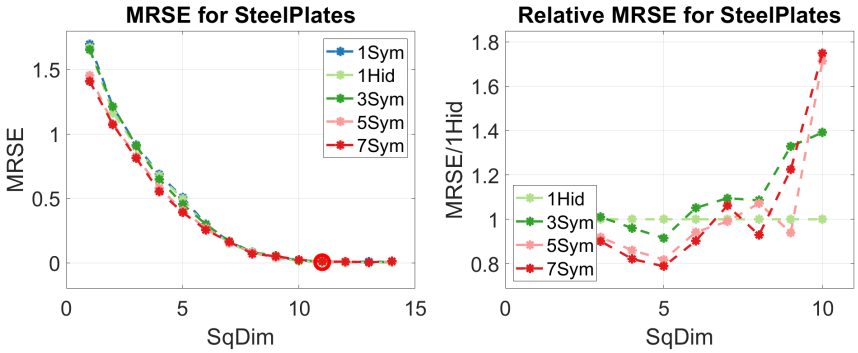

Figure 30: Relative performances for SteelPlates.

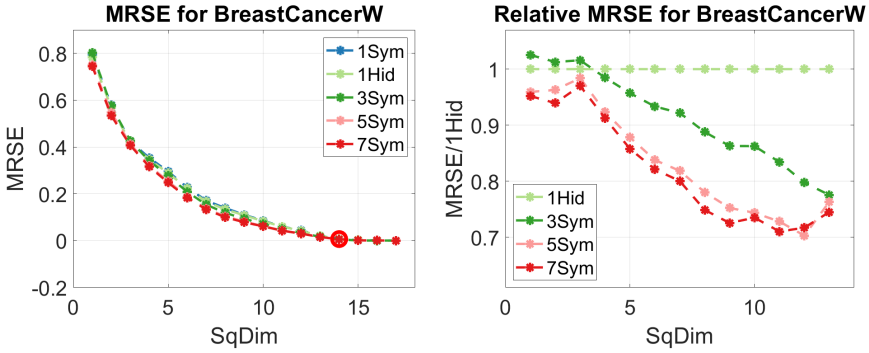

Figure 31: Relative performances for BreastCancerW.

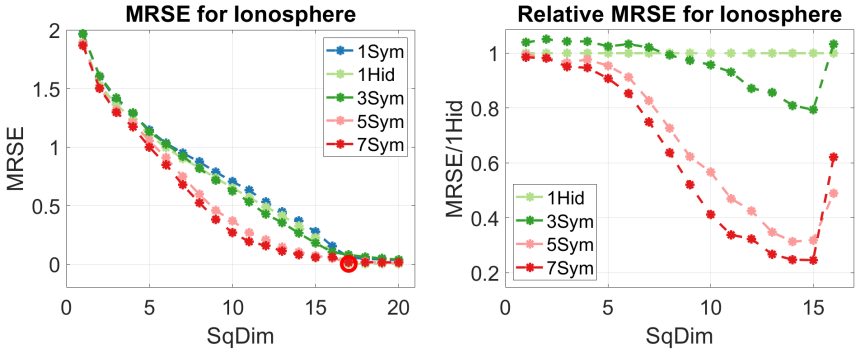

Figure 32: Relative performances for Ionosphere.

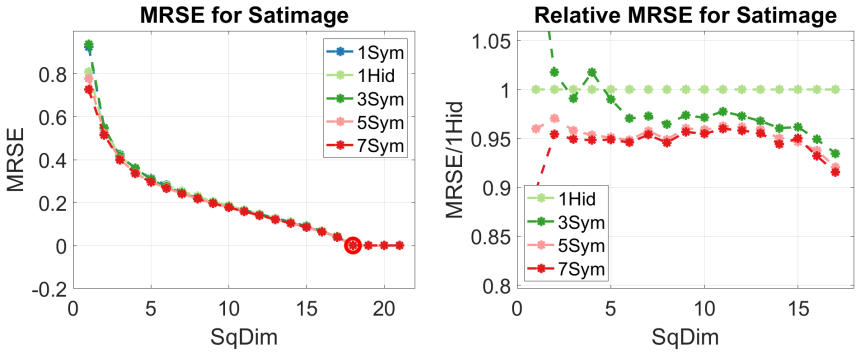

Figure 33: Relative performances for Satimage.

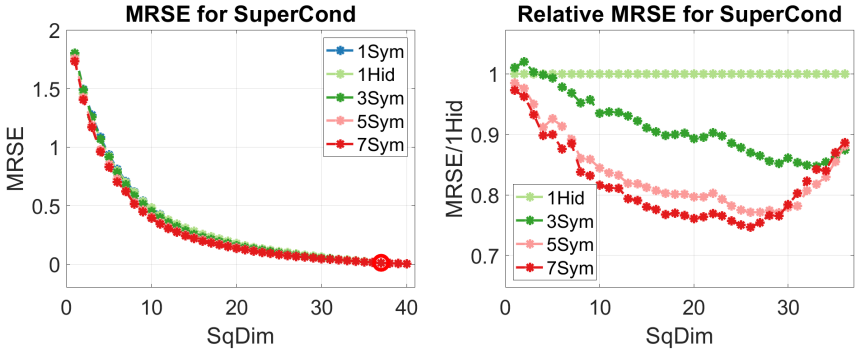

Figure 34: Relative performances for SuperCond.

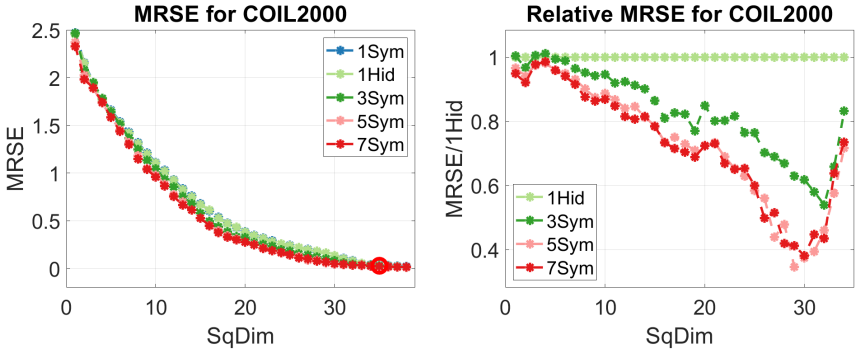

Figure 35: Relative performances for COIL2000.

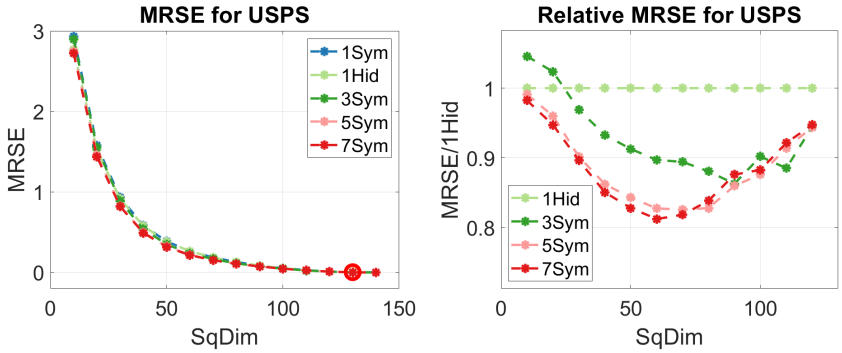

Figure 36: Relative performances for USPS.

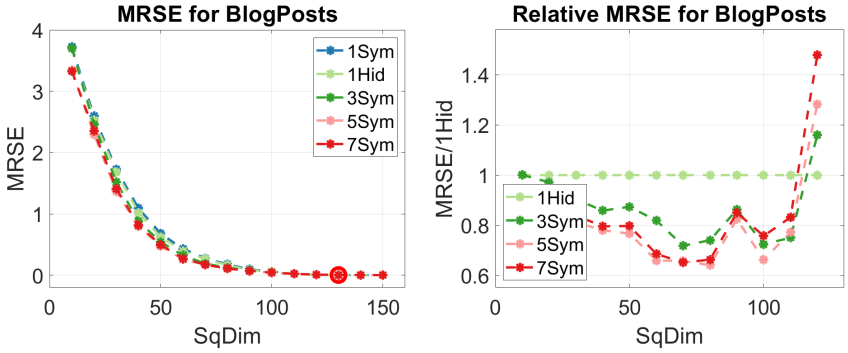

Figure 37: Relative performances for BlogPosts.

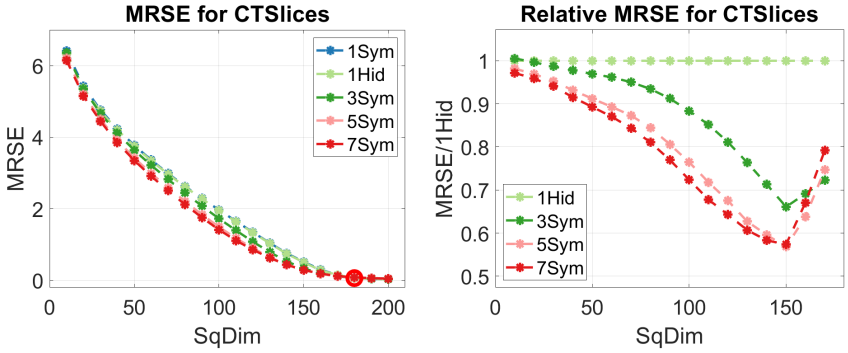

Figure 38: Relative performances for CTSlices.

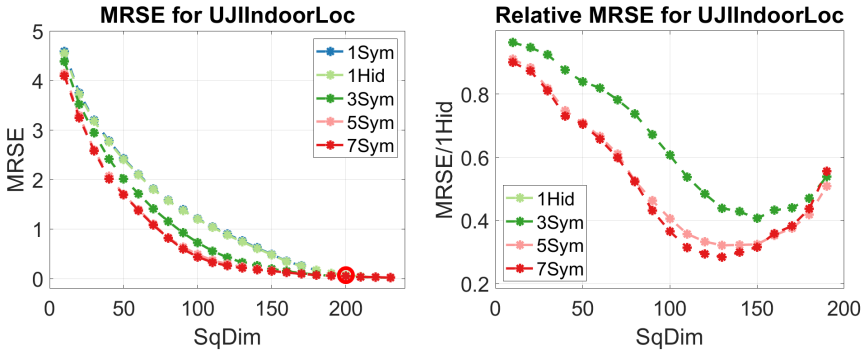

Figure 39: Relative performances for UJIIndoor.

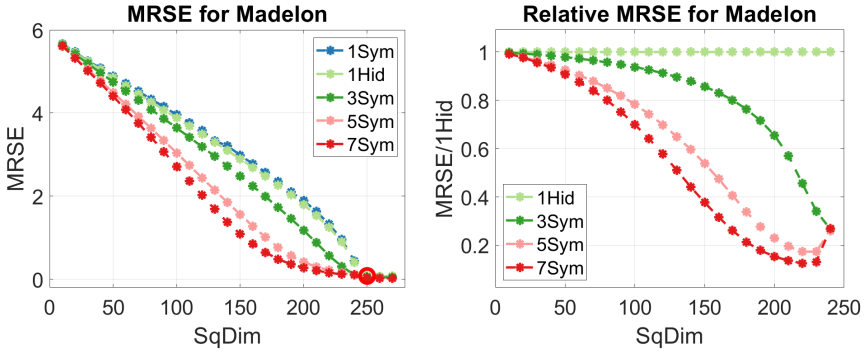

Figure 40: Relative performances for Madelon.

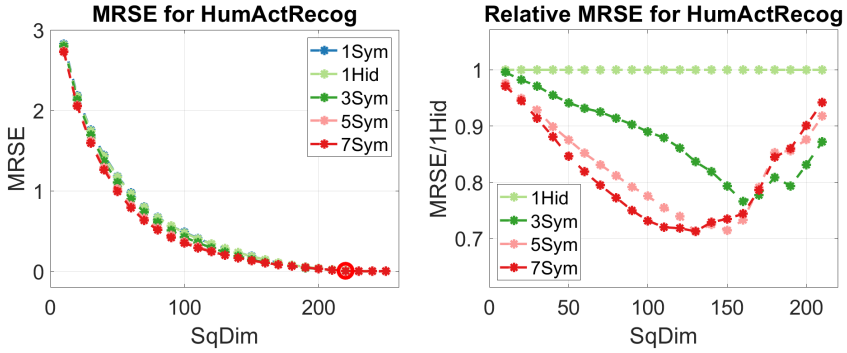

Figure 41: Relative performances for HumActRecog.

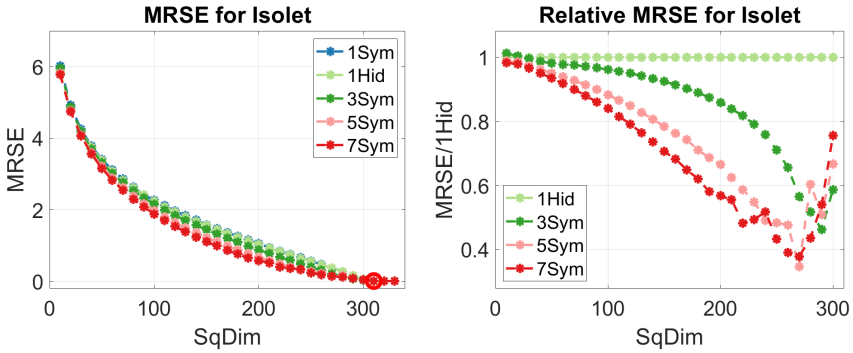

Figure 42: Relative performances for Isolet.

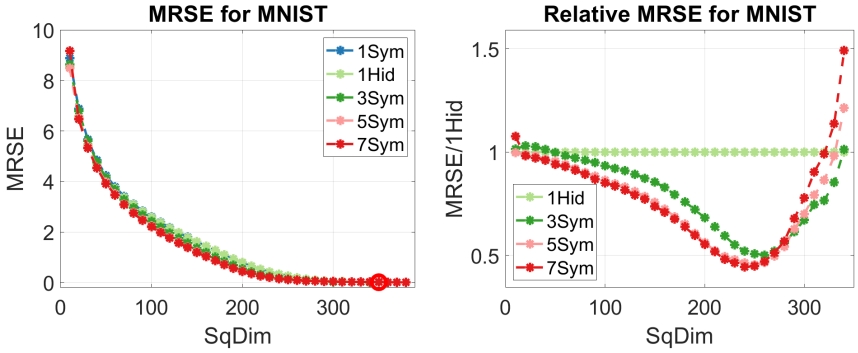

Figure 43: Relative performances for MNIST.

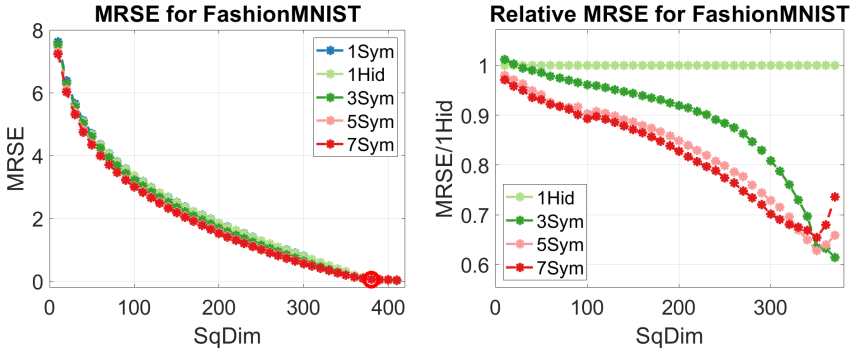

Figure 44: Relative performances for FashMNIST.

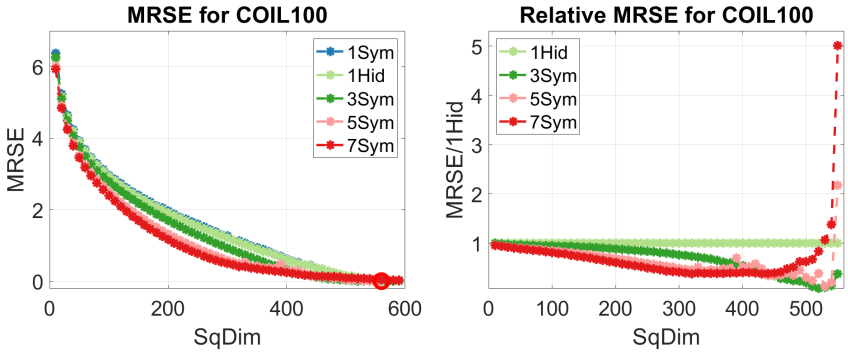

Figure 45: Relative performances for COIL100.

Descriptive statistics of the efficiencies of symmetric models are given in Tables 1 and 2. In each cell of the tables, both the mean efficiency and the maximal efficiency are given. The latter includes, in parentheses, the squeezing dimension where it was encountered.

Table 1: Efficiencies of symmetric models for small-dimension datasets.

Dataset	1Sym mean	1Sym max (dim)	3Sym mean	3Sym max (dim)	5Sym mean	5Sym max (dim)	7Sym mean	7Sym max (dim)
Glass	0.91	1.03 (2)	0.91	1.10 (3)	1.18	1.31 (3)	1.15	1.40 (3)
Wine	0.97	0.99 (1)	1.07	1.22 (6)	1.25	1.59 (6)	1.36	1.75 (6)
Letter	1.00	1.00 (1)	1.03	1.06 (6)	1.10	1.14 (6)	1.11	1.15 (6)
SML2010	0.94	0.99 (1)	0.95	1.07 (5)	1.12	1.23 (3)	1.15	1.32 (3)
FrogMFCCs	0.99	1.00 (7)	1.04	1.17 (8)	1.10	1.23 (8)	1.11	1.23 (8)
SteelPlates	0.94	0.99 (4)	0.94	1.09 (5)	1.04	1.22 (5)	1.04	1.27 (5)
BreastCancerW	0.98	0.99 (11)	1.10	1.29 (13)	1.22	1.42 (12)	1.24	1.41 (11)
Ionosphere	0.91	0.97 (3)	1.04	1.26 (15)	1.74	3.19 (14)	2.07	4.06 (15)
Satimage	0.99	1.00 (10)	1.02	1.07 (17)	1.05	1.09 (17)	1.06	1.12 (1)
SuperCond	1.00	1.00 (30)	1.10	1.18 (33)	1.20	1.30 (29)	1.23	1.34 (26)
COIL2000	0.99	1.02 (16)	1.24	1.85 (32)	1.49	2.89 (29)	1.48	2.62 (30)

Table 2: Efficiencies of symmetric models for large-dimension datasets.

Dataset	1Sym mean	1Sym max (dim)	3Sym mean	3Sym max (dim)	5Sym mean	5Sym max (dim)	7Sym mean	7Sym max (dim)
USPS	0.99	1.00 (90)	1.08	1.16 (90)	1.13	1.21 (70)	1.14	1.23 (60)
BlogPosts	0.95	1.00 (110)	1.18	1.39 (70)	1.28	1.56 (80)	1.23	1.53 (70)
CTSlices	0.99	1.00 (170)	1.17	1.51 (150)	1.30	1.76 (150)	1.32	1.74 (150)
UJIIndoor	0.99	0.99 (60)	1.69	2.46 (150)	2.15	3.11 (130)	2.24	3.51 (130)
Madelon	0.97	1.00 (30)	1.38	3.74 (240)	2.29	5.77 (220)	3.01	8.02 (220)
HumActRec	0.99	1.00 (160)	1.15	1.31 (160)	1.22	1.40 (130)	1.24	1.40 (130)
Isolet	0.99	1.00 (290)	1.22	2.16 (290)	1.44	2.89 (270)	1.55	2.65 (270)
MNIST	0.99	1.00 (200)	1.32	1.99 (260)	1.42	2.20 (250)	1.41	2.25 (240)
FashMNIST	0.99	1.00 (320)	1.15	1.63 (370)	1.22	1.59 (350)	1.23	1.53 (350)
COIL100	0.98	1.00 (310)	2.18	11.19 (520)	1.96	8.51 (530)	1.79	2.63 (430)
COIL100-Min	0.98	1.00 (310)	1.75	8.05 (510)	1.79	4.22 (510)	1.87	2.63 (430)

Conclusions:

- During the early phases of searching the intrinsic dimension, deep networks provide smaller autoecoding errors compared to the shallow models. Usually, the mean efficiencies of the two deepest models 5Sym and 7Sym is better than that of 1Sym or 3Sym, but the quantitative difference varies; for many datasets, there is only a slight improvement. The mean efficiency is highest for UJIIndoor, Ionosphere, Madelon, and COIL100, where the last three datasets are characterized by high data dimension/number of observations, n/N, ratio (0.09, 0.11, and 0.08, respectively).

- Close to the intrinsic dimension, the benefits of deeper models are usually lost. This is illustrated in the left figures above but is even more clearly visible in the right figures, where the relative error plots of the deeper models are usually united with the shallow results for the last values.

- The loss of the improved efficiency of the deeper models is particularly visible in Fig. 45 (right) and in Table 2 for COIL100. The reason for such a behavior with COIL100 is the value of ID, 560, which was obtained with the smaller value of the threshold τ = 3e-3 of large-dimension datasets (see Fig. 24, right). Therefore, with COIL100, we also tested an alternative approach to the identification of ID, where we apply the same thresholding technique to the minimum autoencoding error of all models. This error plot and its absolute speed of change are depicted in Fig. 46, with the resulting ID 520. Zoom of the identification decision and the corresponding reduced (only up to 510) plot of relative efficiencies are contained in Fig. 47. The summary of the efficiencies for this modified way to identify ID are given in the last line "COIL100-Min" in Table 2. It can be concluded that for this largest dimensional dataset, the use of the minimum autoencoding error of the models yielded more reasonable results.

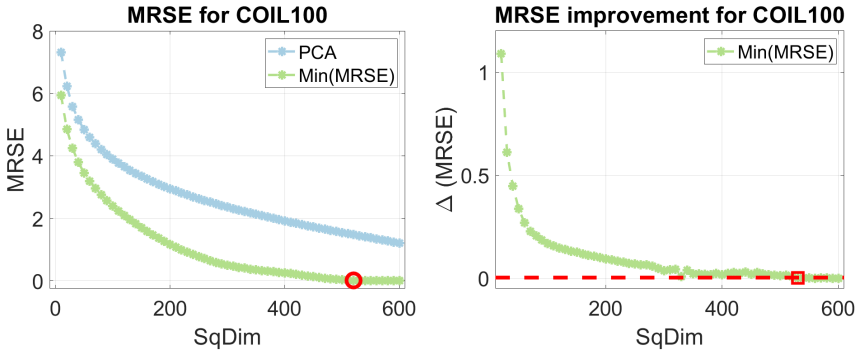

Figure 46: Identification of the ID for COIL100 from the minimum AE error.

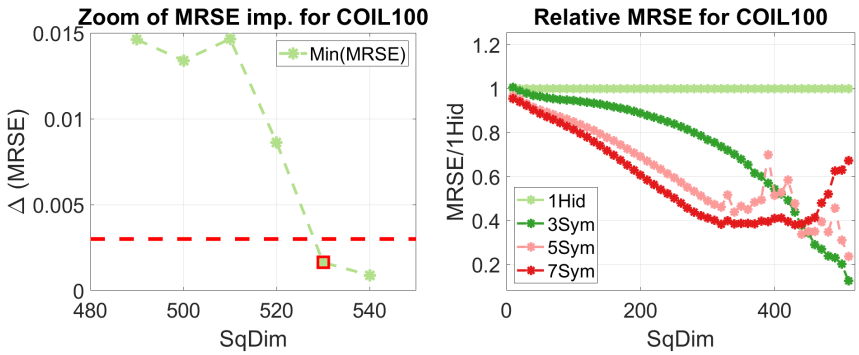

Figure 47: Zoomed identification of ID (left) and Relative performances (right) for COIL100 from the minimum AE error.

5 Role of Minibatching

Next, we consider the effect of using distribution optimal (DOP; see Section 2) minibatching during optimization. This is studied in two parts, by first providing similar depictions as in the previous section and then presenting an individual case study.

5.1 Efficiency of deeper models with minibatching

As demonstrated in the previous sections (cf. all figures so far and Tables 1 and 2), the qualitative behavior of the results is similar between small- and large-dimension data. Therefore, we consider only small-dimension datasets in these tests. The optimization settings follow the following specifications: in addition to the same fixed settings as above for 1Sym pretrainer (used for 1Hidand in stacking of all deeper models) and 1Hid, we apply Adam-DOP8 (with eight DOP minibatches), with ε_a = 1e-8 and MxIts = 15 000 or 20 000. More precisely, for datasets with several thousands of observations, the minibatch based approximate cost function and gradient determination is computationally particularly advantageous. Therefore, for Letter, FrogM-FCCs, Satimage, SuperCond, and COIL2000, roughly similar CPU time for Adam-DOP8 compared to the L-BFGS tests reported above was obtained by using a maximum of 20 000 iterations, while for other datasets this was obtained by using a maximum of 15 000 iterations.

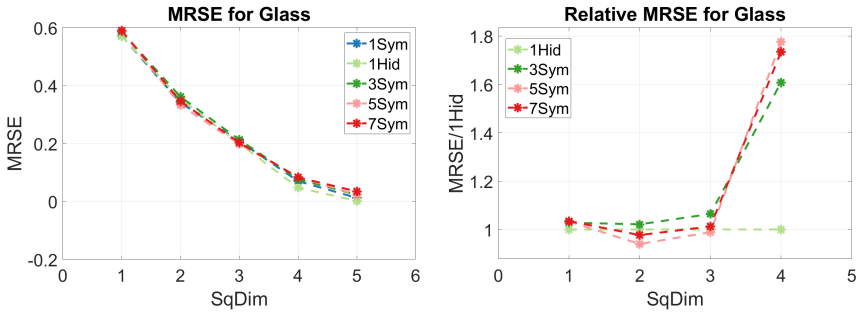

Figure 48: Comparison of all models for Glass.

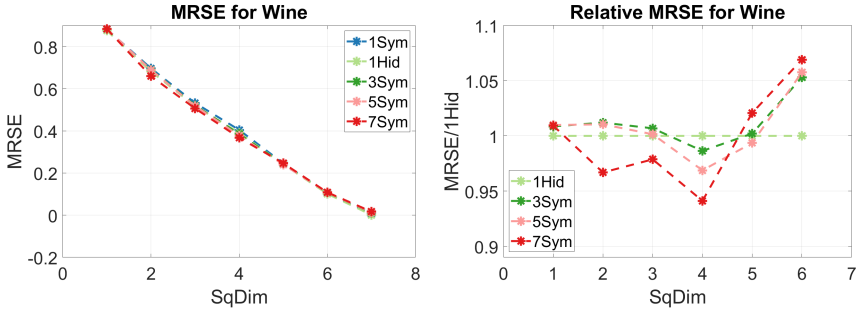

Figure 49: Comparison of all models for Wine.

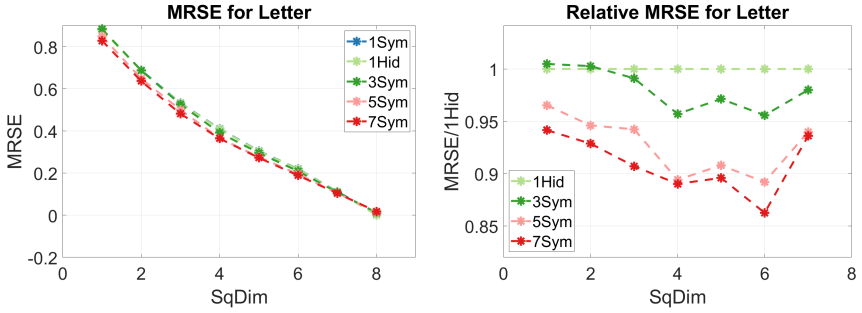

Figure 50: Comparison of all models for Letter.

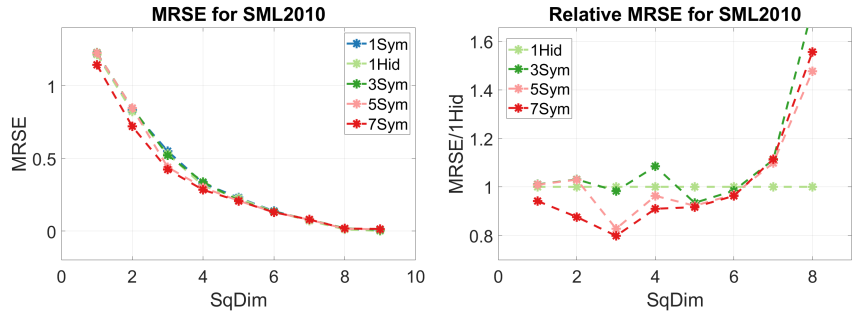

Figure 51: Comparison of all models for SML2010.

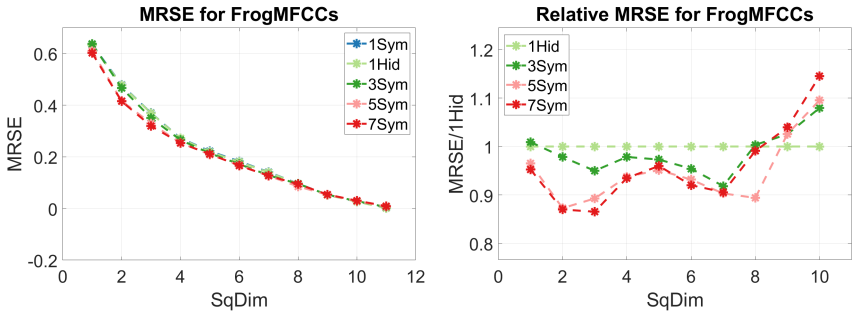

Figure 52: Comparison of all models for FrogMFCCs.

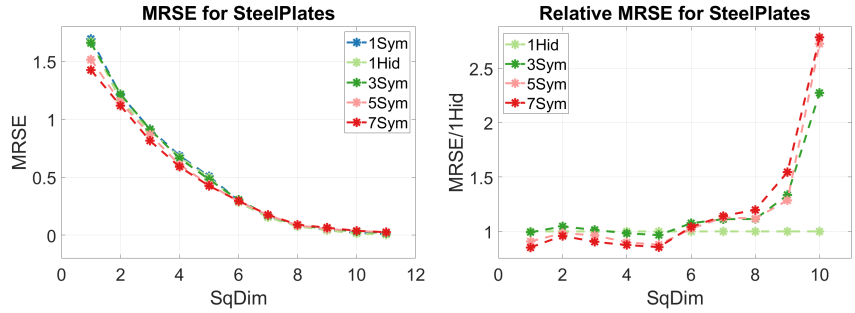

Figure 53: Comparison of all models for SteelPlates.

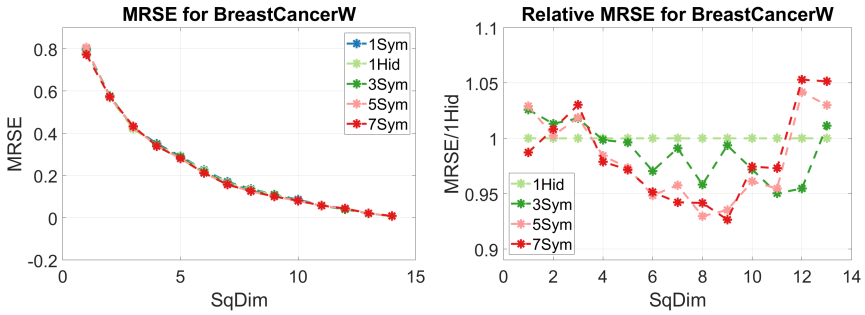

Figure 54: Comparison of all models for BreastCancerW.

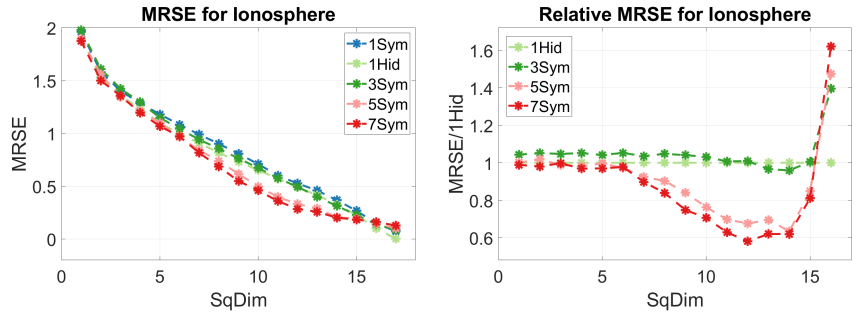

Figure 55: Comparison of all models for Ionosphere.

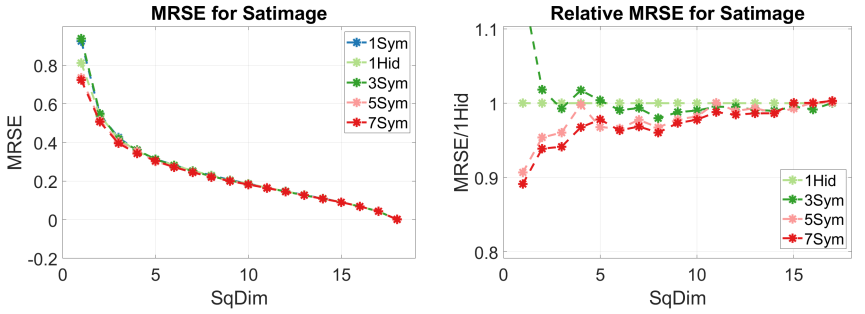

Figure 56: Comparison of all models for Satimage.

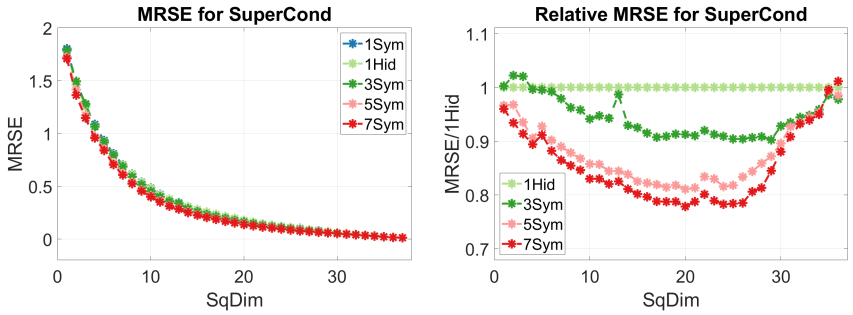

Figure 57: Comparison of all models for SuperCond.

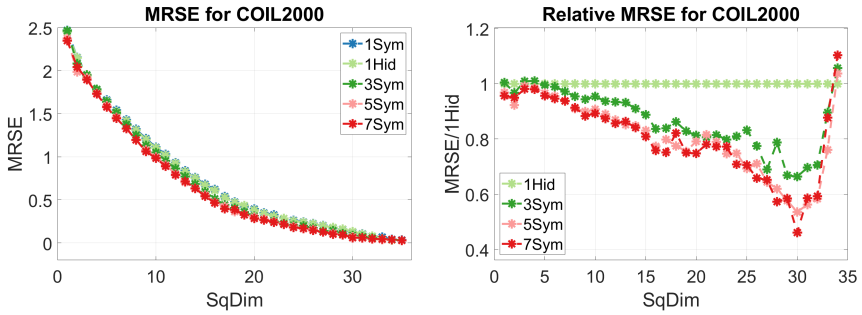

Figure 58: Comparison of all models for COIL2000.

In Table 3, the same efficiency summary as in Table 1 for the L-BFGS fine-tuner, is given for Adam-DOP8. For a quantitative comparison between the results of the two fine-tuners, we include the quotient *nSym* (Adam-DOP8)/nSym* (L-BFGS)* for 3SYM, 5SYM, and 7SYMin the mean columns in parentheses. If this quotient is less than one, then the efficiency of the Adam-DOP8 fine-tuner is the corresponding fraction of that of the L-BFGS with complete data.

Conclusion: The use of minibatches in fine-tuning leads to a clear loss of efficiency of the deeper models compared to the complete data fine-tuning results of the previous section. The amount of deterioration varies: For Letter, the results with Adam-DOP8 are almost the same as those with L-BFGS but there is a large discrepancy for Ionosphere. These extreme cases were obtained for datasets with minimum and maximum ratio of n/N (0.0008 vs. 0.09). The following were the means for the performance ratios between Adam-DOP8 and L-BFGS given in parentheses in the mean columns of Table 3: 3SYM 0.96, 5SYM 0.87, and 7SYM 0.86. This confirms that deeper models have more challenges with the minibatch-based fine-tuning, but the hindrance for the two deepest models remains at the same level.

Table 3: Efficiency of the symmetric models with Adam-DOP8 for small-dimension datasets.

	1Sym		3Sym		5Sym		7Sym	
Dataset	mean	max	mean	max	mean	max	mean	max
Glass	0.92	1.04 (2)	0.88 (0.97)	0.98 (2)	0.90 (0.76)	1.06 (2)	0.89 (0.77)	1.02 (2)
Wine	0.98	0.99 (1)	0.99 (0.93)	1.01 (4)	0.99 (0.79)	1.03 (4)	1.00 (0.74)	1.06 (4)
Letter	1.00	1.00 (1)	1.02 (0.99)	1.05 (6)	1.08 (0.98)	1.12 (6)	1.10 (0.99)	1.16 (6)
SML2010	0.94	0.99 (1)	0.93 (0.98)	1.07 (5)	0.99 (0.88)	1.21 (3)	1.03 (0.90)	1.25 (3)
FrogMFCCs	0.99	1.00 (3)	1.02 (0.98)	1.09 (7)	1.06 (0.96)	1.15 (2)	1.05 (0.95)	1.16 (3)
SteelPlates	0.93	0.99 (1)	0.89 (0.95)	1.04 (5)	0.93 (0.89)	1.14 (5)	0.93 (0.89)	1.17 (1)
BreastCancerW	0.98	1.00 (11)	1.01 (0.92)	1.05 (11)	1.02 (0.84)	1.08 (8)	1.02 (0.82)	1.08 (9)
Ionosphere	0.91	0.97 (3)	0.96 (0.92)	1.04 (14)	1.16 (0.67)	1.58 (14)	1.22 (0.59)	1.72 (12)
Satimage	0.99	1.00 (17)	1.00 (0.98)	1.02 (8)	1.02 (0.97)	1.10 (1)	1.03 (0.97)	1.12 (1)
SuperCond	1.00	1.00 (25)	1.06 (0.96)	1.11 (29)	1.15 (0.96)	1.23 (20)	1.18 (0.96)	1.28 (20)
COIL2000	0.98	1.06 (30)	1.17 (0.94)	1.51 (30)	1.28 (0.86)	1.87 (30)	1.29 (0.87)	2.17 (30)

5.2 On the convergence of optimizers

To this end, we compared different fine-tuners with the USPS dataset. The squeezing dimension wax fixed into $n_{\tilde{L}} = 60$, which according to Table 2 yielded the maximum efficiency for 7Sym. The pretraining was performed using L-BFGS-DOP2. In fine-tuning, always using the same weights after pretraining, we compared four scenarios: *i*) L-BFGS with complete data, *ii*) Adam with complete data, *iii*) L-BFGS-DOP2, and *iv*) Adam-DOP2. The optimization settings, fully specified in the SM, were selected in such a manner that, first, the L-BFGS and Adam with complete data utilized approximately the same CPU time. For DOP2 versions, we then simply selected twice the number of maximum iterations to ensure, actually, a slightly larger CPU time for the two minibatch versions.

In the figures below, we illustrate the behavior of the cost function $\mathcal{J}(\{\mathbf{W}^l\})$ during the optimization. The x-axis in the plots includes percentages with respect to the maximum number of iterations. The first four plots in Figures 59 and 60 illustrate the behavior of L-BFGS-DOP2 pretrainer for 3Sym and 5Sym in the first figure and 7Sym in the second figure. Note that the latter figure also contains the zoom of the normalized (divided by $\mathcal{J}^0$) cost function behavior. The remaining figures below illustrate the cost function during fine-tuning with all three models and four fine-tuners. A quantitative summary of the results is provided after the figures in Table 4.

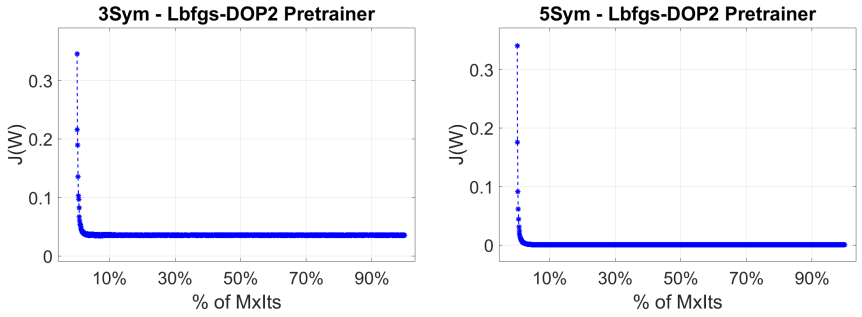

Figure 59: 3SYM and 5SYM pretraining results for L-BFGS-DOP2.

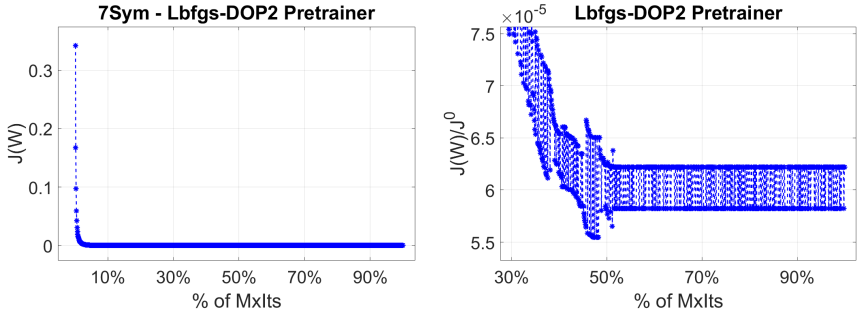

Figure 60: 7SYM pretraining results with L-BFGS-DOP2.

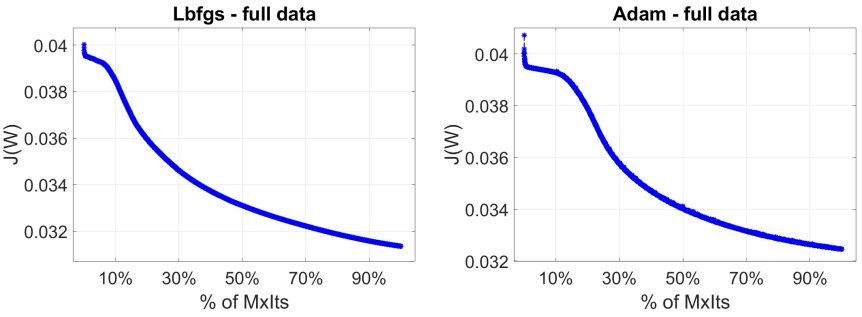

Figure 61: 3SYM results for L-BFGS and Adam finetuners with entire data.

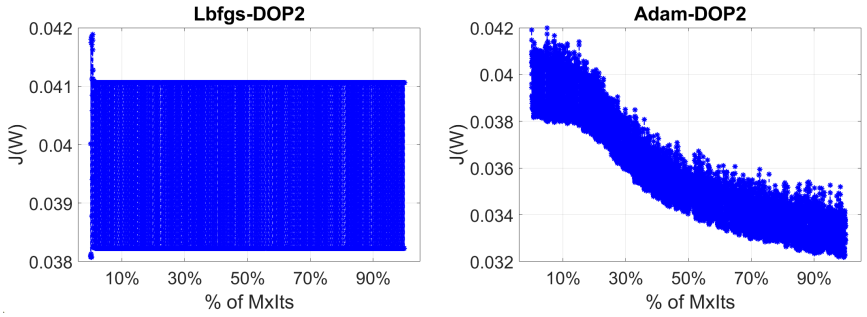

Figure 62: 3SYM results for L-BFGS-DOP2 and Adam-DOP2 finetuners.

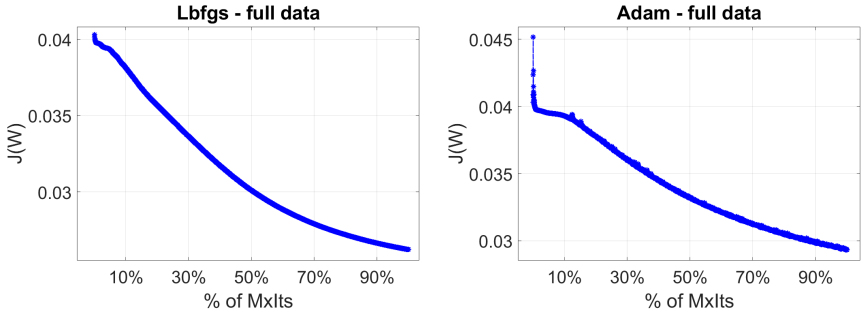

Figure 63: 5SYM results for L-BFGS and Adam finetuners with entire data.

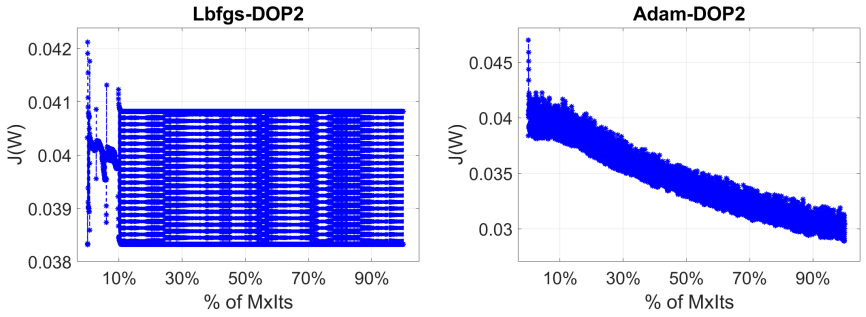

Figure 64: 5SYM results for L-BFGS-DOP2 and Adam-DOP2 finetuners.

Table 4: Summary of finetuning results with different optimizers. The best results for different models are emphasized with the bold font.

Model	$\mathcal{J}^0$	$\mathcal{J}^*$	AEErr	CPU	$\mathcal{J}^*$	AEErr	CPU
		L-BFGS - whole data			Adam - whole data		
3SYM	4.001e-2	**3.135e-2**	**2.37e-1**	210.2	3.246e-2	2.41e-1	**206.2**
5SYM	4.032e-2	**2.622e-2**	**2.18e-1**	**425.7**	2.934e-2	2.31e-1	426.1
7SYM	4.037e-2	**2.553e-2**	**2.16e-1**	577.5	2.872e-2	2.29e-1	**574.0**
		L-BFGS-DOP2			Adam-DOP2		
3SYM	4.001e-2	3.996e-2	2.65e-1	244.7	3.315e-2	2.44e-1	246.7
5SYM	4.032e-2	4.021e-2	2.66e-1	511.9	3.006e-2	2.33e-1	490.0
7SYM	4.037e-2	4.021e-2	2.67e-1	654.1	2.894e-2	2.29e-1	649.9

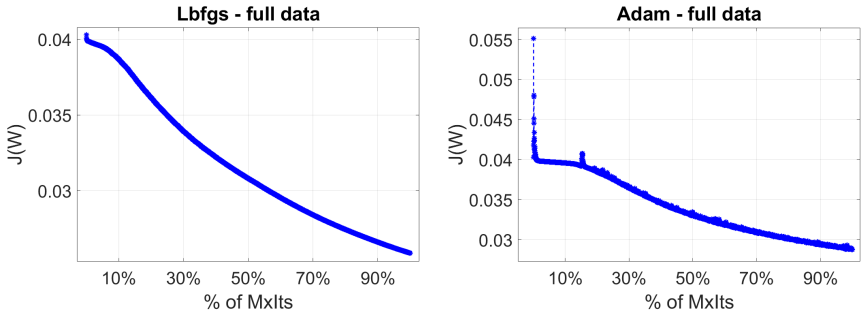

Figure 65: 7SYM results for L-BFGS and Adam finetuners with entire data.

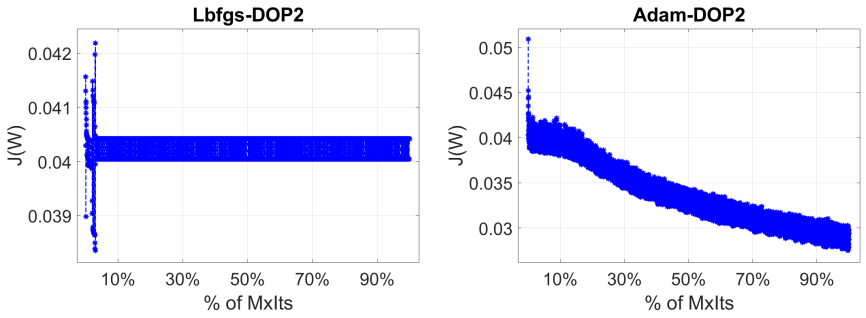

Figure 66: 7SYM results for L-BFGS-DOP2 and Adam-DOP2 finetuners.

Conclusions:

- Let us first comment the pretraining figures: It appears from Fig. 60 (left) that L-BFGS-DOP2 quickly does a good job and could be terminated before 10% of the iteration budget. However, as the zoom

on the right demonstrates, the cost function value actually continues to decrease during the first half (over 50% of iteration budget) of the search. This plot indicates that the normalized cost function value is zigzagging but, as can be seen from the y-axis, this happens on the 1e-5 scale. To conclude, during pretraining when optimizing a network with only one hidden layer, the L-BFGS-DOP2 performs adequately.

- From Figs. 65 and 66 and from Table 4 we can conclude that the L-BFGS optimizer with the complete dataset was the best fine-tuning method: It was the most accurate in optimization and autoencoding, and was either fastest or at most 2% slower than the Adam (in 3Sym). All values of AEErr in the table illustrate the tight coupling of the quality of optimization and autoencoding.

- The complete data fine-tuners worked better than the DOP2-based versions. Adam with two minibatches had clearly better convergence than the approximate second-order method L-BFGS-DOP2, which showed a very bad performance. In all these tests, it made progress in the early search phase but was then stuck to jump between two non-optimal values. L-BFGS-DOP2 could have been terminated earlier by using a similar stopping criterion as that of Adam. Moreover, Adam-DOP2 showed a zigzag behavior, but with a decreasing overall trend.

- The inferior behavior of the L-BFGS-DOP2 here and Adam-DOP8 in the previous section combined with the decent overall autoencoding results demonstrate the usefulness of stacking: Even with highly inaccurate individual optimizers, we still obtain reasonable results—though by no means the best possible ones.

- Interestingly, when the depth of the network increases and we construct the layerwise initialization from heads to center, the initial value of the cost function $\mathcal{J}^0$ in Table 4 is also slightly increasing for the deeper models. This is one indication that the optimization problem to be solved becomes more complicated when the depth of the network increases. As explained in Section 5.1, we could also pretrain larger portions of the deeper networks than just the individual layers. This would take more CPU time before fine-tuning but could generate better initial configurations.

All observations above illustrate the importance of sufficient accuracy in solving the optimization problem for the nonlinear autoencoding part. For the deeper models, their superiority while searching the intrinsic dimension is appropriately revealed only by the use of the L-BFGS optimizer with the complete data and sufficiently accurate stopping criteria.

6 Assessing generalization

Next, we demonstrate the generalization of the additive autoencoder. Its use for unseen data, like the validation datasets below, was detailed in Remark 1 of the main paper. The same optimization settings as in Section 4 are used. We restrict ourselves to a small sample of the datasets, for which a separate validation data was given in the UCI repository. More precisely, we use Letter (size of training data $N = 16000$, size of validation data $N_v = 4000$—that is, 80%–20% portions of all data; number of nonconstant featureas $n = 16$), UJIIndoor ($N = 19937$, $N_v = 1111$, 95%-5% portions; $n = 473$), HumActRecog ($N = 7351$, $N_v = 2946$, 71%–29% portions; $n = 561$), and MNIST ($N = 60000$, $N_v = 10000$, 86%–14% portions; $n = 666$). Note that because all data are used as is, we have no information or guarantees on how well the data distributions in the training and validation sets actually match each other.

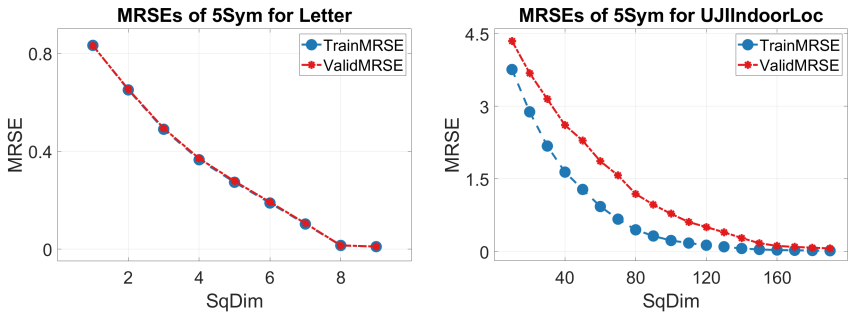

Figure 67: Generalization of Letter and UJIIndoor.

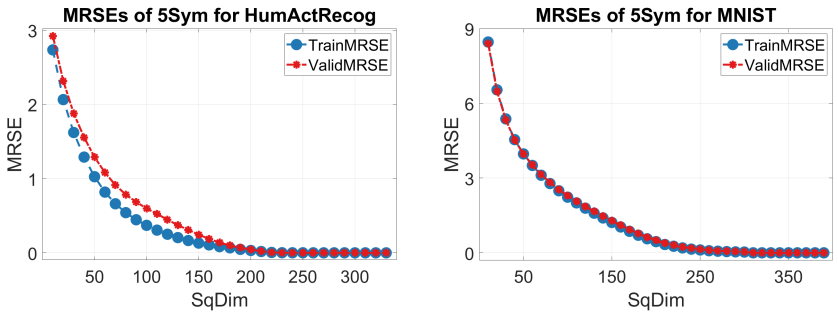

Figure 68: Generalization of HumActRecog and MNIST.

As anticipated, both training-validation portions and the data dimension affected the generalization results. For Letter, with 80%–20% division between training-validation sizes, small number of features, and minimal n/N ratio, we witnessed a perfect match between training and validation MRSE values in Fig. 67 (left). The same held true for MNIST in Fig. 68

(right). The largest discrepancy between the training and validation errors was obtained for UJIIndoor (Fig. 67 right), which has the most deviating 95%–5% portions with almost 500 features. HumActRec (Fig. 68 left) was somewhere in the middle in its behavior, with clearly visible deviation during the early search phase. Because of the data portions (∼70%–30%), this behavior raises doubts regarding the compatibility between the training and validation sets.

The visual assessment was augmented by computing the correlation coefficient between the MRSE values of the training and validation sets. The following values confirmed the conclusions of the visual inspection: Letter 1.0000, UJIIndoor 0.9766, HumActRecog 0.9939, and MNIST 0.9999. Finally, an important observation from the figures is that when the squeezing dimension is increased up to the intrinsic dimension, then also the validation error tends to the same error level as the training error. Therefore, the additive autoencoder determined using the training data explained the variability of the validation data with the same error level.

7 Assessing autoencoder's structure and implementation

To this end, we present a small comparison between the properties of different autoencoding model structures and their implementations. More precisely, the classical form of autoencoder would mean direct application of the feedforward neural network after data normalization. When we add a linear operator to the overall data transformation, as suggested in this work, we have basically two options on how to use an existing autoencoder for the residual estimation: we could apply this nonlinear transformation to the residual in the reduced dimensional space and in decoding apply the inverse of the linear operator to reconstruct data and assess error. In our proposition, reduced data from the linear transformation is first mapped back to the original dimension using the inverse of the linear operator and an autoencoder is then trained for encapsulating the unexplained residual in the original dimension.

To compare both how *i*) these two forms of integrating linear and nonlinear transformations behave and *ii*) how our reference implementations relate to a proprietary autoencoder, we consider our nonsymmetric one-hidden-layer 1Hid model against Matlab's own autoencoding method[3]. It has similar structure than our formulation in equation (4) of the main body but this routine also uses a sparsity regularizer based on Kullback-Leibler divergence to favor sparse output. Parameters of the Matlab routine are either default or aligned with our implementation: to use the same number of iterations (2000), linear activation on the outer layer, and the same value of the

[3] `https://se.mathworks.com/help/deeplearning/ref/trainautoencoder.html`

regularization coefficient $\alpha = 1e-6$ in 'L2WeightRegularization'. In these tests, all small-dimension datasets and USPS representing large-dimension datasets are used.

We assess accuracy (MRSE reconstruction error) between three different results: 1) '1HidRed(MatL)' refers to a case where Matlab's own autoencoding routine is applied in the reduced dimension after PCA and the reconstruction error is computed with the inverse PCA by multiplying with $\mathbf{U}^T$ from AE's output. 2) '1Hid(MatL)' refers to a case where Matlab's own autoencoding routine is applied according to our proposition to data obtained after linear trend estimation in the original data dimension, as defined in formula (2) in the main body of the paper. 3) '1Hid(Prop)' is our own implemented method. We remind that this method and its CPU time include pretraining of the 1SYM model. The error plots for these methods are given in left figures below.

On right figures below, we present the CPU time ratios for each value of the hidden dimension for '1Hid(MatL)' divided by '1Hid(Prop)'. Values over one would suggest that the proprietary implementation took more CPU time than ours.

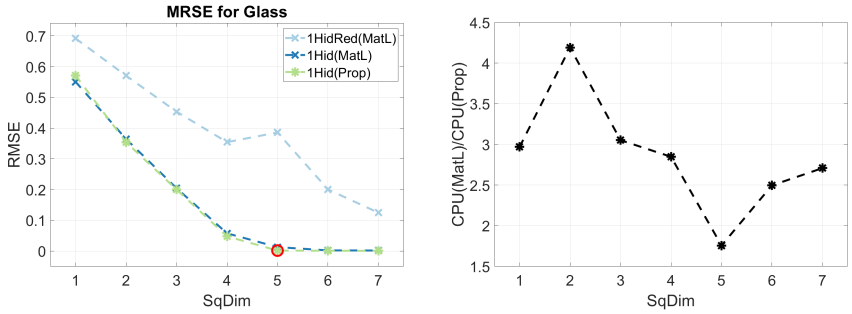

Figure 69: Comparison of forms and realizations of AE for Glass.

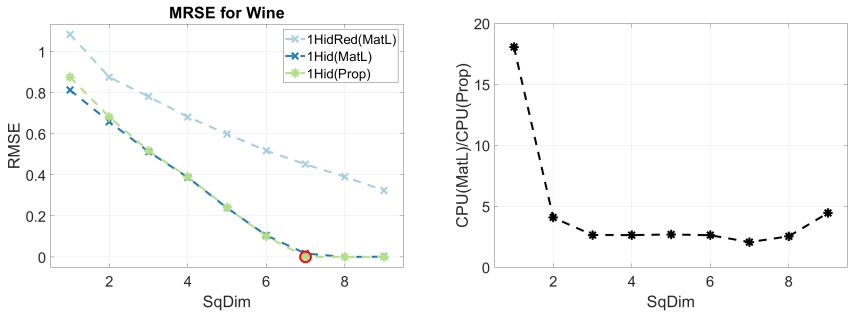

Figure 70: Comparison of forms and realizations of AE for Wine.

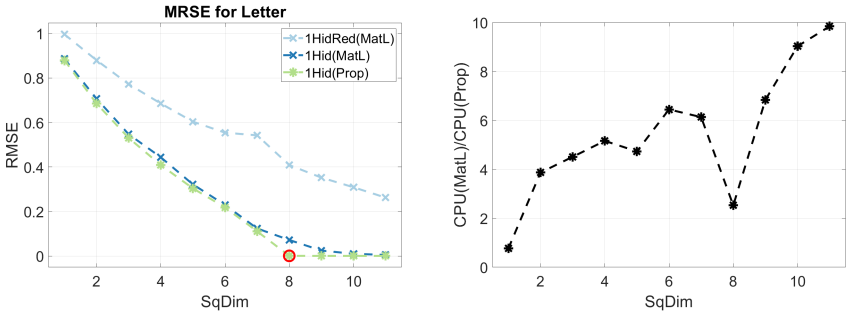

Figure 71: Comparison of forms and realizations of AE for Letter.

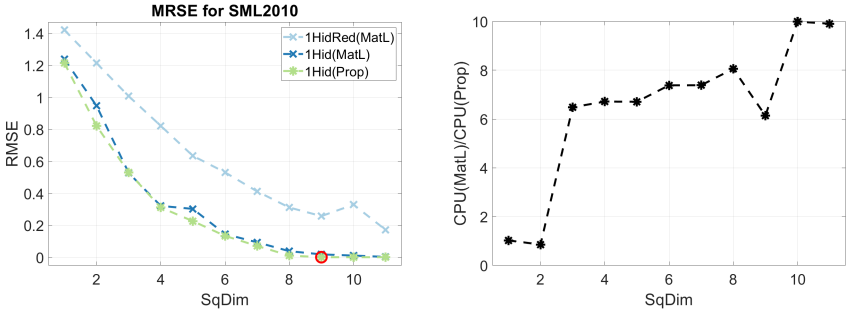

Figure 72: Comparison of forms and realizations of AE for SML2010.

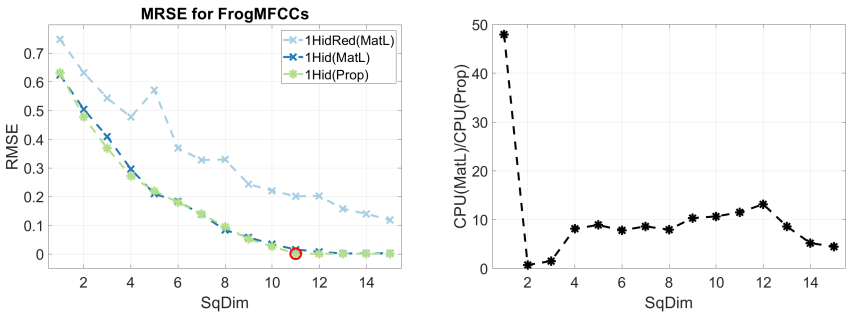

Figure 73: Comparison of forms and realizations of AE for FrogMFCCs.

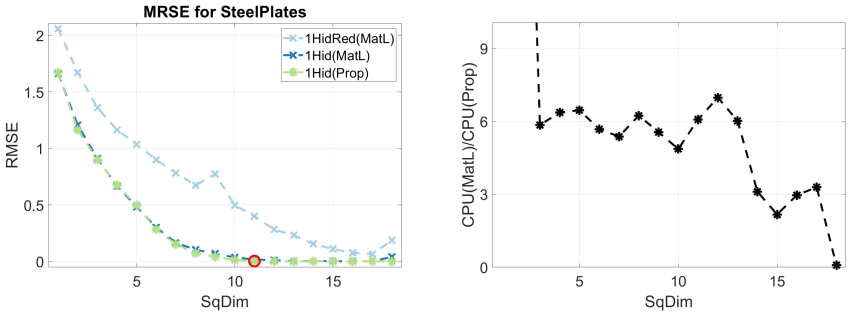

Figure 74: Comparison of forms and realizations of AE for SteelPlates.

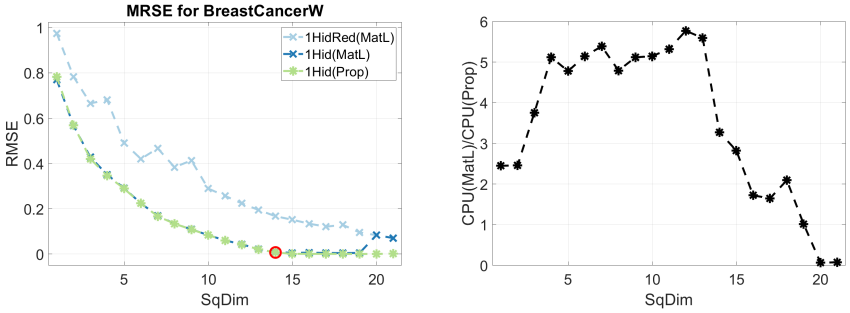

Figure 75: Comparison of forms and realizations of AE for BreastCancerW.

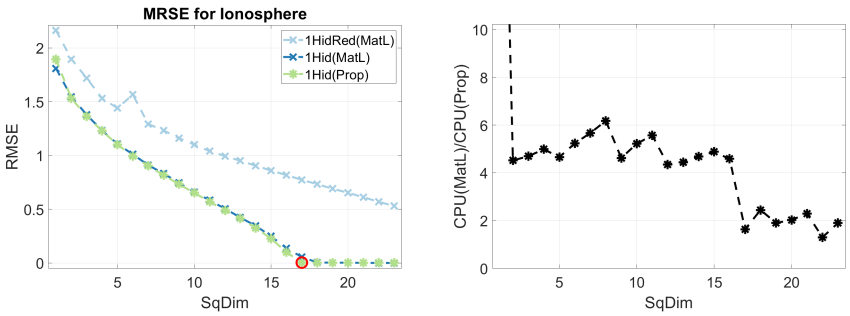

Figure 76: Comparison of forms and realizations of AE for Ionosphere.

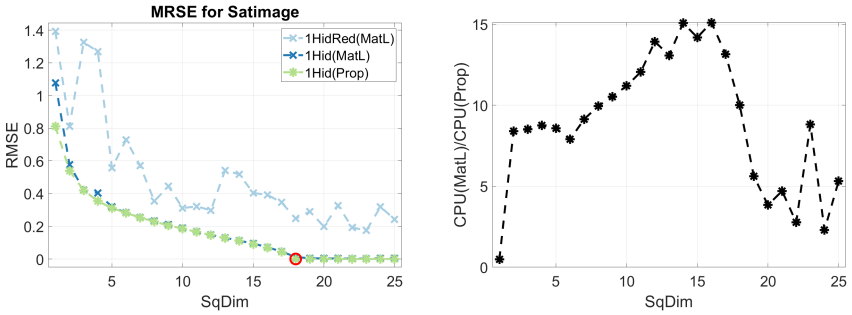

Figure 77: Comparison of forms and realizations of AE for Satimage.

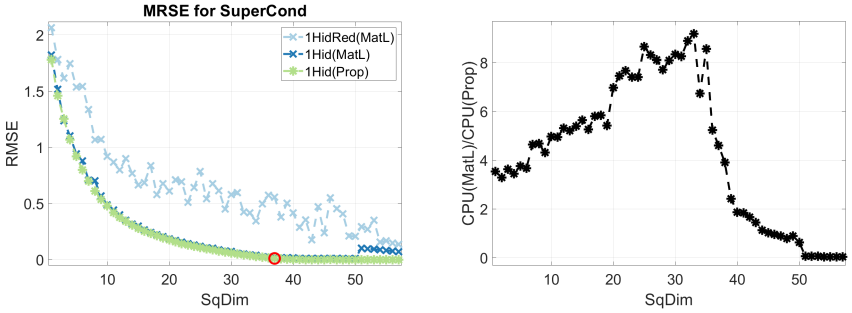

Figure 78: Comparison of forms and realizations of AE for SuperCond.

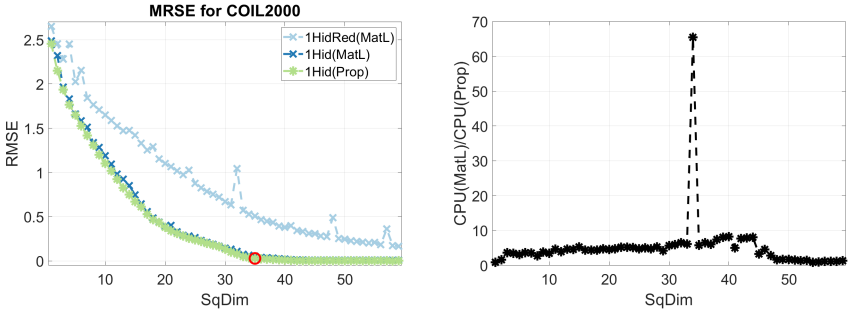

Figure 79: Comparison of forms and realizations of AE for COIL2000.

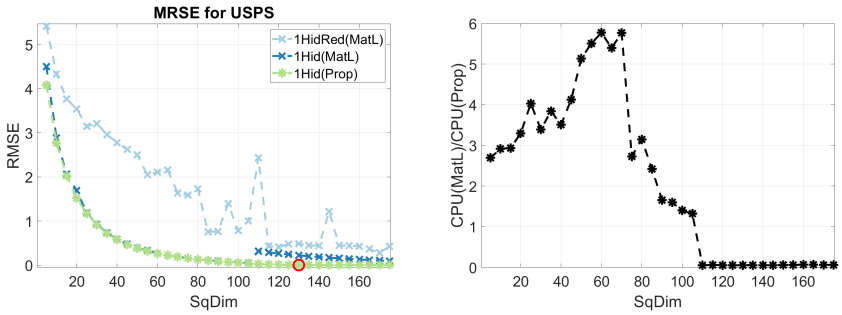

Figure 80: Comparison of forms and realizations of AE for USPS.

We conclude that use of autoencoder in the reduced dimension after linear operator does not provide small autoencoder errors or identify intrinsic dimension. On the other hand, both Matlab's own and our implemented autoencoders are able to reduce the reconstruction to a stable, almost zero level. However, Matlab's own routine may have unexpected search behavior (COIL2000 and USPS; right figures) and inferior accuracy in or near the intrinsic dimension (Wine, Letter, and USPS; left figures) compared to our implementation. Most importantly, our method (which can be applied with any number and size of layers) is typically many times faster than the Matlab version. In general, this test indicates that different autoencoder models as depicted in Section 1.1 of the main paper could be used for the nonlinear residual estimation.

References

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M. & Farhan, L. (2021), 'Review of deep learning: concepts, CNN architectures, challenges, applications, future directions', *Journal of big Data* **8**(1), 1–74.

Carletti, V., Greco, A., Percannella, G. & Vento, M. (2020), 'Age from faces in the deep learning revolution', *IEEE transactions on pattern analysis and machine intelligence* **42**(9), 2113–2132.

Chen, S. & Zhao, Q. (2018), 'Shallowing deep networks: Layer-wise pruning based on feature representations', *IEEE transactions on pattern analysis and machine intelligence* **41**(12), 3048–3056.

Dennis Jr., J. E. & Schnabel, R. B. (1996), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Vol. 16, Siam.

Elsken, T., Metzen, J. H. & Hutter, F. (2019), 'Neural architecture search: A survey', *The Journal of Machine Learning Research* **20**(1), 1997–2017.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press.

Guo, C., Pleiss, G., Sun, Y. & Weinberger, K. Q. (2017), 'On calibration of modern neural networks'. arXiv preprint arXiv:1706.04599.

Hornik, K., Stinchcombe, M. & White, H. (1989), 'Multilayer feedforward networks are universal approximators.', *Neural Networks* **2**(5), 359–366.

Kärkkäinen, T. (2014), On cross-validation for MLP model evaluation, *in* 'Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)', Springer, pp. 291–300.

Kärkkäinen, T. & Heikkola, E. (2004), 'Robust formulations for training multilayer perceptrons', *Neural Computation* **16**(4), 837–862.

Kingma, D. P. & Ba, J. (2015), Adam: A method for stochastic optimization, *in* 'Proceedings of International Conference on Learning Representations'. ArXiv: 1412.6980.

Lathuilière, S., Mesejo, P., Alameda-Pineda, X. & Horaud, R. (2020), 'A comprehensive analysis of deep regression', *IEEE transactions on pattern analysis and machine intelligence* **42**(9), 2065–2081.

Le, Q. V., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B. & Ng, A. Y. (2011), On optimization methods for deep learning, *in* 'Proceedings of the 28th International Conference on Machine Learning', pp. 265–272.

Moreno-Torres, J. G., Sáez, J. A. & Herrera, F. (2012), 'Study on the impact of partition-induced dataset shift on k-fold cross-validation', *IEEE Transactions on Neural Networks and Learning Systems* **23**(8), 1304–1312.

Needell, D., Srebro, N. & Ward, R. (2016), 'Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm', *Mathematical Programming* **155**(1-2), 549–573.

Nocedal, J. (1980), 'Updating quasi-Newton matrices with limited storage', *Mathematics of Computation* **35**(151), 773–782.

Nocedal, J. & Wright, S. (2006), *Numerical Optimization*, Springer Science & Business Media.

Pinkus, A. (1999), 'Approximation theory of the mlp model in neural networks', *Acta Numerica* **8**, 143–195.

Sejnowski, T. J. (2020), 'The unreasonable effectiveness of deep learning in artificial intelligence', *Proceedings of the National Academy of Sciences* . www.pnas.org/cgi/doi/10.1073/pnas.1907373117.

Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A. & Goldstein, T. (2016), Training neural networks without gradients: A scalable ADMM approach, *in* 'International Conference on Machine Learning (ICML)', PMLR, pp. 2722–2731.

Yu, S. & Principe, J. C. (2019), 'Understanding autoencoders with information theoretic concepts', *Neural Networks* **117**, 104–123.