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 19, 2022

Abstract

An additive autoencoder for dimension reduction, which is composed of a serially performed bias estimation, linear trend estimation, and nonlinear residual estimation, is proposed and analyzed. Computational experiments confirm that an autoencoder of this form, with only a shallow network to encapsulate the nonlinear behavior, is able to identify an intrinsic dimension of a dataset with a low autoencoding error. This observation leads to an investigation in which shallow and deep network structures, and how they are trained, are compared. We conclude that the deeper network structures obtain lower autoencoding errors during the identification of the intrinsic dimension. However, the detected dimension does not change compared to a shallow network.

1 Introduction

Dimension reduction is one of the typical data transformation techniques used in data mining. Both linear and nonlinear techniques can be used to transform a set of observations into a smaller dimension (Burges et al. 2010). A specific and highly popular set of nonlinear methods are provided with autoencoders, AE (Schmidhuber 2015), which by using the original inputs as targets integrate unsupervised and supervised learning for dimension reduction. The main purpose of this paper is to propose and thoroughly test an autoencoding model, which comprises an additive combination of linear and nonlinear dimension reduction techniques through serially performed bias estimation, linear trend estimation, and nonlinear residual estimation. Preliminary, limited investigations of such a model structure have been reported in (Kärkkäinen & Rasku 2020, Kärkkäinen 2022).

With the proposed autoencoding model, we consider its ability to estimate the intrinsic dimensionality of data (Fukunaga & Olsen 1971, Camastra 2003, Lee & Verleysen 2007). According to Fukunaga (1982), the intrinsic

*Manuscript submitted to review

dimension can be defined as the size of the lower-dimension manifold where data lies without information loss. With linear principal component analysis (PCA), this loss can be measured with the explained variance which is measured by the eigenvalues of the covariance matrix Jolliffe (2002). Indeed, the use of an autoencoder to estimate the intrinsic dimension can be considered a nonlinear extension of the projection method based on PCA (Facco et al. 2017). However, in the nonlinear case measures for characterizing the essential information and how this is used to reduce the dimensionality vary (Camastra & Staiano 2016, Navarro et al. 2017).

Wang et al. (2016) concluded that a shallow autoencoder shows the best performance when the size of the squeezing dimension is approximately around the intrinsic dimension. In this direction, techniques that are closely related to our work were proposed by Bahadur & Paffenroth (2020), where the intrinsic dimension was estimated using an autoencoder with sparsity-favoring l_1 penalty and singular value proxies of the squeezing layer's encoding. In the experiments, the superiority of the autoencoder compared to PCA was concluded. This and the preliminary work by Bahadur & Paffenroth (2019) applied *a priori* fixed architectures of the autoencoder and different autoencoding error measure compared to our work. Here, multiple feedforward models are used and compared, with a simple thresholding technique to detect the intrinsic dimension based on the data reconstruction error.

Interestingly, our experiments reveal that the intrinsic dimension can be identified by using only a shallow feedforward network as the nonlinear residual operator in the additive autoencoding model. This results from including the linear operator in the overall transformation and considering the unexplained residual in the original data dimension. This does not happen with the classical autoencoder (without linear term) or if an autoencoder would be used in the reduced dimension after the linear transformation. These phenomena, with the models and techniques fully specified in the subsequent sections, is illustrated in Fig. 1. Therefore, in addition to exploring the capabilities of revealing the intrinsic dimension we assess the advantages of applying deeper networks as nonlinear operators. Apparently, our results diversify views on the general superiority of deeper network architectures. They are also linked to the existing challenges that researchers need to address with deep learning techniques (Chen & Zhao 2018, Lathuilière et al. 2020, Ghods & Cook 2021).

1.1 On Autoencoders

Feedforward autoencoders have a versatile history, beginning from (Cottrell 1985, Bourlard & Kamp 1988). Their development as part of the evolution from shallow network models into deep learning techniques with various network architectures has been depicted in numerous large and comprehensive

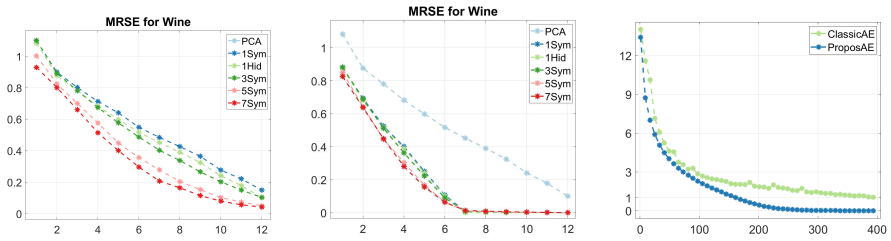

Figure 1: Residual errors with Wine dataset for the usual autoencoders with different number of hidden layers (left), for the proposed, additive autoencoders (middle), and between these two models with five-hidden-layers for the MNIST dataset (right). Here x-axis contains the squeezing dimension and y-axis the autoencoding (reconstruction) error. Deeper models provide lower autoencoding errors on the left, but, only with the additional linear operator as proposed here, the intrinsic dimension is revealed on the middle and right figures. Improvement due to the depth of the model is significant on the left but only moderate on the middle, where all models are stricly better than the linear PCA alone. On the right, better capability of the proposed autoencoder to encapsulate the variability of MNIST compared to the classical approach is clearly visible.

reviews (Schmidhuber 2015, LeCun et al. 2015, Jordan & Mitchell 2015). Therefore, we only provide a brief summary of these techniques below.

Deep feedforward autoencoding was highly influenced by the seminal work of Hinton & Salakhutdinov (2006). The work emphasized the importance of pretraining and the usability of stacking (i.e., layer-by-layer construction of the deeper architecture). Such techniques, by directing the determination of weights to a potential region of the search space, particularly alleviate the vanishing gradient problem (see (Schmidhuber 2015, Section 5.9) and references therein).

As summarized—for example, by (Liu et al. 2017), many architectures and training variants for deep autoencoders (AE) have been proposed over the years:

- *Denoising AEs (DAE)* in which noise (see (Ho et al. 2010)) is added into the training data (Vincent et al. 2010, Chen et al. 2015, Ismail Fawaz et al. 2019, Probst & Rothlauf 2020, Ma et al. 2020).
- *Sparse AEs (SAE)* in which the number of active, non-,zero weights is minimized (Filzmoser et al. 2012) (see also (Schmidhuber 2015, Section 5.6.4)).
- *Contractive AEs (CAE)* in which the reconstruction phase is penalized (Diallo et al. 2021).

- *Separable AEs (SAE)* in which two separate deep autoencoders are applied to model signal and noise spectra (Sun et al. 2015) (cf. Siamese neural networks that do the opposite and use shared weights (Ahrabian & BabaAli 2019)).

- *Graph AEs (GAE)* in which graphs, or their nodes, are encoded into latent representations and back (Wu et al. 2021, Hou et al. 2022, Yoo et al. 2022).

- *Variational AEs (VAE)* which are deep generative models that utilize Bayesian networks in learning probability distribution of data for encoding and decoding (Dai et al. 2018, Burkhardt & Kramer 2019, Zhao et al. 2021, Takahashi et al. 2022).

- *Regularized AEs (RAE)* in which the suppression of the derivatives of the encoder and regularization function orthogonal to the manifold provide local characterization of data-generation density (Alain & Bengio 2014).

- *Multi-modal AEs* can handle and unify the processing of different data modalities (Janakarajan et al. 2022).

- *Other AEs* typically integrate concepts and techniques from relevant areas, for instance, autoencoder bottlenecks (AE-BN) that are based on Deep Belief Networks (Sainath et al. 2012) and rough autoencoders (RAE) where rough set based neurons are used in the layers (Khodayar et al. 2017).

A wide variety of tasks and domains has been and can be addressed with autoencoders. AEs are typically used in numerous application domains in scenarios where transfer learning can be utilized—for example, in speech processing (Deng et al. 2017), time series prediction (Sun et al. 2018), fault diagnostics (Sun et al. 2019), and machine vision (Kim et al. 2020). In addition, interesting unsupervised hybrids are provided—for example, by clustering techniques that incorporate AEs for feature transformation (Min et al. 2018, McConville et al. 2021) and unsupervised AE-based hashing methods that can be used for large-scale information retrieval (Zhang & Qian 2021). Further, AEs have been used for the estimation of data distribution (Khajenezhad et al. 2020). Use of a variational AE for joint estimation of a normal latent data distribution and the corresponding contributing dimensions has been addressed in (Ikeda et al. 2018). Yet another use case of autoencoders is outlier detection, which might need statistically robust first-order fitting techniques instead of second-order least-squares (Gao et al. 2020, Kärkkäinen & Heikkola 2004). Data imputation has been realized using a shallow autoencoder in (Narayanan et al. 2002) and, more recently, using deep autoencoders mainly for spatio-temporal data in (Tran et al.

2017, Abiri et al. 2019, Zhao et al. 2020, Li et al. 2020, Sangeetha & Kumaran 2020, Ryu et al. 2020).

1.2 Contributions and contents

The main contribution of the paper is the derivation and evaluation of the additive autoencoding model. We provide an experimental confirmation of the new autoencoder's ability to reveal the intrinsic dimension and study the effect of model depth. Based on the similar residual idea than with the model, we also depict a simple layerwise pretraining technique. With minor role, mostly covered in the Supplementary Information (SI), we also discuss and provide an experimental illustration of the difficulties of currently popular deep learning techniques in realizing the potential of deep network models. Overall, our results suggest that current and upcoming applications in deep learning could be improved by using an explicit separation of the linear and nonlinear aspects of the data-driven model. Moreover, it might be helpful to apply more accurate training techniques.

The remainder of the paper is organized in the following manner: In Section 2, we discuss the formalization of the proposed method as a whole. In Section 3, we describe the computational experiments and summarize the main findings. In Section 4, we provide the overall conclusions and discussion. In the SI, we provide more background and preliminary material and, especially, report the computational experiments as a whole. Main findings solely covered on SI confirm the quality of the implementation of the methods and especially indicate that different autoencoder models, as depicted in the previous section, could be used for the nonlinear residual estimation.

2 Methods

In this section, we describe the methods used here as part of the autoencoding approach. In the following account, we assume that a training set of N observations $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^n$, is given.

2.1 The autoencoding model

In mathematical modelling, linear and nonlinear models are typically treated separately (Bellomo & Preziosi 1994). Following Kärkkäinen (2022), according to Taylor's formula, in the neighborhood of a point $\mathbf{x}_0 \in \mathbb{R}^n$, the value of a sufficiently smooth real-valued function $f(\mathbf{x})$ can be approximated as

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T\nabla^2 f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \ldots,$$

where $\nabla f(\mathbf{x}_0)$ denotes the gradient vector and $\nabla^2 f(\mathbf{x}_0)$ the Hessian matrix at $\mathbf{x}_0$. According to (Dennis Jr. & Schnabel 1996, Lemma 4.1.5), there exists $\mathbf{z} \in l(\mathbf{x}, \mathbf{x}_0)$ (a line segment connecting the two points), such that

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)^T(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \nabla^2 f(\mathbf{z})(\mathbf{x} - \mathbf{x}_0). \quad (1)$$

This formula yields the sufficient condition of $\mathbf{x} \in \mathbb{R}^n$ to be the local minimizer of a convex f (whose Hessian is positive semidefinite) (Nocedal & Wright 2006, Theorem 2.2): $\nabla f(x) = 0$. However, another interpretation of the formula above is that we can locally approximate the value of a smooth function as a sum of its bias (i.e., constant level), a linear term, and a nonlinear higher-order residual operator. This observation is the starting point for proposing an autoencoder that has exactly such an additive structure.

The bias estimation simply involves the elimination of its effect through normalization by subtracting the data mean and scaling each feature separately into the same range $[-1, 1]$ with the scaling factor $\frac{2}{\max(x)-\min(x)}$. Thus, we combine the mean component from z-scoring and the scaling component from min-max scaling. The reason for this is that the unit value of the standard deviation in z-scoring does not guarantee equal ranges, and min-max scaling into $[-1, 1]$ does not preserve the zero mean.

In the second phase, we estimate the linear behavior of the normalized data by using the classical principal component transformation (Bishop 1995). For a zero-mean vector $\mathbf{x} \in \mathbb{R}^n$, the transformation to a smaller-dimensional space $m < n$ spanned by m principal components (PCs) is given by $\mathbf{y} = \mathbf{U}^T\mathbf{x}$, where $\mathbf{U} \in \mathbb{R}^{n \times m}$ consists of the m most significant (as measured by the sizes of the corresponding eigenvalues) eigenvectors of the covariance matrix. Thus, because of the orthonormality of $\mathbf{U}$, the unexplained residual variance of the PC coordinates (i.e., the linear trend in $\mathbb{R}^m$) *in the original space* (see SI, Section 7) can be estimated in the following manner:

$$\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{U}\mathbf{y} = \mathbf{x} - \mathbf{U}\mathbf{U}^T\mathbf{x} = (\mathbf{I} - \mathbf{U}\mathbf{U}^T)\mathbf{x}. \quad (2)$$

This transformation is referred to as PCA. With erroneous data or data with missing values, mean and classical PCA can be replaced with their statistically robust counterparts (Kärkkäinen & Saarela 2015).

In the third, nonlinear phase, we apply the classical fully connected feedforward autoencoder to the residual vectors in (2). As anticipated by the scaling, the tanh activation function $f(x) = \frac{2}{1+\exp(-2x)} - 1$ is used. This ensures the smoothness of the entire transformation and the optimization problem of determining the weights. The currently popular rectified linear units are nondifferentiable (Kärkkäinen & Heikkola 2004) and, therefore, are not theoretically compatible with the gradient-based methods (Goodfellow et al. 2016, Section 6.3.1). The importance of differentiability was also noted in (Ghods & Cook 2021).

The formalism introduced by Kärkkäinen (2002) is used for the compact derivation of the optimality conditions. By representing the layerwise activation using *diagonal function-matrix* $\mathcal{F} = \mathcal{F}(\cdot) = \mathbb{D}iag\{f_i(\cdot)\}_{i=1}^m$, where $f_i \equiv f$, the output of a feedforward network with L layers and linear activation on the final layer reads as

$$\mathbf{o} = \mathbf{o}^L = \mathcal{N}(\tilde{\mathbf{x}}) = \mathbf{W}^L \mathbf{o}^{(L-1)}, \tag{3}$$

where $\mathbf{o}^0 = \tilde{\mathbf{x}}$ for an input vector $\tilde{\mathbf{x}} \in \mathbb{R}^{n_0}$, and $\mathbf{o}^l = \mathcal{F}(\mathbf{W}^l \mathbf{o}^{(l-1)})$ for $l = 1, \ldots, L-1$. To allow the formal adjoint transformation to be used as the decoder, we assume that L is even and that the bias nodes are not included in (3). The dimensions of the weight matrices are then given by $\dim(\mathbf{W}^l) = n_l \times n_{l-1},\ l = 1, \ldots, L$. In the autoencoding context, $n_L = n_0$ and $n_l, 0 < l < L$, define the sizes (the number of neurons) of the hidden layers with the squeezing dimension $n_{L/2} < n_0$.

To determine the weights in (3), we minimize the regularized mean least-squares cost function of the form

$$\begin{aligned}\mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L) =& \frac{1}{2N} \sum_{i=1}^N \left\| \mathbf{W}^L \mathbf{o}_i^{(L-1)} - \tilde{\mathbf{x}}_i \right\|^2 \\ &+ \frac{\alpha}{2\sqrt{\sum_{l=1}^L \#(\mathbf{W}_1^l)}} \sum_{l=1}^L \left\| \mathbf{W}^l - \mathbf{W}_0^l \right\|^2, \end{aligned} \tag{4}$$

where $\|\cdot\|$ denotes the Frobenius norm and $\#(\mathbf{W}_1^l)$ the number of rows of $\mathbf{W}^l$. Let α be fixed to 1e-6 throughout; to simplify the notations, we define $\beta = \alpha/\sqrt{\sum_{l=1}^L |\mathbf{W}_1^l|}$. The underlying idea in (4) is to average in both terms: in the first, the data fidelity (least-squares error, LSE) term, and in the second, the regularization term. Averaging the first term with $\frac{1}{N}$ implies that the term scales automatically by the size of the data subset, for instance, in minibatching, thereby providing an approximation of the entire LSE on a comparable scale. In the second term, because α is fixed, the inverse scaling constant $1/\sqrt{\sum_{l=1}^L |\mathbf{W}_1^l|}$ balances the effect of the regularization compared to the data fidelity for networks with a different number of layers of different sizes. Because (4) will be minimized with local optimizers, we simply use the initial guesses $\{\mathbf{W}_0^l\}$ of the weight values in the second term to improve the local coercivity of (4) and to restrict the magnitude of the weights, thereby attempting to improve generalization (Gouk et al. 2021). Because of the residual approximation, the random initialization of the weight matrices is generated from the uniform distribution $\mathcal{U}([-0.1, 0.1])$. s

The gradient matrices $\nabla_{\mathbf{W}^l} \mathcal{J}(\{\mathbf{W}^l\}_{l=1}^L),\ l = L, \ldots, 1$, for (4) are of the following form (see Kärkkäinen (2002)):

$$\nabla_{\mathbf{W}^l} \mathcal{J}(\{\mathbf{W}^l\}) = \frac{1}{N} \sum_{i=1}^N \mathbf{d}_i^l \, (\mathbf{o}_i^{(l-1)})^T + \beta \left(\mathbf{W}^l - \mathbf{W}_0^l \right), \tag{5}$$

where the layerwise error backpropagation reads as

$$\mathbf{d}_i^L = \mathbf{e}_i = \mathbf{W}^L \mathbf{o}_i^{(L-1)} - \tilde{\mathbf{x}}_i, \tag{6}$$

$$\mathbf{d}_i^l = \mathbb{D}iag\{(\mathcal{F})'(\mathbf{W}^l\, \mathbf{o}_i^{(l-1)})\}\, (\mathbf{W}^{(l+1)})^T\, \mathbf{d}_i^{(l+1)}. \tag{7}$$

The use of different weights in the encoding—that is, in the transformation until layer $L/2$—and decoding, from layer $L/2$ to L, implies more flexibility in the residual autoencoder but also rougly doubles the amount of weights to be determined. Therefore, it is common to use the formal adjoint $(\mathbf{W}^1)^T\mathcal{F}((\mathbf{W}^2)^T\mathcal{F}(\ldots(\mathbf{W}^{L/2})^T))$ of the encoder as the decoder. Then, it is easy to see that the layerwise optimality conditions for $l = 1, \ldots, L/2$ read as

$$\nabla_{\mathbf{W}^l}\mathcal{J} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{d}_i^l\, (\mathbf{o}_i^{(l-1)})^T + \mathbf{o}_i^{(\tilde{l}-1)}\, \left(\mathbf{d}_i^{\tilde{l}}\right)^T + \beta\left(\mathbf{W}^l - \mathbf{W}_0^l\right), \tag{8}$$

where $\tilde{l} = L - (l-1)$. For convenience, we define $\tilde{L} = L/2$—that is, the number of layers to be optimized with the symmetric models.

We note that when the layerwise formulae above are used with vector-based optimizers, we always need to reshape operations to toggle between the weight matrices and a column vector of all weights.

Remark 1 *Let us briefly summarize the use of the additive autoencoder for an unseen dataset after it has been estimated (and the corresponding data structures have been stored) for the training data through the three phases. First, data is normalized through mean subtraction and feature scaling into the same range* $[-1, 1]$. *Then, residuals according to formula* (2) *are computed and this residual data is fed to the feedforward autoencoder. Again due to* (2)*, the reduced, m-dimensional representation of new data is obtained as a sum of its PC projection and the output of the autoencoder's squeezing layer. Formula* (2) *shows that the explicit formation of the residual data between linear and nonlinear representations can be replaced by setting* $\widetilde{\mathbf{W}}^1 = \mathbf{W}^1(\mathbf{I} - \mathbf{U}\mathbf{U}^T)$ *and using this as the first transformation layer of the autoencoder for the normalized, unseen data.*

2.2 Layerwise pretraining

The core idea of the proposed autoencoding model—additive combination of operators of different complexity—can be applied in the layerwise pretraining, that is, stacking of the nonlinear autoencoding part as well. We propose to use an identical network structure and learning paradigm for this purpose differently from, e.g., (Hinton & Salakhutdinov 2006). A similar idea appears with the deep residual networks (ResNets) in He et al. (2016), where consecutive residuals are stacked together using layer skips—for example, over two

or three layers with batch normalization. Moreover, in ResNets, the layer skips can introduce additional weight matrices to the deep network model. However, the layer-by-layer pretraining of the symmetric autoencoder, from the heads toward the inner layers, can be simply performed directly.

The approach is illustrated in Fig. 2. For three hidden layers with two unknown weight matrices, $\mathbf{W}^1$ and $\mathbf{W}^2$, we first estimate $\mathbf{W}^1$ with the given data $\{\tilde{\mathbf{x}}_i\}$. Then, the output data of the estimated layer $\{\mathbf{W}^1\tilde{\mathbf{x}}_i\}$ are used as the training data (the input and the desired output) for the second layer $\mathbf{W}^2$. Thereafter, the entire network is fine-tuned by optimizing over both weight matrices. The process from the heads to the inner layers is naturally enlarged for a larger number of hidden layers. We could then also apply partial fine-tuning—for example, to fine-tune the three hidden layers during the process of constructing a five-hidden-layer network. However, according to our tests and similar to Hinton & Salakhutdinov (2006), the layerwise pretraining suffices before fine-tuning the entire network. A special case of utilizing a simpler structure is the one-hidden-layer case: The symmetric model 1Sym with one weight matrix if first optimizer to obtain $\mathbf{W}^1$ and then used in the form $((\mathbf{W}^1)^T, \mathbf{W}^1)$ as the initial guess for optimizing the nonsymmetric model 1Hid with two weight matrices. Again such an approach could be generalized to multiple hidden-layer case for the nonsymmetric, deep autoencoding model.

Remark 2 *As stated in the introduction, stacking attempts to mitigate the vanishing gradient problem, which may prevent the adaptation of the weights in deeper layers. We assessed the possibility of such a phenomenon by studying the relative changes in the weight matrix norms* $(\big|\|\mathbf{W}_0^l\| - \|\mathbf{W}_*^l\|\big|/\|\mathbf{W}_0^l\|)$ *while fine-tuning the symmetric autoencoders with 3–7 layers (*3Sym, 5Sym,

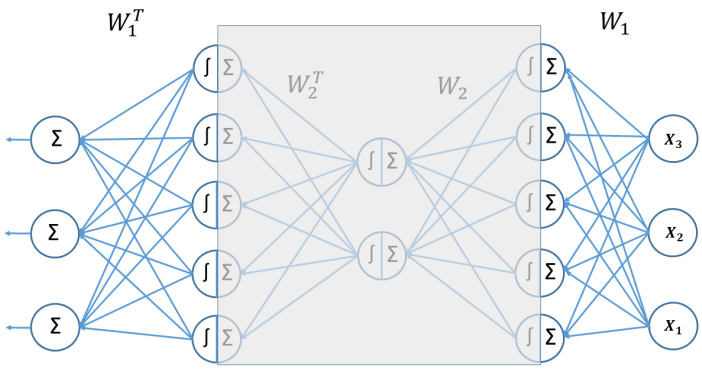

Figure 2: Layerwise pretraining from heads to inner layers. The most outer layer is trained first and its residual is then fed as training data for the next hidden layer until all layers have been sequentially pretrained.

and 7SYM*; see Section 3). The subscripts '0' and '∗' refer to the initial and final weight matrix values, respectively. This study revealed that the relative changes in the weights in the deeper layers were not on a smaller numerical scale compared to the other layers. Apparently, the double role of the layers in the symmetric models as part of the encoder and the decoder, with the corresponding effect on the gradient as seen in formula* (8)*, is also helpful in avoiding a vanishing gradient.*

2.3 Determination of intrinsic dimension

The basic procedure to determine the intrinsic dimension is to gradually increase the size of the squeezing layer and to seek a small value of the reconstruction error measuring autoencoding error, with a knee point (Thorndike 1953) indicating that the level of nondeterministic residual noise has been reached in autoencoding. Instead of the usual root mean squared error (RMSE), we apply the mean root squared error (MRSE) to compute the autoencoding error:

$$e = \frac{1}{N}\sqrt{\sum_{i=1}^{N}|\mathbf{x}_i - \mathcal{N}(\mathbf{x}_i)|^2}, \tag{9}$$

where $\{\mathbf{x}_i\}$ is assumed to be normalized and $\mathcal{N}$ denotes the application of the autoencoder. This choice was made because in Kärkkäinen (2014), MRSE correlated better with the independent validation error. In practice, the difference between the RMSE and the MRSE is only the scaling factor, $1/\sqrt{N}$ vs. $1/N$. After the linear PC trend estimation, the MRSE is obtained by using (9) for the residual data defined in formula (2). Note that the reconstruction error is a strict error measure and its use requires higher accuracy from autoencoding compared to other measures: with the Wine dataset in Fig. 1, the linear PCA needs all dimensions of the rotated coordinate axis for the reconstruction whereas already 10 principal components out of 13 would explain over 96% of the data variance.

An example of determining the intrinsic dimension of the Glass dataset (see the next section) is presented in Fig. 3. In the figure, the x-axis "SqDim" presents the squeezing dimension and the y-axis on the left the "MRSE" and on the right its change "Δ(MRSE)" (i.e., backward difference) for the symmetric model with one hidden dimension (1SYM) and the corresponding nonsymmetric model 1HID. The intrinsic dimension of data is detected by first locating a sufficiently small change in the autoencoding error (the right plot). For this purpose, a user-defined threshold τ = 4e-3 is applied. The detected dimension 5 on the left, marked with a circle, is the dimension below the threshold on the right minus one. The intrinsic dimension 5 is also characterized by a clear knee point in the MRSE behavior.

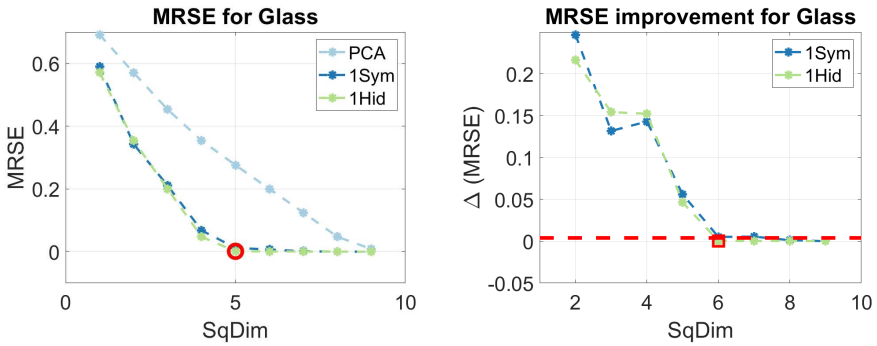

Figure 3: Identification of the intrinsic dimension for the Glass dataset. The hidden dimension (plus one) on the left is captured by the sufficiently small error improvement on the right.

3 Results

The main focus of the computational experiments, which are fully reported in the Supplementary Information (SI), was to investigate the ability of the proposed additive autoencoder model to represent a dataset in a lower-dimensional space. Therefore, we confined ourselves to the use of Matlab as the platform (mimicking the experiments in Hinton & Salakhutdinov (2006)) to have full control over the realization of the methods in order to study the effects of different parameters and configurations. Reference implementation of the proposed method and its basic testing is available in GitHub[1].

We apply and compare the following set of techniques to approximate the nonlinear residual of the autoencoder, after normalization and identification of the linear trend: 1Hid (model with one hidden layer and separate weight matrices for the encoder and the decoder), 1Sym (symmetric model with one hidden layer and a shared weight matrix), 3Sym (three-hidden-layer symmetric model with two shared weight matrices), 5Sym, and 7Sym. To systematically increase the flexibility and the approximation capability of the deeper models, the sizes of the layers for $l = \tilde{L}, \ldots, 1$ are given below, where $n_{\tilde{L}}$ is the size of the squeezing layer:

3Sym: $n_{\tilde{L}} - 2n_{\tilde{L}} - n$,

5Sym: $n_{\tilde{L}} - 2n_{\tilde{L}} - 4n_{\tilde{L}} - n$,

7Sym: $n_{\tilde{L}} - 2n_{\tilde{L}} - 3n_{\tilde{L}} - 4n_{\tilde{L}} - n$.

Note that for $n_{\tilde{L}} > n/2$ the size of the second layer and, therefore, the dimension of the first intermediate representation, is larger than the input dimension for all these models.

[1] `https://github.com/TommiKark/AdditiveAutoencoder`

Table 1: Results of the identification of the intrinsic dimension for small-dimension datasets. The intrinsic dimensions were identified with the reduction rates varying between 0.41–0.54. The SteelPlates and COIL2000 (with the most discrete feature profile) have the best reduction rate. The residual errors are between 1.1e-2–4.3e-4.

Dataset	N	n	ID	Red	MRSE	FeatProf (%)
Glass	214	10	5	0.50	1.3e-3	10-40-50-0
Wine	178	13	7	0.54	1.2e-3	0-46-54-0
Letter	20 000	16	8	0.50	9.4e-4	0-100-0-0
SML2010	2 763	17	9	0.53	5.4e-4	0-12-18-71
FrogMFCC	7 195	22	11	0.50	1.1e-3	0-0-5-95
SteelPlates	1 941	27	11	0.41	4.3e-3	11-11-56-22
BreastCancerW	569	30	14	0.47	6.9e-3	0-0-100-0
Ionosphere	351	33	17	0.52	1.9e-3	3-0-97-0
SatImage	6 435	36	18	0.50	4.3e-4	0-75-25-0
SuperCond	21 263	82	37	0.45	1.1e-2	2-1-12-84
COIL2000	5 822	85	35	0.41	2.8e-2	99-1-0-0

3.1 Identification of the intrinsic dimension

The first purpose of the experiments was to search for the intrinsic dimension of a dataset via autoencoding. This was done using the shallow models 1Sym and 1Hid. The optimization settings and visualization of all results are given in the online SM.

The experiments were carried out for two groups of datasets, one with small-dimension data (less than 100 features) and the other with large-dimension data (up to 1024 features). The datasets were obtained from the UCI repository (Dua & Graff 2017), except the FashMNIST, which was downloaded from GitHub[2]. For most of the datasets, only the training data was used; however, with Madelon, the given training, testing, and validation datasets were combined. The datasets do not contain missing values. The constant features were identified and eliminated according to whether the difference between the maximum and minimum values of a feature was less than $\sqrt{MEps}$, where $MEps$ denotes machine epsilon (this is classically used numerical proxy of zero, see (Dennis Jr. & Schnabel 1996, p. 12)). Because of this preprocessing, the number of features n in Tables 1 and 2 is not necessarily the same as that in the UCI repository.

During the search, the squeezing dimension for the small-dimension datasets began from one and was incremented one by one up to $n - 1$. For the large-dimension cases, we began from 10 and used increments of 10 until the maximum squeezing dimension $\lfloor 0.6 \times n \rfloor$ was reached (cf. the

[2] `https://github.com/zalandoresearch/fashion-mnist`

Table 2: Results of the identification of the intrinsic dimension for large-dimension datasets. The intrinsic dimensions were identified with the reduction rates varying between 0.39–0.55. The HumActRec dataset with a continuous feature profile has the best reduction rate. The residual errors are between 8.0e-2–2.9e-3.

Dataset	N	n	ID	Red	MRSE	FeatProf (%)
USPS	9 298	256	130	0.51	2.9e-3	0-0-0-100
BlogPosts	52 397	277	130	0.47	3.9e-3	79-8-12-1
CTSlices	53 500	379	180	0.47	6.3e-2	8-4-10-78
UJIIndoor	19 937	473	200	0.42	7.0e-2	26-73-1-0
Madelon	4 400	500	250	0.50	7.9e-2	2-31-67-0
HumActRec	7 351	561	220	0.39	8.0e-2	0-2-0-98
Isolet	7 797	617	310	0.50	9.4e-3	1-6-14-80
MNIST	60 000	717	350	0.49	9.3e-3	9-14-77-0
FashMNIST	60 000	784	380	0.48	5.0e-2	0-2-98-0
COIL100	7 200	1 024	560	0.55	6.4e-3	0-11-89-0

"Red" values in Tables 1 and 2). The experiments were run with Matlab on a Laptop with 2.3GHz Intel i7 processor and 64 GB RAM and on a server with a Xeon E5-2690 v4 CPU and 384 GB of memory.

In Tables 1 and 2, we present the name of the dataset, the number of observations N, the number of features n, and the detected intrinsic dimension ID. The autoencoding error trajectories and thresholdings are illustrated for all datasets in the online SM. The detection threshold for small-dimension datasets was fixed to $\tau =$ 4e-3. The reduction rate ID/n for the intrinsic data dimension is reported in the Red column, and the autoencoding error of 1Hid for ID according to (9) is included in the MRSE column. There is no averaging over the data dimension n in (9), so that for higher-dimension datasets this error is expected to remain larger. This was probably one of the reasons why, for large-dimension datasets, we needed to use two values of the threshold τ (based on visual inspection; see the zoomed illustrations in the SM): 3e-3 for USPS, BlogPosts, HumActRec, MNIST, and COIL100, and 3e-2 for the remaining five datasets.

For the analysis, we also included a depiction of how discrete or continuous the set of features for a dataset is. We categorized the features into four groups based on the number of unique values (UV) each feature has: C1 = $\{\text{UV} \leq 10\}$, C2 = $\{10 < \text{UV} \leq 100\}$, C3 = $\{100 < \text{UV} \leq 1000\}$, and C4 = $\{1000 < \text{UV}\}$. The FeatProf column in Tables 1 and 2 presents the proportions of C1–C4 in percentages.

Conclusions

Examples of the ID detection are given in Figs. 3 (Glass), 4 (Letter on the left, SuperCond on the right), 5 (MNIST), and 6 (FashMNIST on the left, COIL100 on the right). Identifications in the first two cases and for FashMNIST are characterized by clear knee-points in IDs. For SuperCond, with gradual decrease of the MRSE, determination of ID is based on the mutual threshold value τ = 4e-3 of small-dimension datasets. Also MNIST has such a behavior and the zoom in Fig. 5 (right) illustrates the detection decision with τ = 3e-3.

Overall, the intrinsic dimensions were successfully identified for all tested datasets. The use of a feedforward network to approximate the nonlinear residual notably decreased the autoencoding error of the linear PCA. The

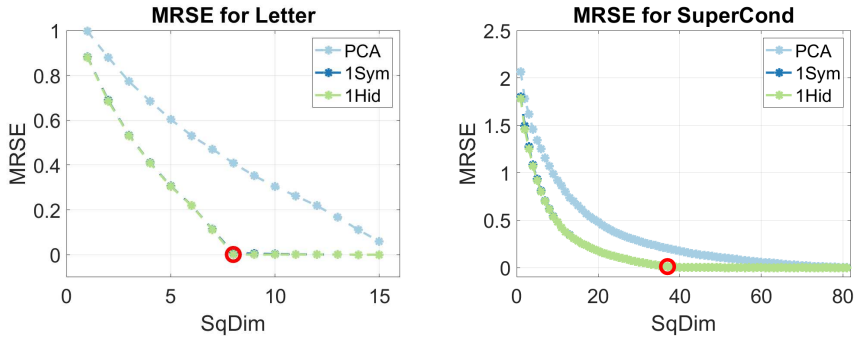

Figure 4: Identification of the intrinsic dimension for the Letter dataset and SuperCond dataset. **Left**: Clearly identified knee-point in ID = 8 for Letter. **Right**: Gradual decrease of MRSE with ID = 37 for SuperCond.

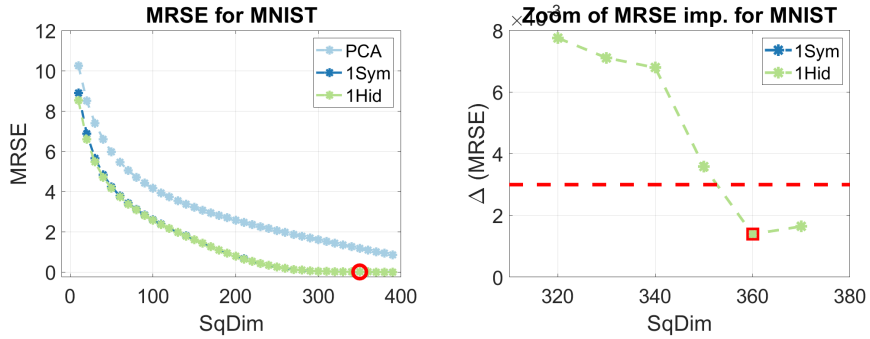

Figure 5: Identification of the intrinsic dimension for the MNIST dataset. The hidden dimension (plus one) on the left is captured by the sufficiently small error improvement on the right. **Left**: Gradual decrease of MRSE with ID = 350 for MNIST. **Right**: Zoom of MRSE improvement with the threshold τ = 3e-3 confirms the detection.

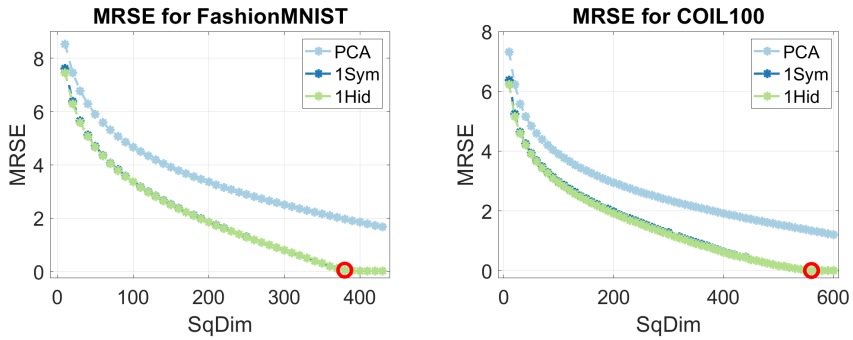

Figure 6: Identification of the intrinsic dimension for the FashionMNISTS dataset and COIL100 dataset. **Left**: Clear knee-point of MRSE with ID = 380 for FashMNIST. **Right**: More gradual decrease of MRSE with ID = 560 for COIL100.

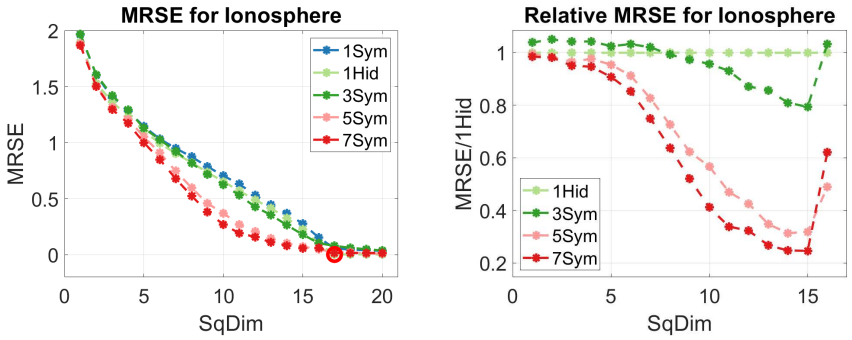

Figure 7: **Left**: Behavior of the MRSE with all residual models for Ionosphere. **Right**: Relative performance of the models with respect to 1Hid. During the search of the ID the deeper models show clear improvement but the detected ID is the same for all models.

overall transformation summarizing the essential behavior of data roughly halved the original dimension: The mean reduction rate over the 21 datasets was 0.48.

The reduction rate was independent of the form of the features—that is, the best reduction rates for small-dimension datasets were obtained with the very categorical COIL2000 and primarily continuous SteelPlates datasets. However, the best reduction rate, 0.39, was obtained for HumActRec, which is characterized by a continuous feature profile. This indicates that we may obtain smaller reduction rates with more continuous sets of features.

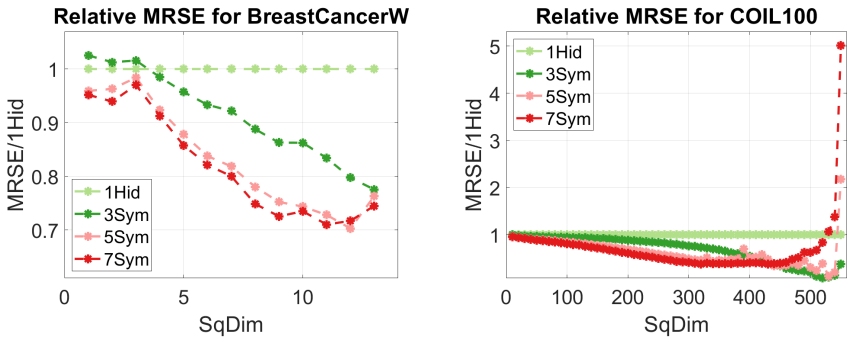

Figure 8: **Left**: Relative performance of the models for BreastCancerW. **Right**: Relative performance of the models for COIL100. The deeper models show clear improvement over the shallow ones but the detected ID stays the same and near ID the improved efficiency may be completely lost.

3.2 Comparison of shallow and deep models

The second aim of the experiments was to examine whether deeper network structures and deep learning techniques (the network structure and optimization of the weights) can improve the identification of the intrinsic dimension and the data restoration ability of the additive autoencoder. This aim is pursued as follows: Here, we compare shallow and deep networks in cases where fine-tuning is performed using a classical optimization approach, i.e., using the L-BFGS optimizer with the complete dataset. In the SI, we report the results of using different minibatch-based approaches. Also detailed depictions of the parameter choices and visualization of the results for

Table 3: 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 4: 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)

all datasets are included there.

In addition to visual assessment, we performed a quantitative comparison between the deep and shallow models. First, the MRSE values of all models were divided with the corresponding value of the 1Hid model's error. This was done for the squeezing dimensions from the first until next to last of ID, to cover the essential search phase of the intrinsic dimension. To exemplify relative performance, 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.

The relative performances are illustrated in Figs. 7 and 8. Descriptive statistics of the efficiencies of symmetric models are given in Tables 3 and 4. In each cell there, both the mean efficiency and the maximal efficiency are provided. The latter includes, in parentheses, the squeezing dimension where it was encountered.

Conclusions

During the early phases of searching the intrinsic dimension, deep networks provide smaller autoecoding errors compared to the shallow models. However, as exemplified in Fig. 7 (left) and is evident from all illustrations in the SI, MRSE in ID is not better for deeper models compared to 1Hid. Therefore, the use of a deeper model would not change the ID values and, in fact, fluctuation of the error compared to 1Hid may hinder the detection of a knee-point and negatively affect the simple thresholding.

Usually, the mean efficiency of the two deepest models, 5Sym and 7Sym, is better than that of 1Sym or 3Sym, but for many datasets, there is only a slight improvement. Overall, the plots for the relative efficiencies between different symmetric models and the quantitative trends in the rows of Tables 3 and 4 include varying patterns. 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). The following are the grand means of the mean efficiencies over all 21 datasets for the symmetric models: 1Sym 0.97, 3Sym 1.19, 5Sym 1.38, and 7Sym 1.44. This concludes that deeper models improve the reduction of the autoencoding error during the search of ID. However, the speed of improvement decreases as a function of the number of layers.

Actually, close to the intrinsic dimension, the benefits of deeper models may be completely lost. This is illustrated in Fig. 8 (right) and in Table 4 for COIL100 (see also, for example, plots of BlogPosts and MNIST in the SI). The reason for such a behavior with COIL100 is the value of ID, 560, which was obtained with the smaller threshold τ = 3e-3 for large-dimension datasets. Therefore, with COIL100, we also tested an alternative approach to the identification of ID, where we apply the same thresholding technique (and the same τ) to the minimum autoencoding error of the all models. This error plot, the identified ID = 520 (for which the reduction rate would be 0.51), and the corresponding reduced set of relative efficiencies are illustrated in Figure 9. The summary of the efficiencies for this modified way to identify ID are given in the last line "COIL100-Min" in Table 4. It can be concluded that for the largest dimensional dataset COIL100, the use of the minimum autoencoding error of the models yielded more reasonable results.

We used a fixed pattern for the sizes of the hidden layers in deeper models: 2-3-4 times the squeezing dimension for 7Sym, the first and last of these coefficients for 5Sym, and the first one for 3Sym. As reported in Section 3.1, the mean reduction rate over the 21 datasets was close to 0.5. Therefore, one may wonder whether this behavior is due to the fact that from this case onwards all the hidden dimensions are larger than the number of features so that a kind of nonlinear kernel trick occurs. In other words, would a different pattern of the hidden dimensions change the results and conclu-

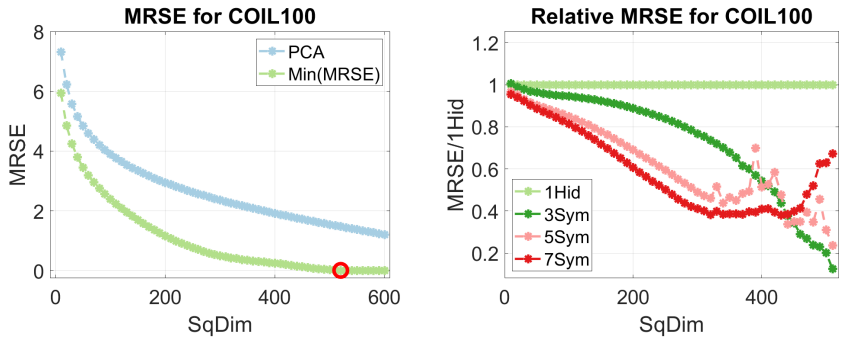

Figure 9: **Left**: Minimum autoencoding error of the all models for COIL100. **Right**: Reduced relative performance of the models for COIL100. For COIL100, use of minimum autoencoding error to identify ID yielded more reasonable results.

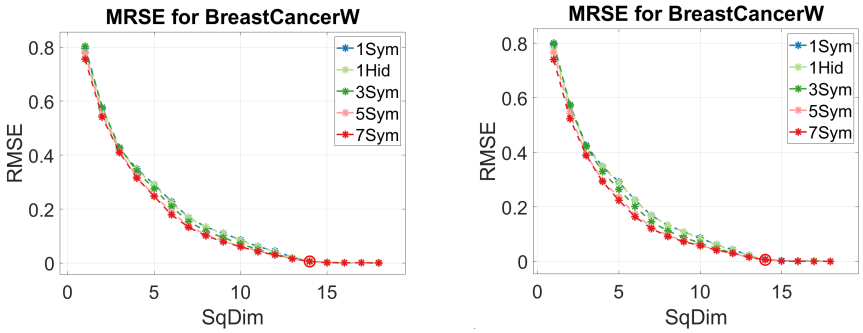

Figure 10: **Left**: RMSEs for BreastCancerW with the original 2-3-4 pattern for the hidden layers. **Right**: RMSEs for BreastCancerW with 3-5-7 pattern for the hidden layers. Slighly smaller errors were encountered during the early search phase of the larger model on the right but the detected ID and the overall behavior remained the same.

sions here? This consideration was tested by considering a 3-5-7 pattern providing much more flexibility for the nonlinear operator compared to the used pattern. These tests are not reported as a whole, because the clearly identified trend of the results is readily exemplified in Fig. 10: Increase of the sizes of the hidden layers slightly improve the reduction rate during early phase of the search but does not change the value of the ID.

3.3 Generalization of the autoencoder

In the last experiments, we demonstrate and evaluate the generalization of the additive 5Sym autoencoder. Search over squeezing dimensions is performed in a similar manner as that done in Section 3.2. We apply a small sample of datasets, for which a separate validation data was given

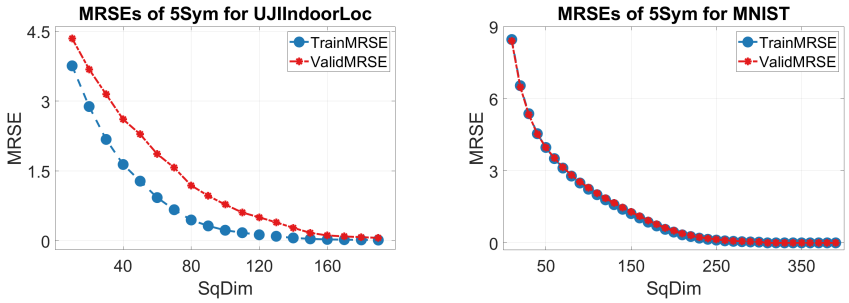

Figure 11: Agreements of training and validation set MRSE values for UJIIndoor (left) and MNIST (right). Large deviation between training-validation errors on the left but perfect match on the right. In ID, similar autoencoding error level is reached with both datasets.

in the UCI repository. More precisely, we use Letter (size of training data $N = 16000$, size of validation data $N_v = 4000$, i.e., 80%–20% portions with respect to the entire 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.

As anticipated, both training-validation portions and the data dimension affected the generalization results. For Letter, with 80%-20% division between training-validation sizes and small number of features, we witnessed a perfect match between the training and validation MRSE values. The same held true for MNIST, which is illustrated in Fig. 11 (right). The largest discrepancy between the training and validation errors, depicted in Figure 11 (left), was obtained for UJIIndoor, which has the most deviating 95%-5% portions with almost 500 features. This dataset also had one of the largest efficiencies (i.e., reduction potential) in Table 4. HumActRec was somewhere in the middle in its behavior, with clearly visible deviation. Because of the data portions (∼70%-30%), the difference raises doubts regarding the quality of the validation set. Note that these considerations provide examples of the possibilities of autoencoders to assess the quality of data.

The visual inspection was augmented by computing the correlation coefficient between the MRSE values in 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 Fig. 11 is that when the squeezing dimension is increased up to the intrinsic dimension, then the validation error tends to the same error level than the training error. Therefore, the

additive autoencoder determined using the training data was always able to explain the variability of the validation data with a compatible accuracy.

4 Conclusions

This study illustrated a case where all main concerns with feedforward mappings summarized in (Hornik et al. 1989, p. 363) were solved: learning was successful, the size and the number of hidden layers were identified, and the deterministic relationship within a dataset was found. Similar to Hinton & Salakhutdinov (2006), stacking was found to be an essential building block for estimating the weights of deep autoencoders. However, the pretraining face was conceptually (the structure and optimization of the weights) one-to-one with the corresponding part of the final, fine-tuned autoencoder. This was different from deep residual networks (He et al. 2016), where layer skips over multiple layers with batch normalization were applied. The other main ingredients of the proposed autoencoder were an automatically scalable cost function with compact layerwise weight calculus and a simple heuristic for determining the intrinsic data dimension. Intrinsic dimensions for all tested datasets with a low autoencoding error were revealed. A similar autoencoding error, and the corresponding intrinsic dimension, was obtained independently on the depth of the network.

One clear advantage of the proposed methodology is the lack of metalevel parameters (e.g., number and form of layers, selection of activation function, detection of the learning rate) that are usually tuned or gridsearched when DNNs are trained. The only parameter that may need adjustment based on visual assessment is τ—that is, the threshold for identifying the hidden dimension. Moreover, because of the observed smoothly decreasing behavior of the autoencoding error, the intrinsic dimension could be searched for more efficiently than just incrementally: One could attempt to utilize one-dimensional optimization techniques like a golden-section search and/or polynomial and spline interpolation to more quickly identify the beginning of the error plateau.

These results challenge the common beliefs and currently popular traditions with deep learning techniques. The experiments summarized here and given in the SI suggest that many existing deep learning results could be improved by using a clear separation of linear and nonlinear data-driven models. Also use of more accurate optimization techniques to determine the weights of such models may be advantageous.

We can use the additive transformation to the intrinsic dimension as a pretrained part for transfer learning with any prediction or classification model (Ghods & Cook 2021). It would be interesting to test in the future whether one should use this as is or would a transformation into a smaller squeezing dimension than the intrinsic one generalize better in prediction

and classification tasks? Another detectable dimensions of the squeezing layer worth investigating, as illustrated in the relative MRSE plots (see also the SI) and in Tables 3 and 4, could be the one with the largest nonlinear gain—that is, with the maximum difference between the PCA error and the autoencoder error or between the shallow and deep results. Moreover, we used global techniques in every part of the autoencoder. The technique might benefit from encoding locally estimated behavior—for example, using convolutional layers for local-level estimation (LeCun et al. 1990). Similarly, other linear transformation techniques and modifications of PCA might provide better performance (Burges et al. 2010, Song et al. 2018, Vogelstein et al. 2021), although in the proposed form, we also need the inverse of the linear mapping to be able to estimate the residual error in the original vector space.

Supplementary information

Complementary background material and some additional methodological comparisons are placed as Supplementary Information. Especially, full coverage of all the results and illustrations of the computational experiments are given there.

Acknowledgments

This work was supported by the Academy of Finland from the project 351579 (MLNovCat).

References

Abiri, N., Linse, B., Edén, P. & Ohlsson, M. (2019), 'Establishing strong imputation performance of a denoising autoencoder in a wide range of missing data problems', *Neurocomputing* **365**, 137–146.

Ahrabian, K. & BabaAli, B. (2019), 'Usage of autoencoders and siamese networks for online handwritten signature verification', *Neural Computing and Applications* **31**(12), 9321–9334.

Alain, G. & Bengio, Y. (2014), 'What regularized auto-encoders learn from the data-generating distribution', *The Journal of Machine Learning Research* **15**(1), 3563–3593.

Bahadur, N. & Paffenroth, R. (2019), 'Dimension estimation using autoencoders', *arXiv preprint arXiv:1909.10702* .

Bahadur, N. & Paffenroth, R. (2020), Dimension estimation using autoencoders with applications to financial market analysis, *in* '2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)', IEEE, pp. 527–534.

Bellomo, N. & Preziosi, L. (1994), *Modelling mathematical methods and scientific computation*, Vol. 1, CRC press.

Bishop, C. M. (1995), *Neural Networks for Pattern Recognition*, Oxford University Press.

Bourlard, H. & Kamp, Y. (1988), 'Auto-association by multilayer perceptrons and singular value decomposition', *Biological cybernetics* **59**(4), 291–294.

Burges, C. J. et al. (2010), 'Dimension reduction: A guided tour', *Foundations and Trends® in Machine Learning* **2**(4), 275–365.

Burkhardt, S. & Kramer, S. (2019), 'Decoupling sparsity and smoothness in the dirichlet variational autoencoder topic model', *Journal of Machine Learning Research* **20**(131), 1–27.

Camastra, F. (2003), 'Data dimensionality estimation methods: a survey', *Pattern recognition* **36**(12), 2945–2954.

Camastra, F. & Staiano, A. (2016), 'Intrinsic dimension estimation: Advances and open problems', *Information Sciences* **328**, 26–41.

Chen, M., Weinberger, K. Q., Xu, Z. & Sha, F. (2015), 'Marginalizing stacked linear denoising autoencoders', *The Journal of Machine Learning Research* **16**(1), 3849–3875.

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.

Cottrell, G. W. (1985), Learning internal representations from gray-scale images: An example of extensional programming, *in* 'Proceedings Ninth Annual Conference of the Cognitive Science Society, Irvine, CA.', pp. 462–473.

Dai, B., Wang, Y., Aston, J., Hua, G. & Wipf, D. (2018), 'Connections with robust PCA and the role of emergent sparsity in variational autoencoder models', *The Journal of Machine Learning Research* **19**(1), 1573–1614.

Deng, J., Frühholz, S., Zhang, Z. & Schuller, B. (2017), 'Recognizing emotions from whispered speech based on acoustic feature transfer learning', *IEEE Access* **5**, 5235–5246.

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

Diallo, B., Hu, J., Li, T., Khan, G. A., Liang, X. & Zhao, Y. (2021), 'Deep embedding clustering based on contractive autoencoder', *Neurocomputing* **433**, 96–107.

Dua, D. & Graff, C. (2017), 'UCI Machine Learning Repository'.
URL: *http://archive.ics.uci.edu/ml*

Facco, E., d'Errico, M., Rodriguez, A. & Laio, A. (2017), 'Estimating the intrinsic dimension of datasets by a minimal neighborhood information', *Scientific reports* **7**(1), 1–8.

Filzmoser, P., Gschwandtner, M. & Todorov, V. (2012), 'Review of sparse methods in regression and classification with application to chemometrics', *Journal of Chemometrics* **26**(3-4), 42–51.

Fukunaga, K. (1982), 'Intrinsic dimensionality extraction', *Handbook of statistics* **2**, 347–360.

Fukunaga, K. & Olsen, D. R. (1971), 'An algorithm for finding intrinsic dimensionality of data', *IEEE Transactions on Computers* **100**(2), 176–183.

Gao, Y., Shi, B., Dong, B., Chen, Y., Mi, L., Huang, Z. & Shi, Y. (2020), RVAE-ABFA: robust anomaly detection for highdimensional data using variational autoencoder, *in* '2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)', IEEE, pp. 334–339.

Ghods, A. & Cook, D. J. (2021), 'A survey of deep network techniques all classifiers can adopt', *Data mining and knowledge discovery* **35**(1), 46–87.

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

Gouk, H., Frank, E., Pfahringer, B. & Cree, M. J. (2021), 'Regularisation of neural networks by enforcing lipschitz continuity', *Machine Learning* **110**(2), 393–416.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 770–778.

Hinton, G. E. & Salakhutdinov, R. R. (2006), 'Reducing the dimensionality of data with neural networks', *Science* **313**(5786), 504–507.

Ho, K., Leung, C.-S. & Sum, J. (2010), 'Objective functions of online weight noise injection training algorithms for MLPs', *IEEE transactions on neural networks* **22**(2), 317–323.

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

Hou, Z., Liu, X., Dong, Y., Wang, C., Tang, J. et al. (2022), Graphmae: Self-supervised masked graph autoencoders, *in* 'Proceedings o the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining', pp. 594–604.

Ikeda, Y., Tajiri, K., Nakano, Y., Watanabe, K. & Ishibashi, K. (2018), 'Estimation of dimensions contributing to detected anomalies with variational autoencoders', *arXiv preprint arXiv:1811.04576* .

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L. & Muller, P.-A. (2019), 'Deep learning for time series classification: a review', *Data mining and knowledge discovery* **33**(4), 917–963.

Janakarajan, N., Born, J. & Manica, M. (2022), A fully differentiable set autoencoder, *in* 'Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining', pp. 3061–3071.

Jolliffe, I. (2002), *Principal Component Analysis*, 2 edn, Springer Verlag.

Jordan, M. I. & Mitchell, T. M. (2015), 'Machine learning: Trends, perspectives, and prospects', *Science* **349**(6245), 255–260.

Kärkkäinen, T. (2002), 'MLP in layer-wise form with applications to weight decay', *Neural Computation* **14**(6), 1451–1480.

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. (2022), On the role of Taylor's formula in machine learning, *in* 'Impact of scientific computing on science and society', Springer Nature. (18 pages, to appear).

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

Kärkkäinen, T. & Rasku, J. (2020), Application of a knowledge discovery process to study instances of capacitated vehicle routing problems, *in* 'Computation and Big Data for Transport, Chapter 6', Computational Methods in Applied Sciences, Springer-Verlag, pp. 1–25.

Kärkkäinen, T. & Saarela, M. (2015), Robust principal component analysis of data with missing values, *in* 'International Workshop on Machine Learning and Data Mining in Pattern Recognition', Springer, pp. 140–154.

Khajenezhad, A., Madani, H. & Beigy, H. (2020), 'Masked autoencoder for distribution estimation on small structured data sets', *IEEE Transactions on Neural Networks and Learning Systems* . Early Access, to appear.

Khodayar, M., Kaynak, O. & Khodayar, M. E. (2017), 'Rough deep neural architecture for short-term wind speed forecasting', *IEEE Transactions on Industrial Informatics* **13**(6), 2770–2779.

Kim, S., Noh, Y.-K. & Park, F. C. (2020), 'Efficient neural network compression via transfer learning for machine vision inspection', *Neurocomputing* **413**, 294–304.

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.

LeCun, Y., Bengio, Y. & Hinton, G. (2015), 'Deep learning', *Nature* **521**(7553), 436–444.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E. & Jackel, L. D. (1990), Handwritten digit recognition with a back-propagation network, *in* 'Advances in Neural Information Processing Systems', pp. 396–404.

Lee, J. A. & Verleysen, M. (2007), *Nonlinear dimensionality reduction*, Springer Science & Business Media.

Li, L., Franklin, M., Girguis, M., Lurmann, F., Wu, J., Pavlovic, N., Breton, C., Gilliland, F. & Habre, R. (2020), 'Spatiotemporal imputation of MAIAC AOD using deep learning with downscaling', *Remote sensing of environment* **237**, 111584.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y. & Alsaadi, F. E. (2017), 'A survey of deep neural network architectures and their applications', *Neurocomputing* **234**, 11–26.

Ma, Q., Lee, W.-C., Fu, T.-Y., Gu, Y. & Yu, G. (2020), 'Midia: exploring denoising autoencoders for missing data imputation', *Data Mining and Knowledge Discovery* **34**(6), 1859–1897.

McConville, R., Santos-Rodriguez, R., Piechocki, R. J. & Craddock, I. (2021), N2d:(not too) deep clustering via clustering the local manifold of an autoencoded embedding, *in* '2020 25th International Conference on Pattern Recognition (ICPR)', IEEE, pp. 5145–5152.

Min, E., Guo, X., Liu, Q., Zhang, G., Cui, J. & Long, J. (2018), 'A survey of clustering with deep learning: From the perspective of network architecture', *IEEE Access* **6**, 39501–39514.

Narayanan, S., Marks, R., Vian, J. L., Choi, J., El-Sharkawi, M. & Thompson, B. B. (2002), Set constraint discovery: missing sensor data restoration using autoassociative regression machines, *in* 'Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02', Vol. 3, IEEE, pp. 2872–2877.

Navarro, G., Paredes, R., Reyes, N. & Bustos, C. (2017), 'An empirical evaluation of intrinsic dimension estimators', *Information Systems* **64**, 206–218.

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

Probst, M. & Rothlauf, F. (2020), 'Harmless overfitting: Using denoising autoencoders in estimation of distribution algorithms', *Journal of Machine Learning Research* **21**(78), 1–31.

Ryu, S., Kim, M. & Kim, H. (2020), 'Denoising autoencoder-based missing value imputation for smart meters', *IEEE Access* **8**, 40656–40666.

Sainath, T. N., Kingsbury, B. & Ramabhadran, B. (2012), Auto-encoder bottleneck features using deep belief networks, *in* '2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)', IEEE, pp. 4153–4156.

Sangeetha, M. & Kumaran, M. S. (2020), 'Deep learning-based data imputation on time-variant data using recurrent neural network', *Soft Computing* **24**(17), 13369–13380.

Schmidhuber, J. (2015), 'Deep learning in neural networks: An overview', *Neural Networks* **61**, 85–117.

Song, L., Ma, H., Wu, M., Zhou, Z. & Fu, M. (2018), A brief survey of dimension reduction, *in* 'International Conference on Intelligent Science and Big Data Engineering', Springer, pp. 189–200.

Sun, C., Ma, M., Zhao, Z., Tian, S., Yan, R. & Chen, X. (2018), 'Deep transfer learning based on sparse autoencoder for remaining useful life prediction of tool in manufacturing', *IEEE transactions on industrial informatics* **15**(4), 2416–2425.

Sun, M., Wang, H., Liu, P., Huang, S. & Fan, P. (2019), 'A sparse stacked denoising autoencoder with optimized transfer learning applied to the fault diagnosis of rolling bearings', *Measurement* **146**, 305–314.

Sun, M., Zhang, X., Zheng, T. F. et al. (2015), 'Unseen noise estimation using separable deep auto encoder for speech enhancement', *IEEE/ACM Transactions on Audio, Speech, and Language Processing* **24**(1), 93–104.

Takahashi, H., Iwata, T., Kumagai, A., Kanai, S., Yamada, M., Yamanaka, Y. & Kashima, H. (2022), Learning optimal priors for task-invariant representations in variational autoencoders, *in* 'Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining', pp. 1739–1748.

Thorndike, R. L. (1953), 'Who belongs in the family?', *Psychometrika* **18**(4), 267–276.

Tran, L., Liu, X., Zhou, J. & Jin, R. (2017), Missing modalities imputation via cascaded residual autoencoder, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition', pp. 1405–1414.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A. & Bottou, L. (2010), 'Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.', *Journal of machine learning research* **11**(12).

Vogelstein, J. T., Bridgeford, E. W., Tang, M., Zheng, D., Douville, C., Burns, R. & Maggioni, M. (2021), 'Supervised dimensionality reduction for big data', *Nature communications* **12**(1), 1–9.

Wang, Y., Yao, H. & Zhao, S. (2016), 'Auto-encoder based dimensionality reduction', *Neurocomputing* **184**, 232–242.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Philip, S. Y. (2021), 'A comprehensive survey on graph neural networks', *IEEE Transactions on Neural Networks and Learning Systems* **32**(1), 4–24.

Yoo, J., Jeon, H., Jung, J. & Kang, U. (2022), Accurate node feature estimation with structured variational graph autoencoder, *in* 'Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining', pp. 2336–2346.

Zhang, B. & Qian, J. (2021), 'Autoencoder-based unsupervised clustering and hashing', *Applied Intelligence* **51**(1), 493–505.

Zhao, J., Nie, Y., Ni, S. & Sun, X. (2020), 'Traffic data imputation and prediction: An efficient realization of deep learning', *IEEE Access* **8**, 46713–46722.

Zhao, Y., Hao, K., Tang, X.-s., Chen, L. & Wei, B. (2021), 'A conditional variational autoencoder based self-transferred algorithm for imbalanced classification', *Knowledge-Based Systems* **218**, 106756: 1–10.

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 $\mathbf{X} = \cup_{i=1}^{k} \mathbf{B}_i$

```
while |X| ≥ k do
    Let x_1 be a random observation from X
    Let x_2, ..., x_k be k − 1 closest neighbors of x_1 from X
    B_i = B_i ∪ {x_i} and X = X \ {x_i}, i = 1, ..., k
end while
Place the remaining observations from X into different subsets B_i, i = 1, ..., |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.

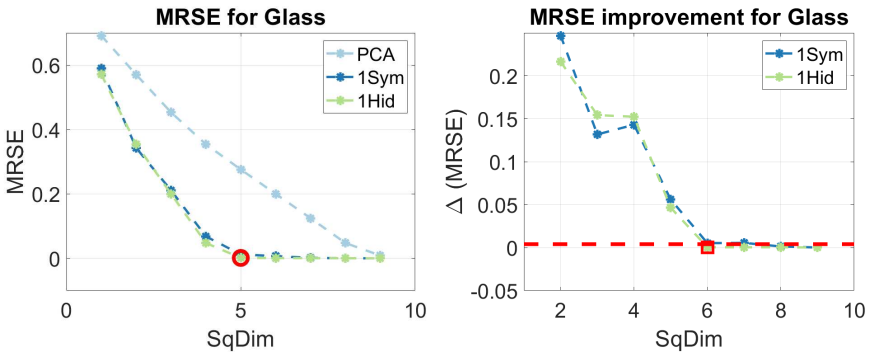

Figure 1: Identification of the intrinsic dimension for Glass.

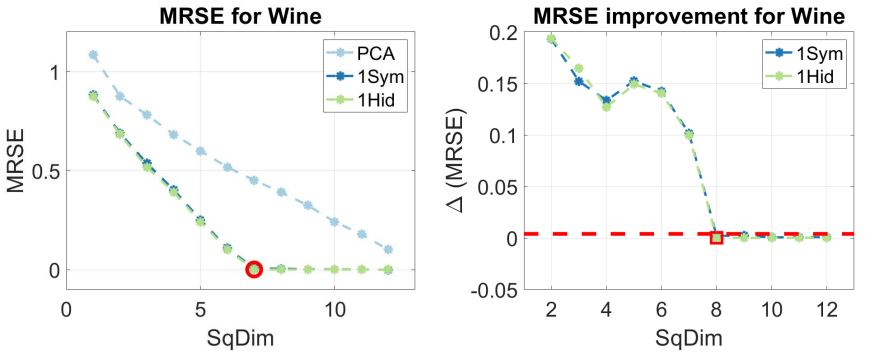

Figure 2: Identification of the intrinsic dimension for Wine.

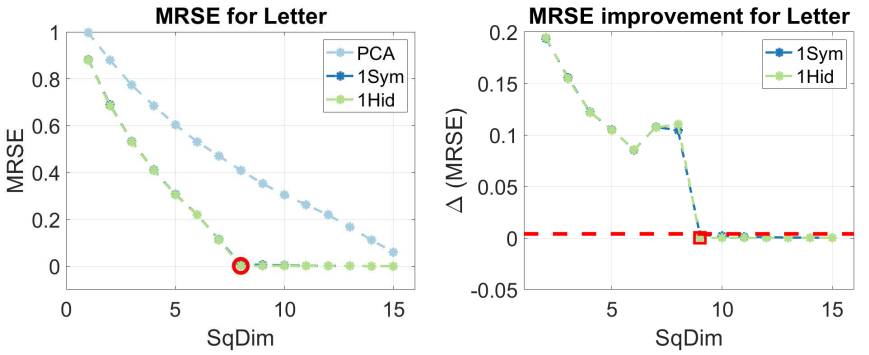

Figure 3: Identification of the intrinsic dimension for Letter.

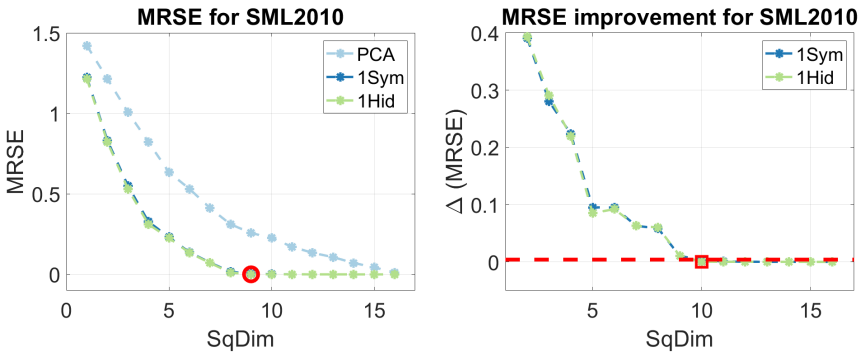

Figure 4: Identification of the intrinsic dimension for SML2010.

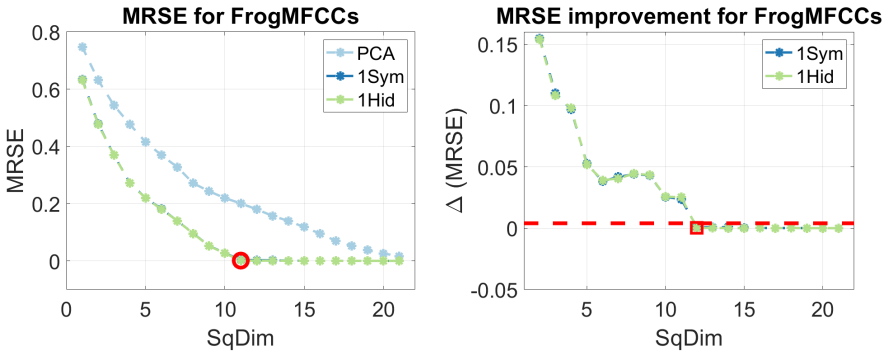

Figure 5: Identification of the intrinsic dimension for FrogMFCCs.

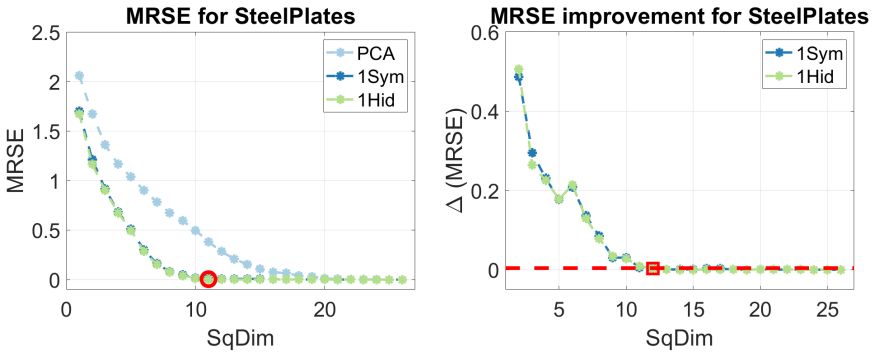

Figure 6: Identification of the intrinsic dimension for SteelPlates.

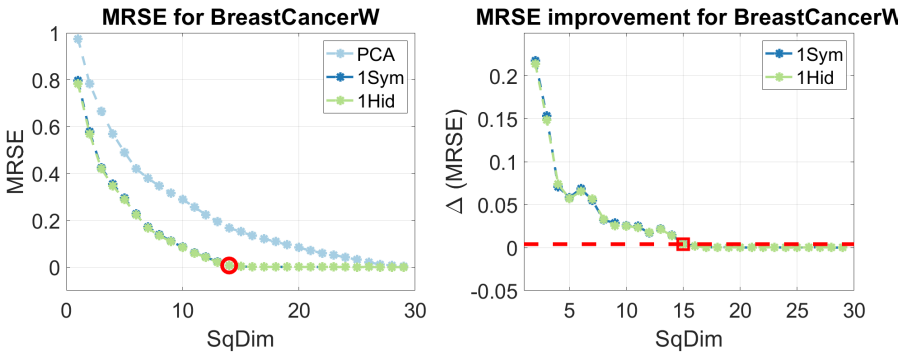

Figure 7: Identification of the intrinsic dimension for BreastCancerW.

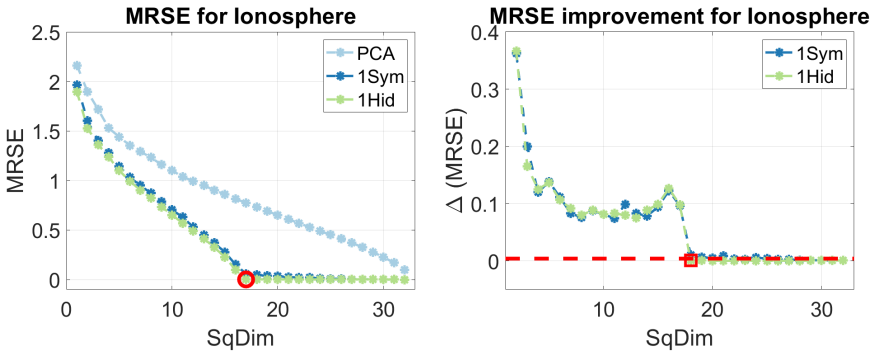

Figure 8: Identification of the intrinsic dimension for Ionosphere.

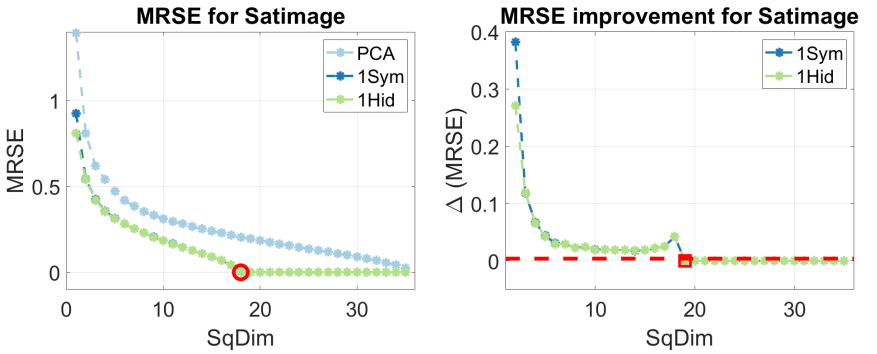

Figure 9: Identification of the intrinsic dimension for Satimage.

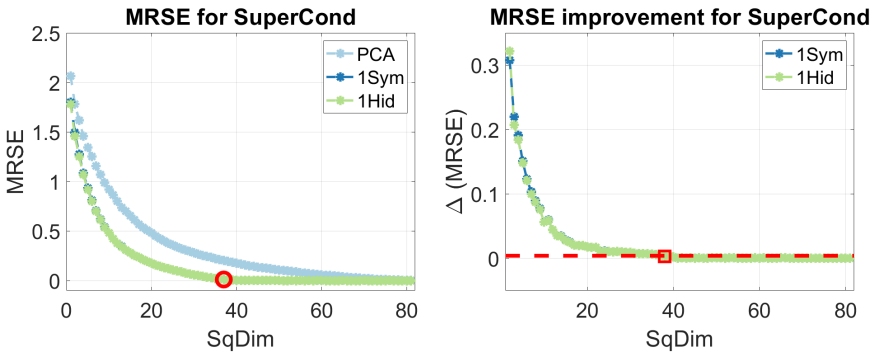

Figure 10: Identification of the intrinsic dimension for SuperCond.

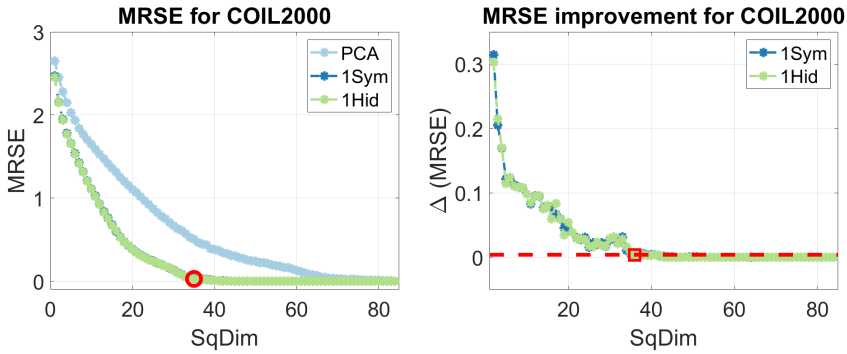

Figure 11: Identification of the intrinsic dimension for COIL2000.

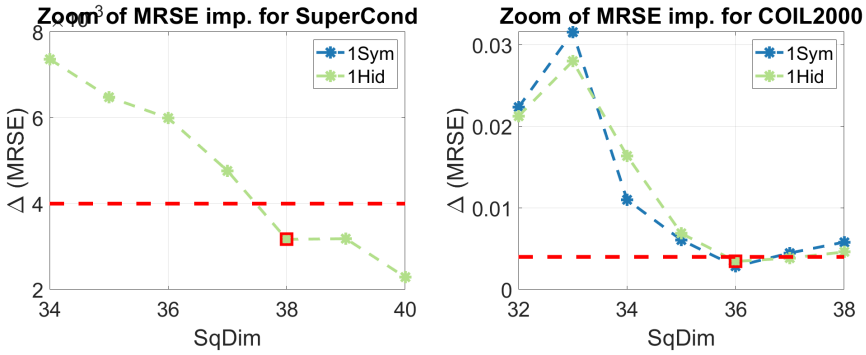

Figure 12: Zooms of the identification of the intrinsic dimension for SuperCond (left) and COIL2000 (right).

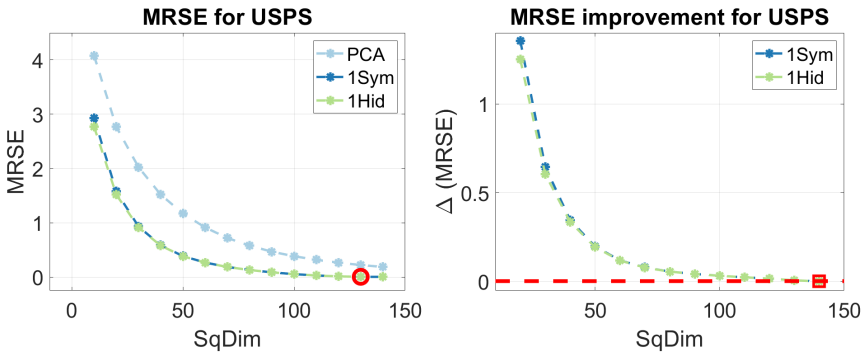

Figure 13: Identification of the intrinsic dimension for USPS (τ = 3e-3).

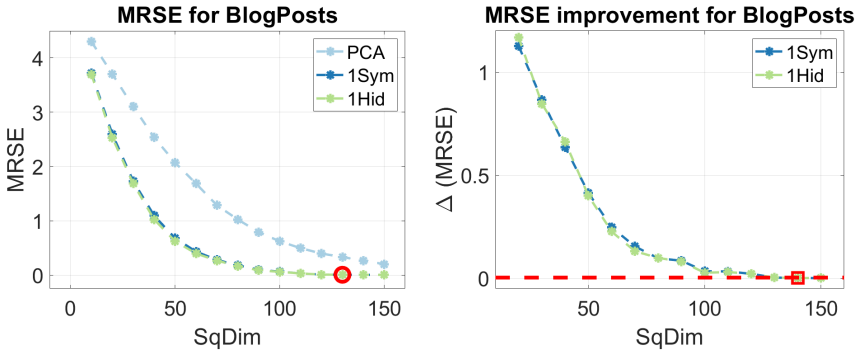

Figure 14: Identification of the intrinsic dimension for BlogPosts (τ = 3e-3).

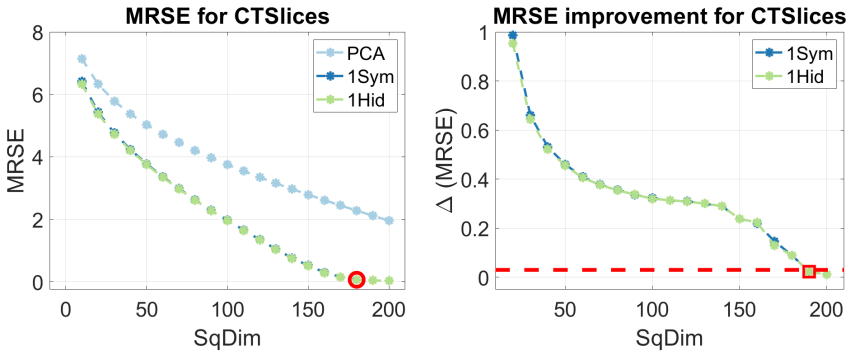

Figure 15: Identification of the intrinsic dimension for CTSlices (τ = 3e-2).

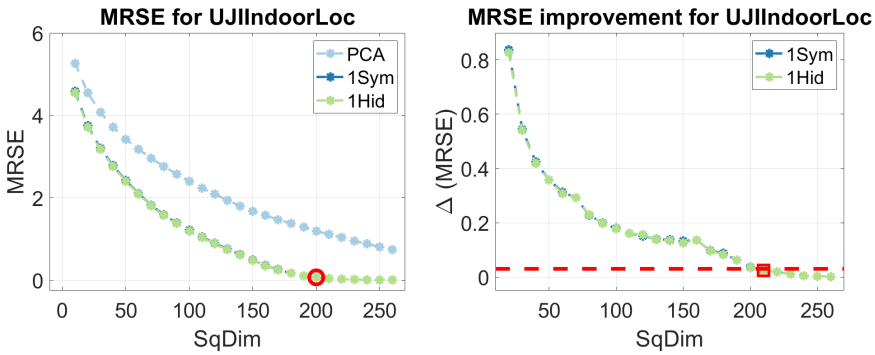

Figure 16: Identification of the intrinsic dimension for UJIIndoor (τ = 3e-2).

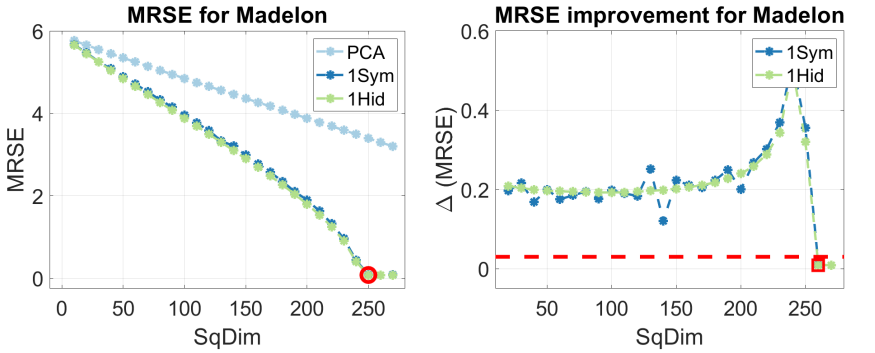

Figure 17: Identification of the intrinsic dimension for Madelon (τ = 3e-2).

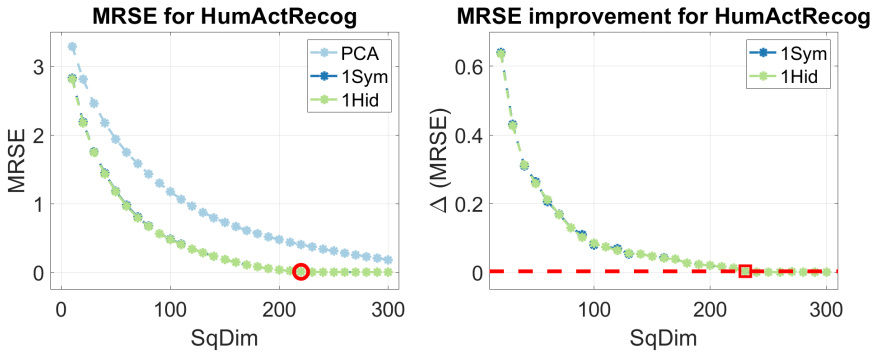

Figure 18: Identification of the intrinsic dimension for HumActRecog (τ = 3e-3).

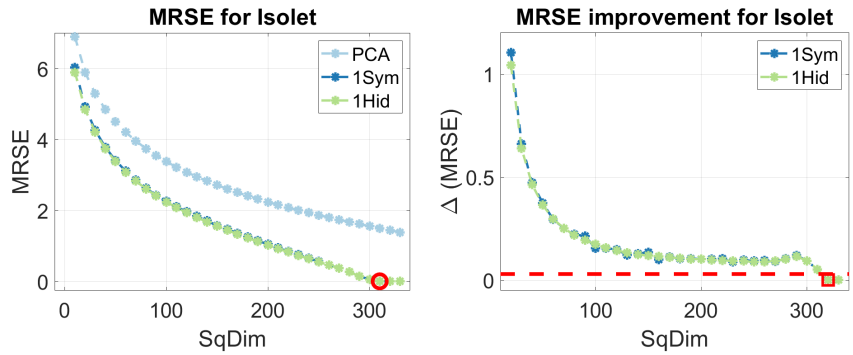

Figure 19: Identification of the intrinsic dimension for Isolet (τ = 3e-2).

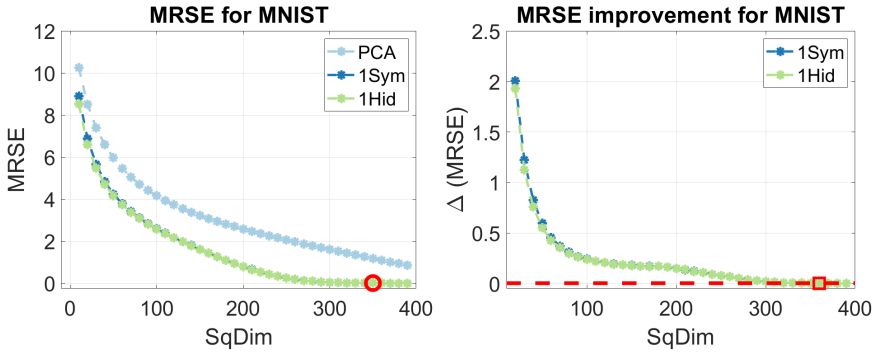

Figure 20: Identification of the intrinsic dimension for MNIST (τ = 3e-3).

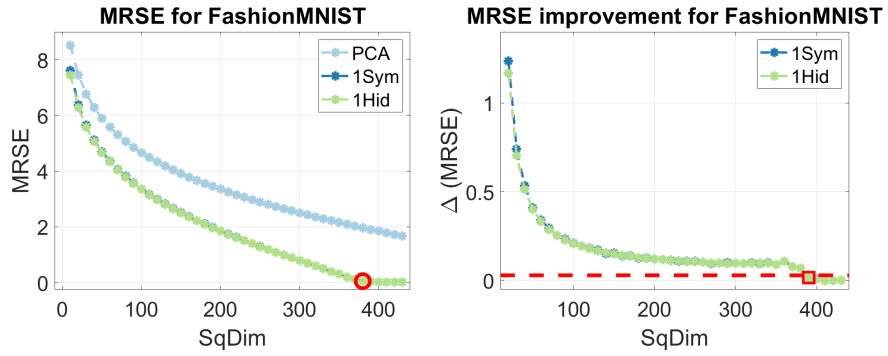

Figure 21: Identification of the intrinsic dimension for FashMNIST (τ = 3e-2).

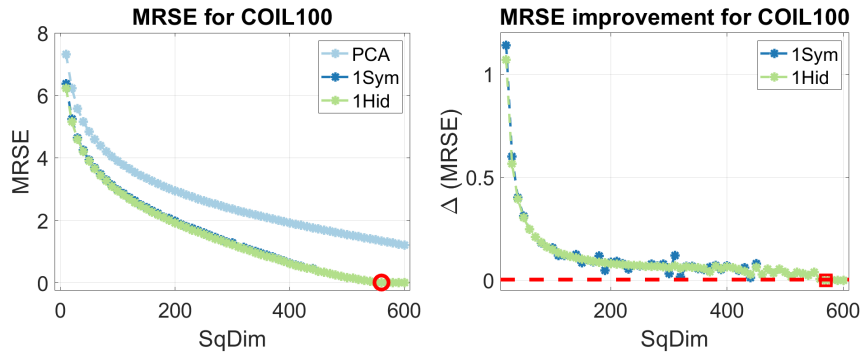

Figure 22: Identification of the intrinsic dimension for COIL100 (τ = 3e-3).

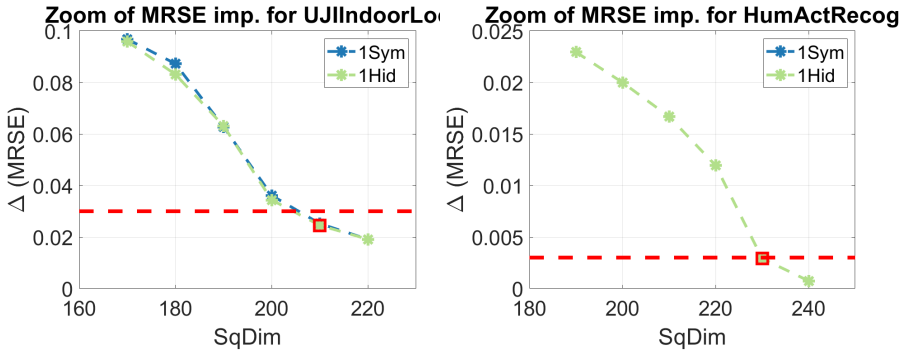

Figure 23: Zooms of the identification for UJIIndoor (left) and HumActRecog (right).

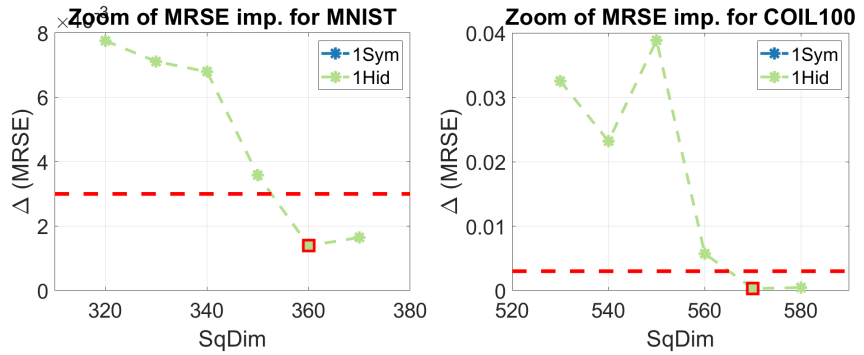

Figure 24: Zooms of the identification for MNIST (left) and COIL100 (right).

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.

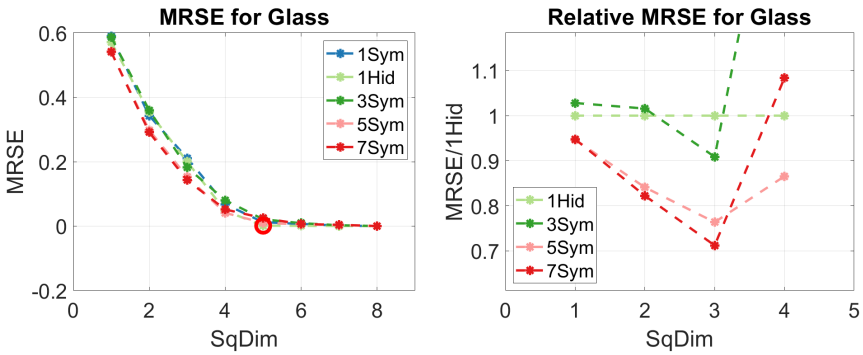

Figure 25: Relative performances for Glass.

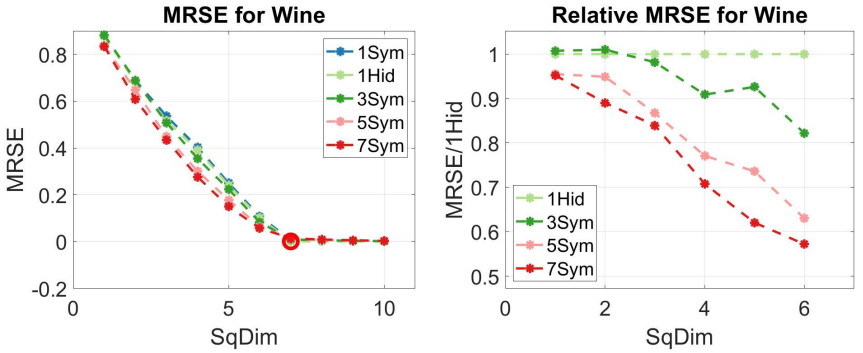

Figure 26: Relative performances for Wine.

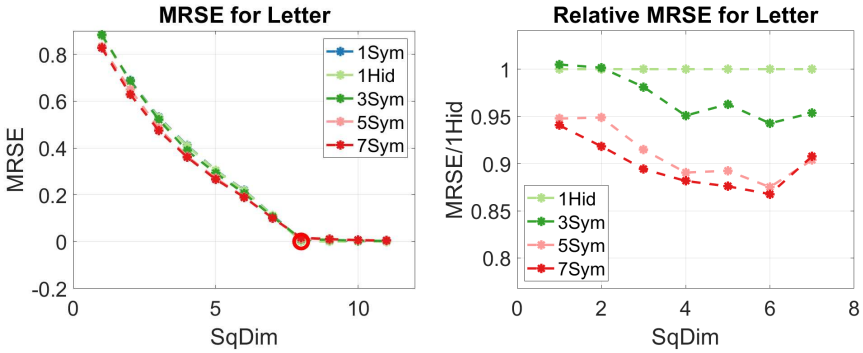

Figure 27: Relative performances for Letter.

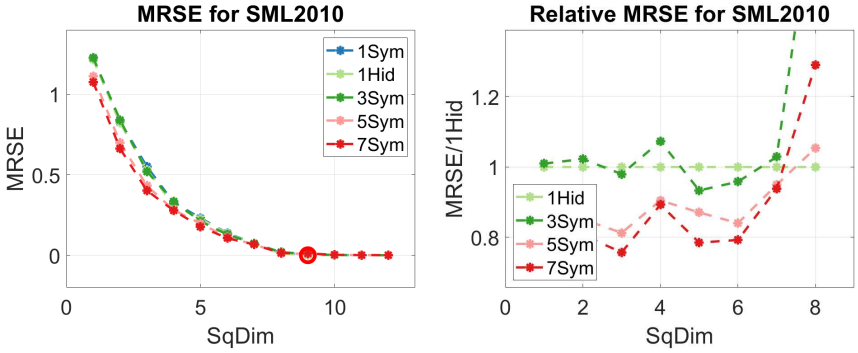

Figure 28: Relative performances for SML2010.

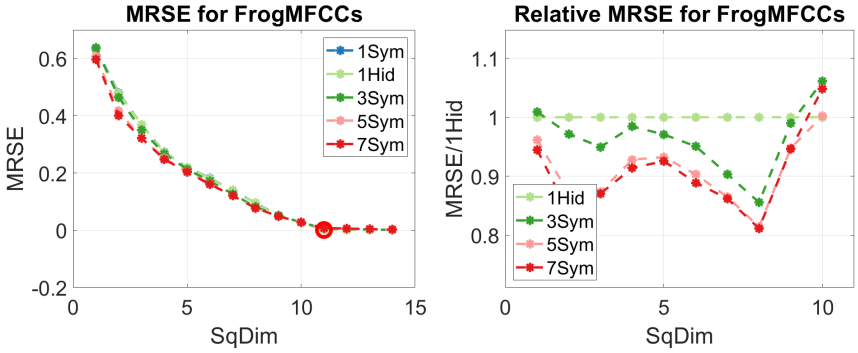

Figure 29: Relative performances for FrogMFCCs.

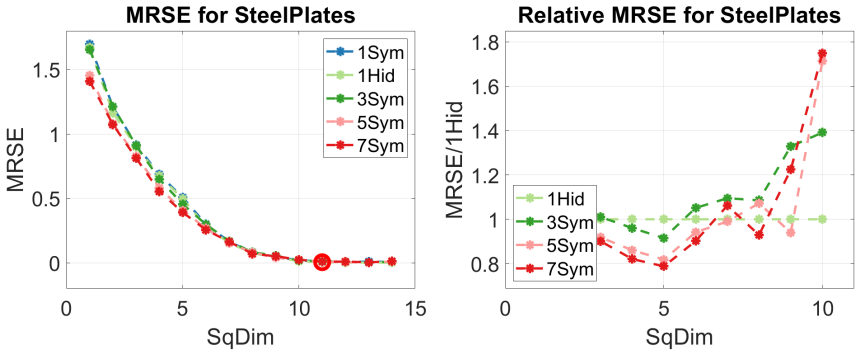

Figure 30: Relative performances for SteelPlates.

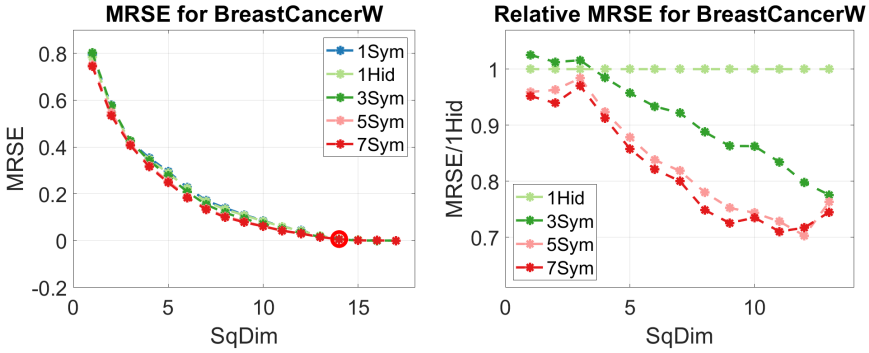

Figure 31: Relative performances for BreastCancerW.

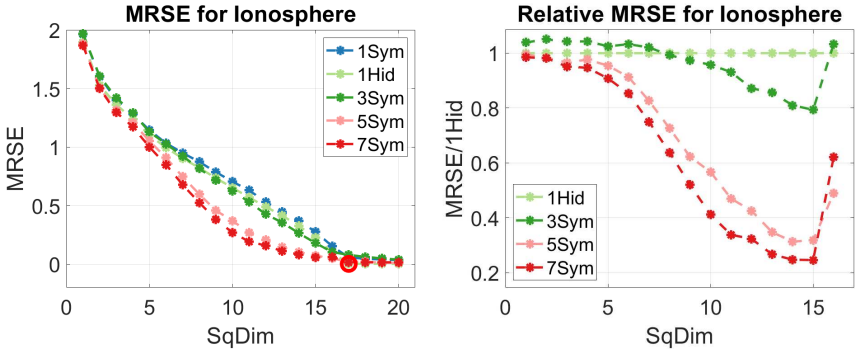

Figure 32: Relative performances for Ionosphere.

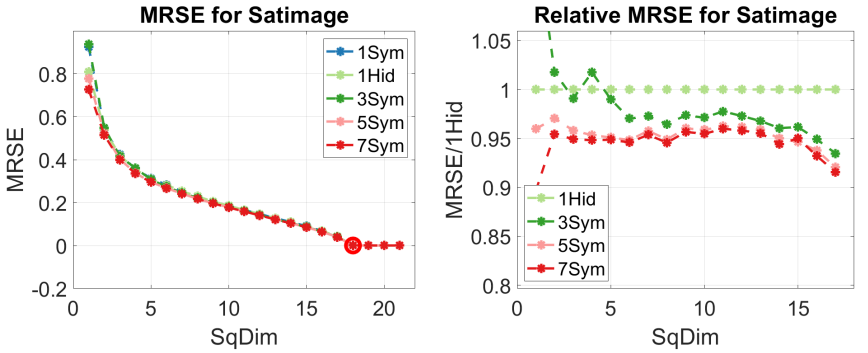

Figure 33: Relative performances for Satimage.

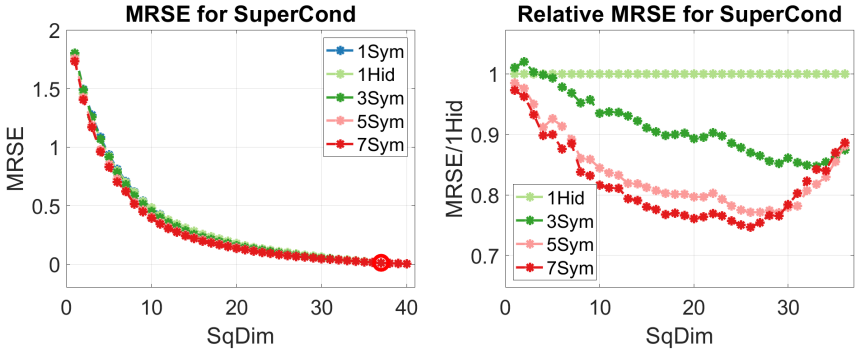

Figure 34: Relative performances for SuperCond.

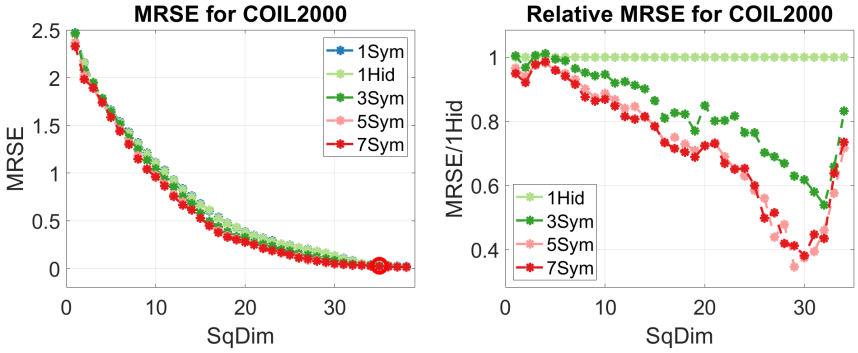

Figure 35: Relative performances for COIL2000.

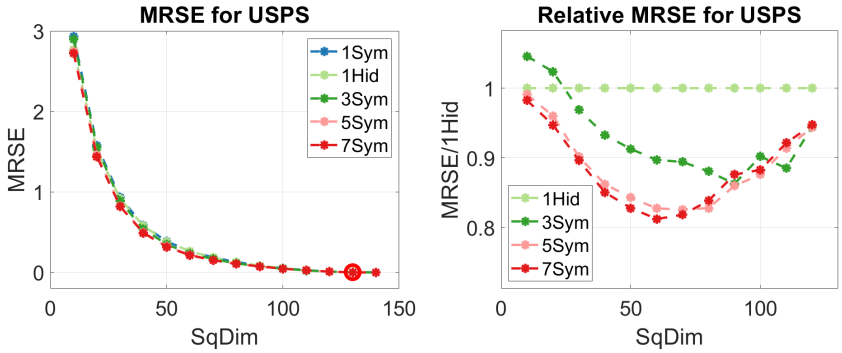

Figure 36: Relative performances for USPS.

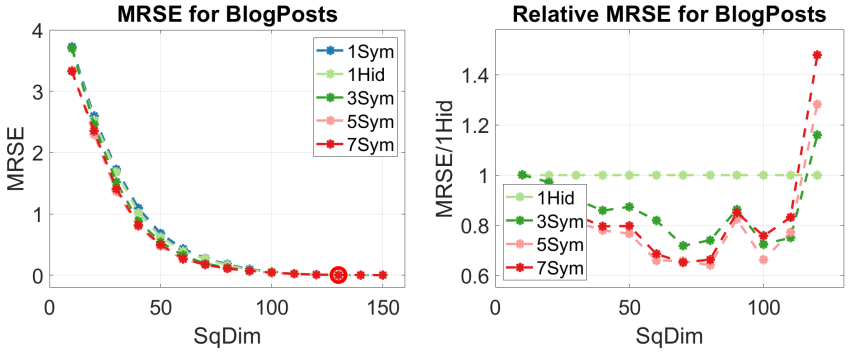

Figure 37: Relative performances for BlogPosts.

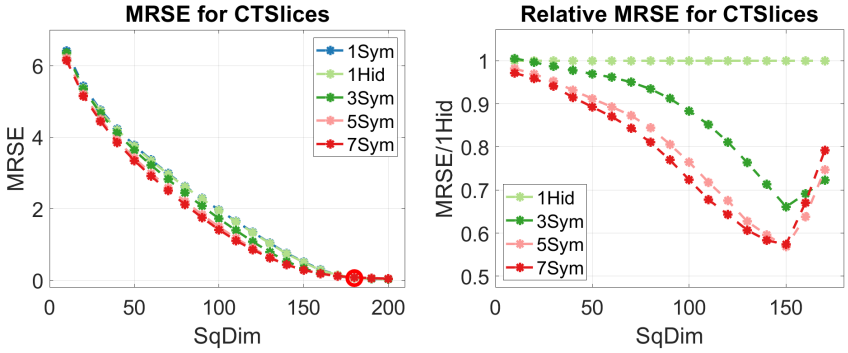

Figure 38: Relative performances for CTSlices.

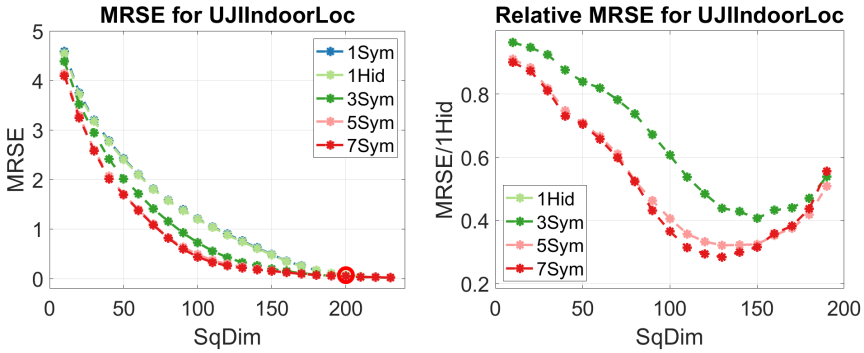

Figure 39: Relative performances for UJIIndoor.

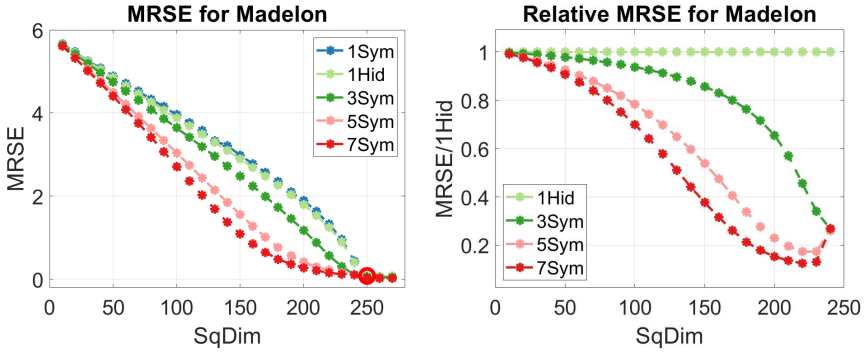

Figure 40: Relative performances for Madelon.

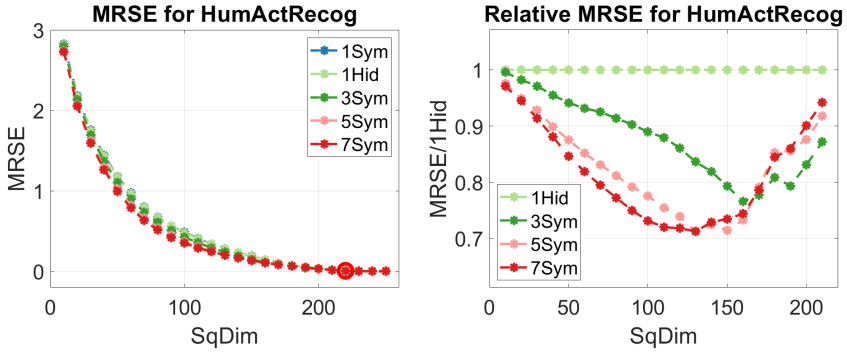

Figure 41: Relative performances for HumActRecog.

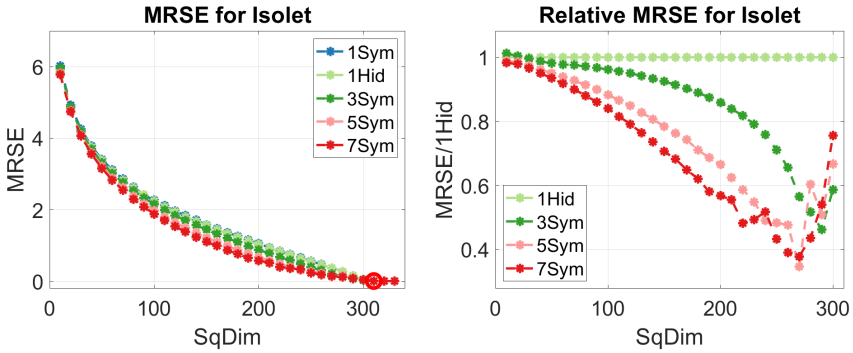

Figure 42: Relative performances for Isolet.

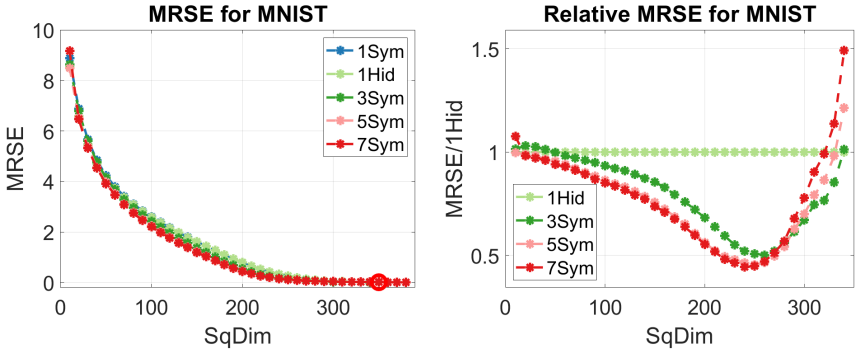

Figure 43: Relative performances for MNIST.

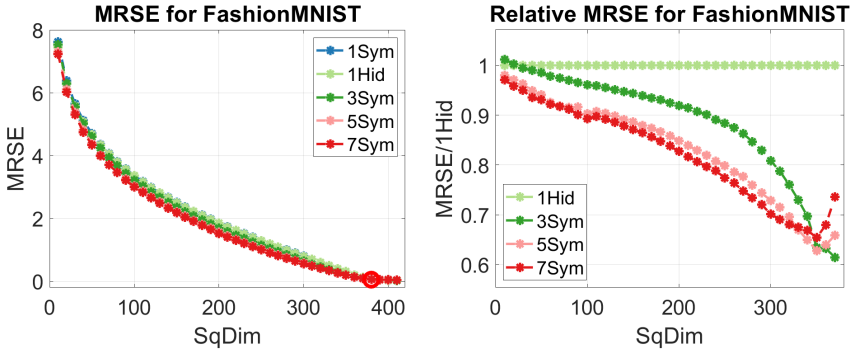

Figure 44: Relative performances for FashMNIST.

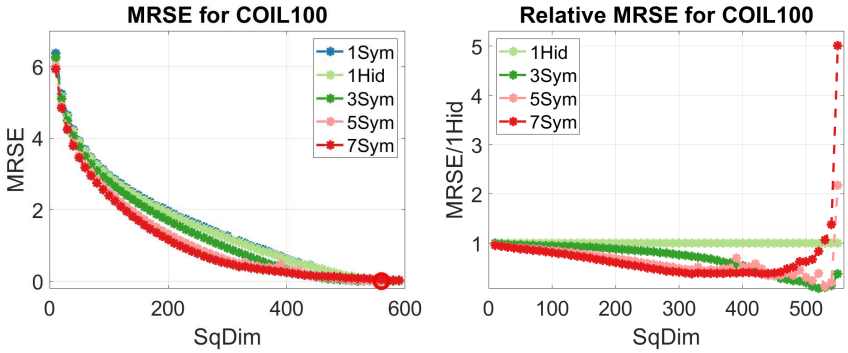

Figure 45: Relative performances for COIL100.

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.

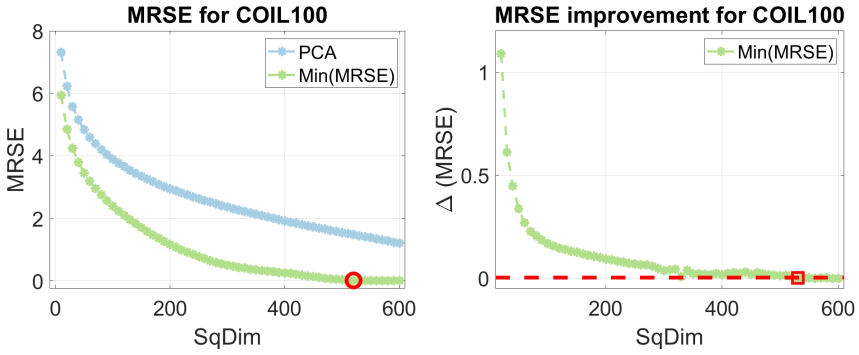

Figure 46: Identification of the ID for COIL100 from the minimum AE error.

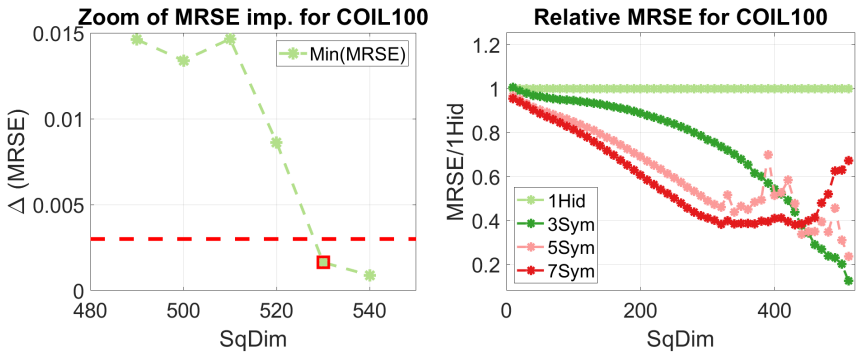

Figure 47: Zoomed identification of ID (left) and Relative performances (right) for COIL100 from the minimum AE error.

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.

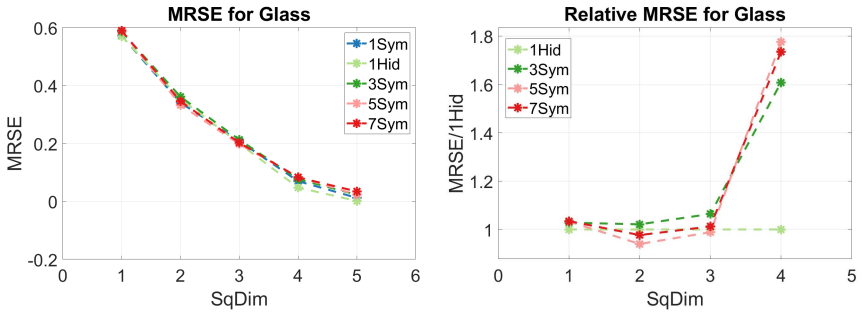

Figure 48: Comparison of all models for Glass.

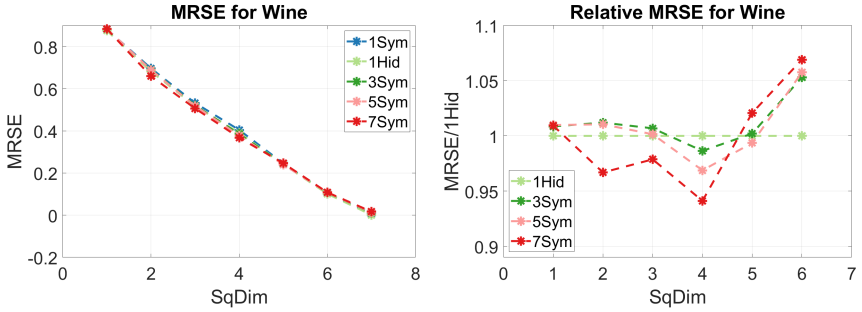

Figure 49: Comparison of all models for Wine.

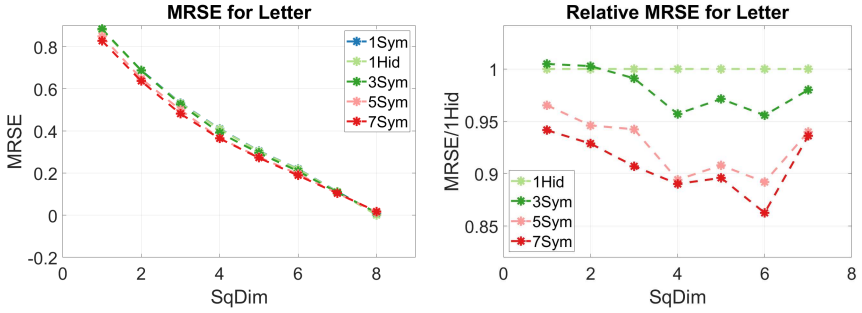

Figure 50: Comparison of all models for Letter.

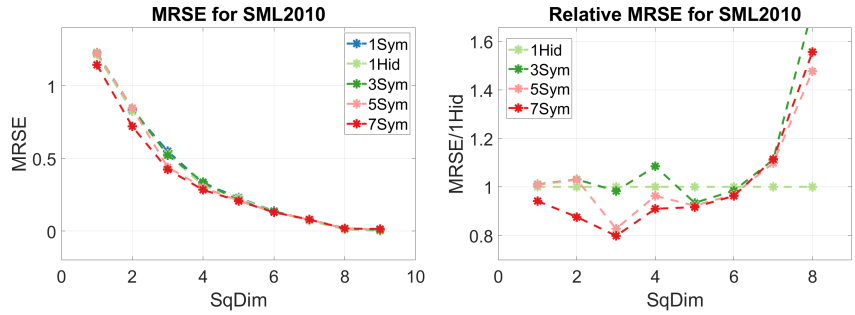

Figure 51: Comparison of all models for SML2010.

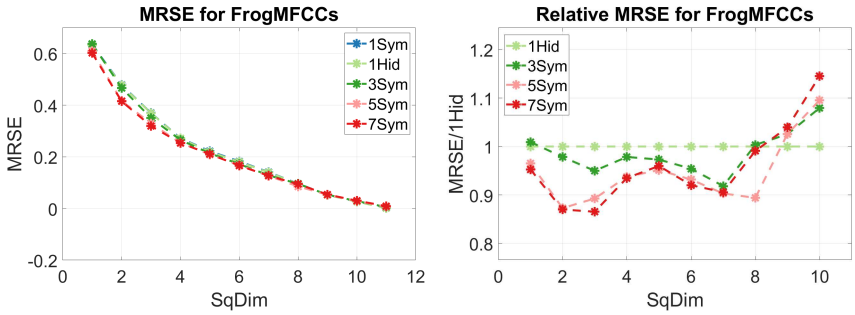

Figure 52: Comparison of all models for FrogMFCCs.

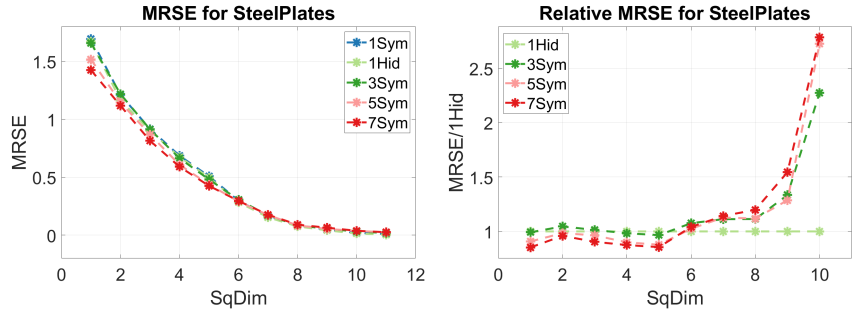

Figure 53: Comparison of all models for SteelPlates.

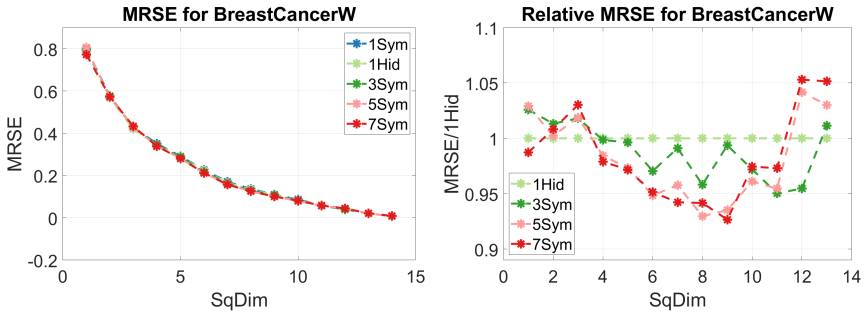

Figure 54: Comparison of all models for BreastCancerW.

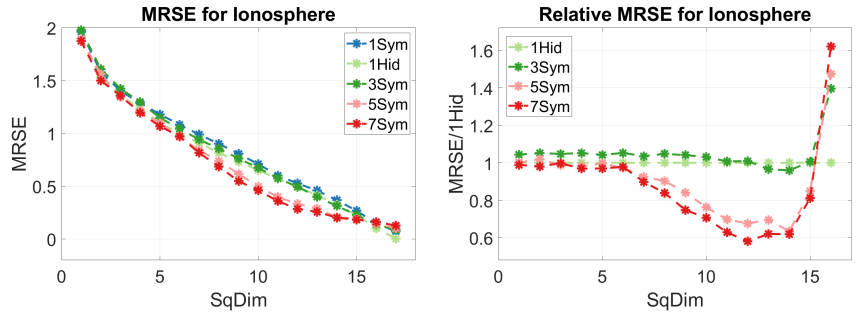

Figure 55: Comparison of all models for Ionosphere.

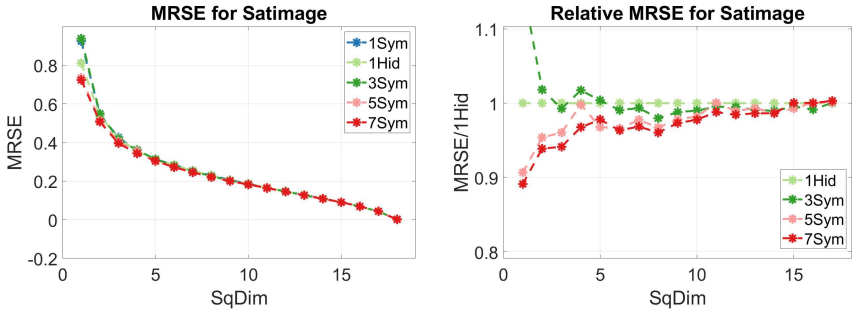

Figure 56: Comparison of all models for Satimage.

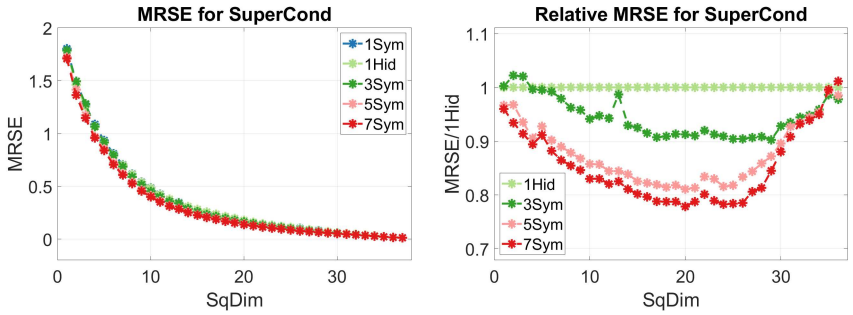

Figure 57: Comparison of all models for SuperCond.

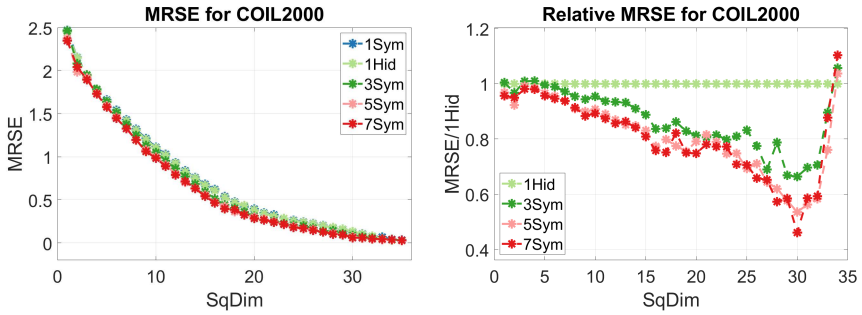

Figure 58: Comparison of all models for COIL2000.

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.

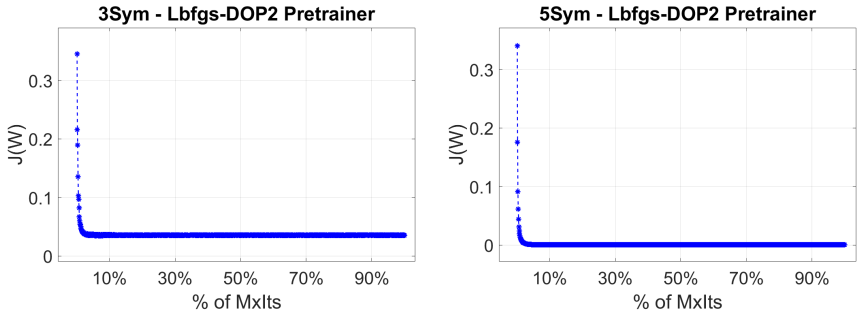

Figure 59: 3SYM and 5SYM pretraining results for L-BFGS-DOP2.

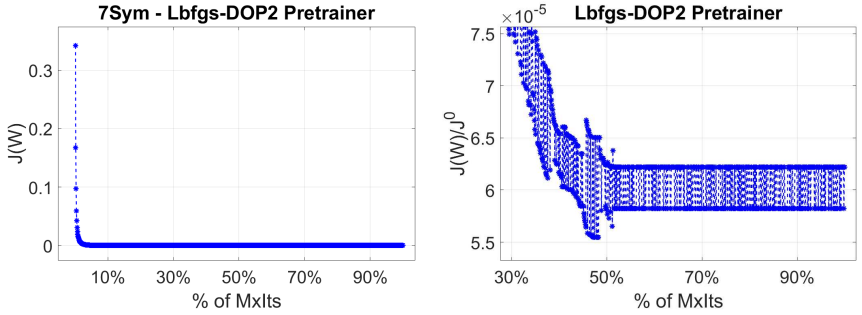

Figure 60: 7SYM pretraining results with L-BFGS-DOP2.

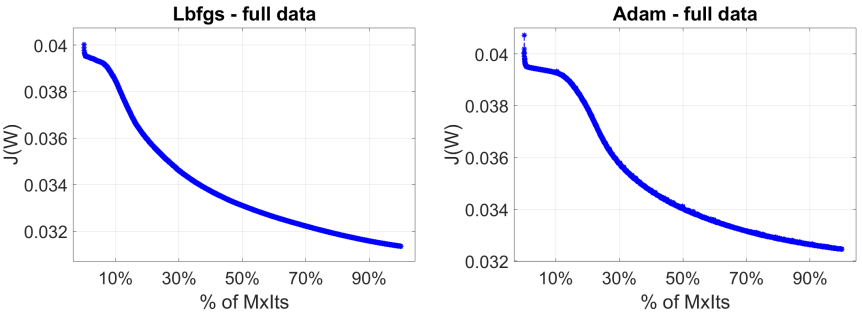

Figure 61: 3SYM results for L-BFGS and Adam finetuners with entire data.

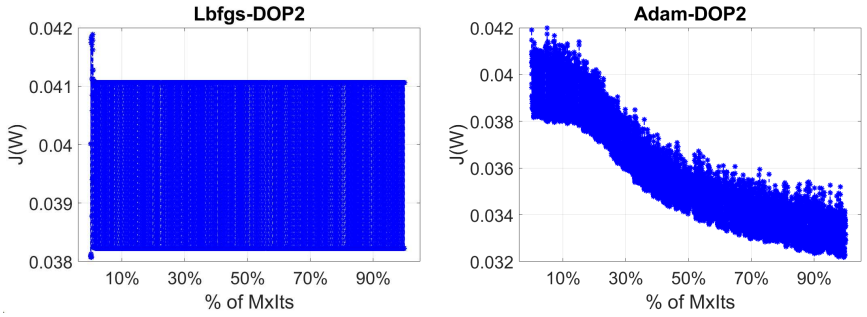

Figure 62: 3SYM results for L-BFGS-DOP2 and Adam-DOP2 finetuners.

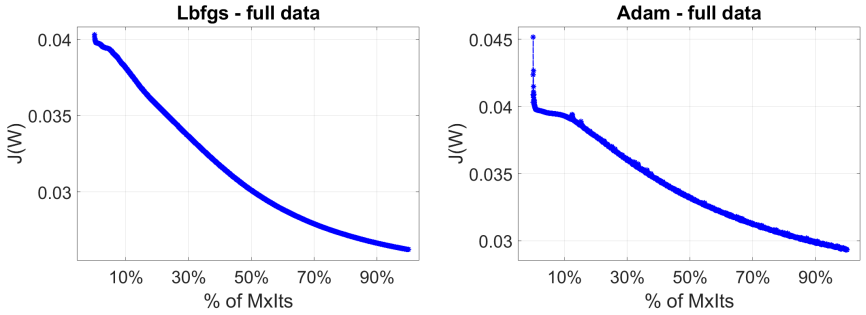

Figure 63: 5SYM results for L-BFGS and Adam finetuners with entire data.

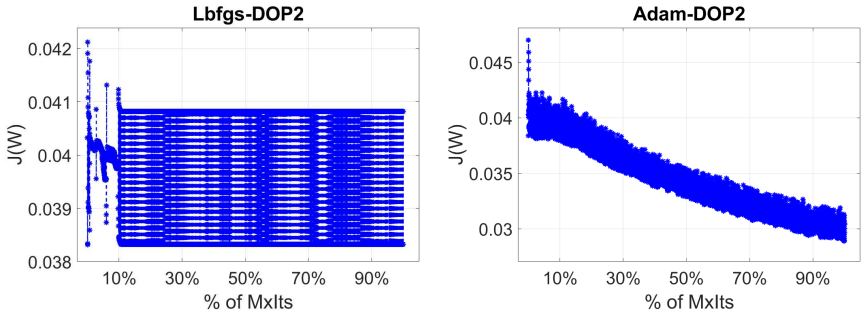

Figure 64: 5SYM results for L-BFGS-DOP2 and Adam-DOP2 finetuners.

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

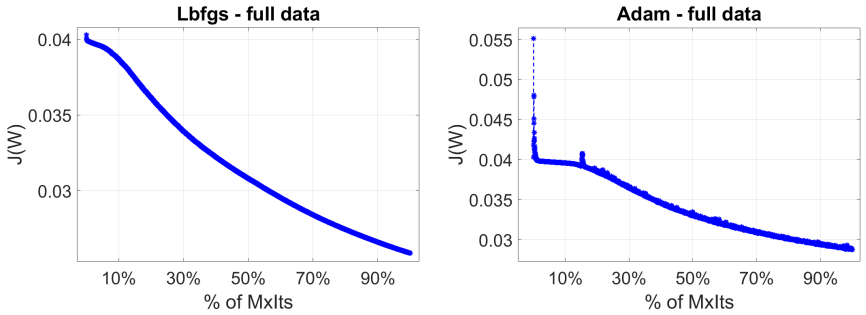

Figure 65: 7SYM results for L-BFGS and Adam finetuners with entire data.

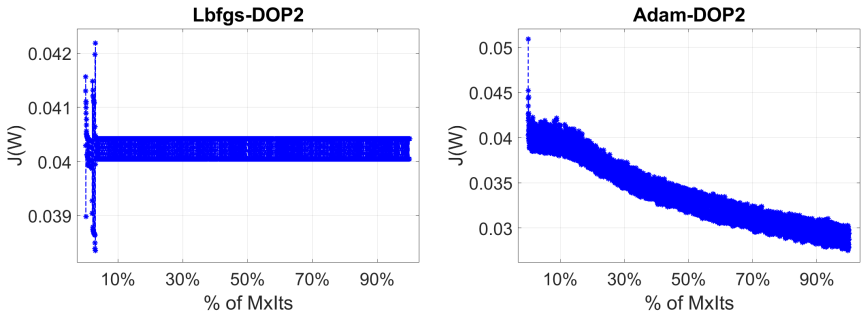

Figure 66: 7SYM results for L-BFGS-DOP2 and Adam-DOP2 finetuners.

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.

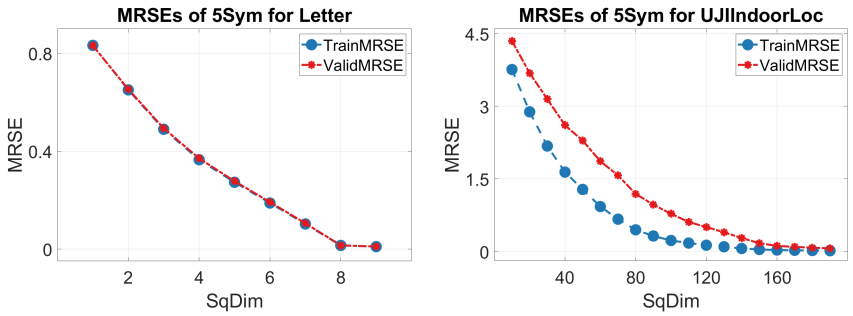

Figure 67: Generalization of Letter and UJIIndoor.

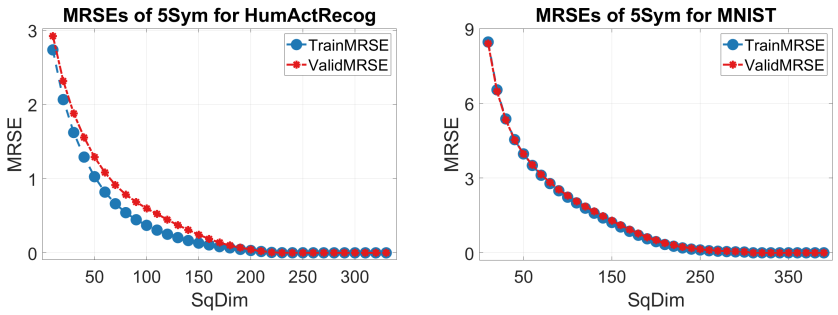

Figure 68: Generalization of HumActRecog and MNIST.

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.

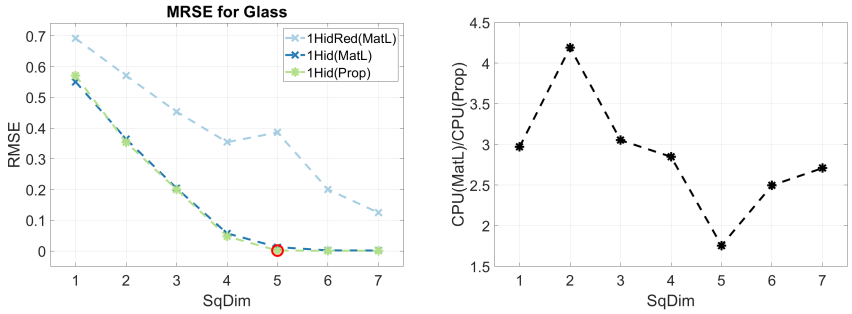

Figure 69: Comparison of forms and realizations of AE for Glass.

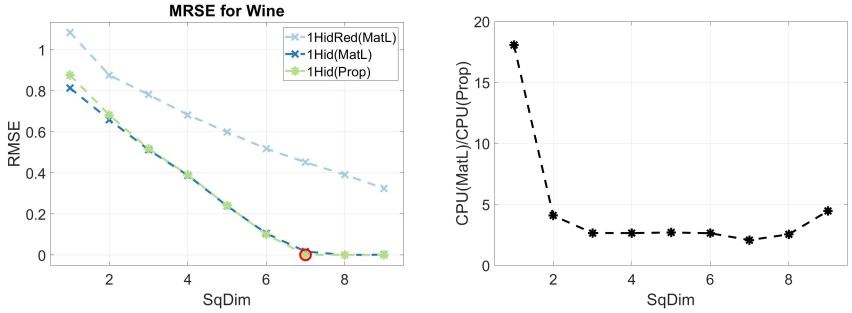

Figure 70: Comparison of forms and realizations of AE for Wine.

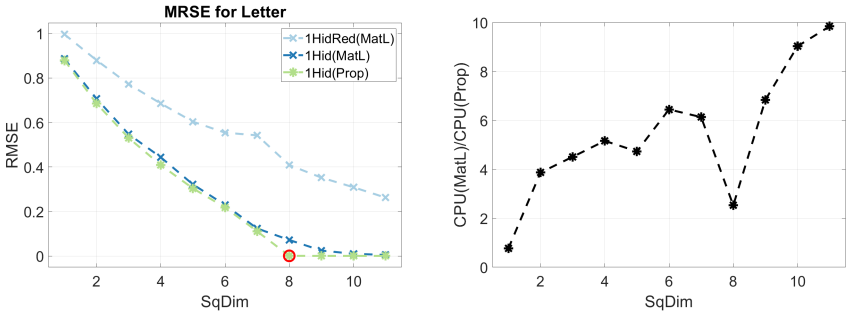

Figure 71: Comparison of forms and realizations of AE for Letter.

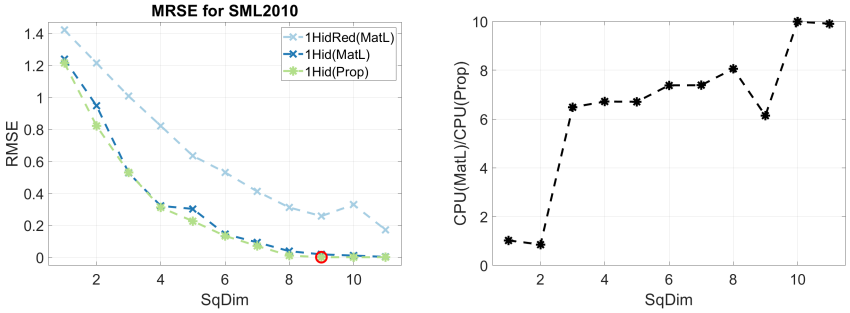

Figure 72: Comparison of forms and realizations of AE for SML2010.

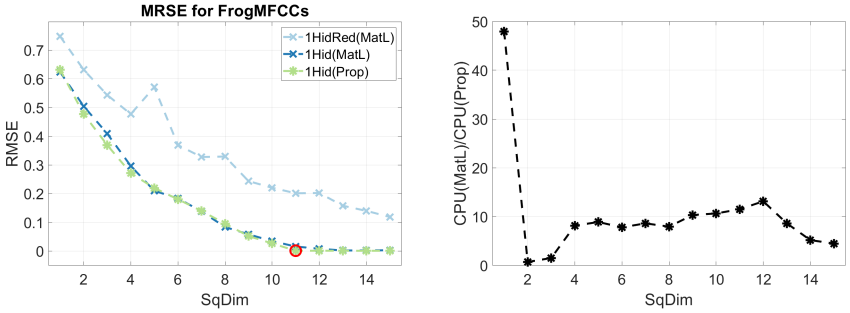

Figure 73: Comparison of forms and realizations of AE for FrogMFCCs.

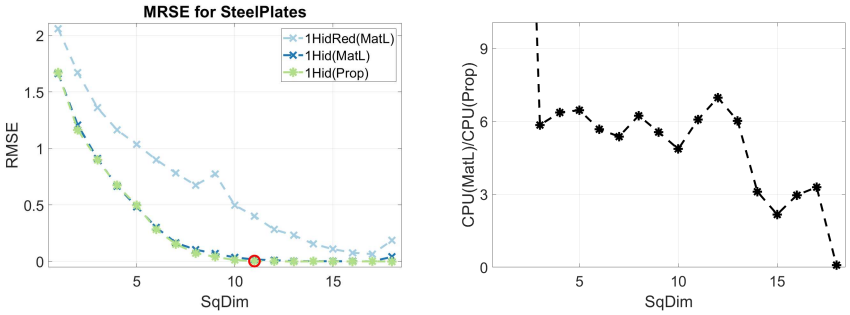

Figure 74: Comparison of forms and realizations of AE for SteelPlates.

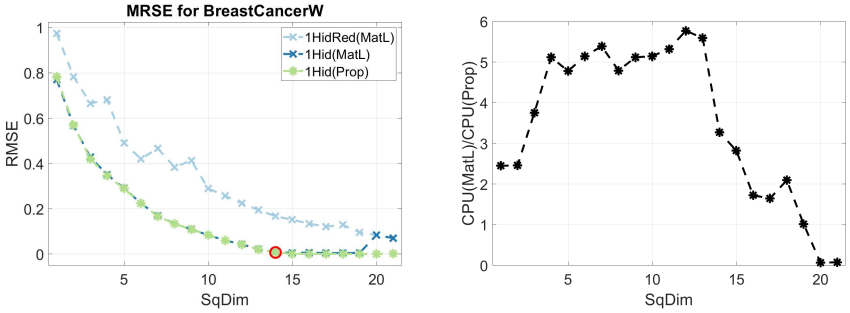

Figure 75: Comparison of forms and realizations of AE for BreastCancerW.

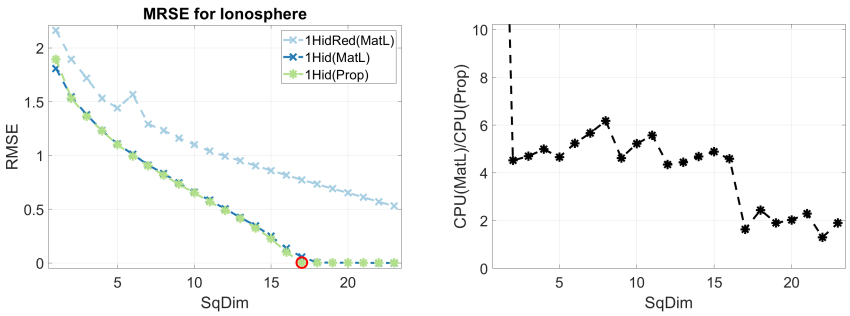

Figure 76: Comparison of forms and realizations of AE for Ionosphere.

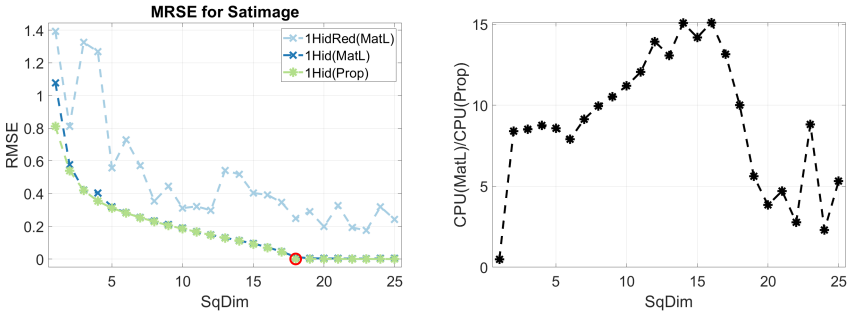

Figure 77: Comparison of forms and realizations of AE for Satimage.

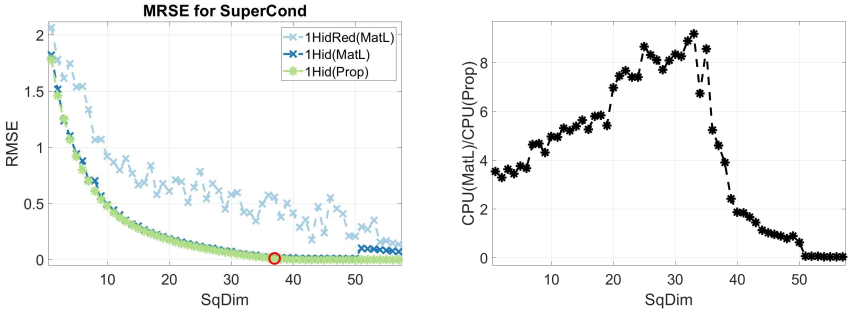

Figure 78: Comparison of forms and realizations of AE for SuperCond.

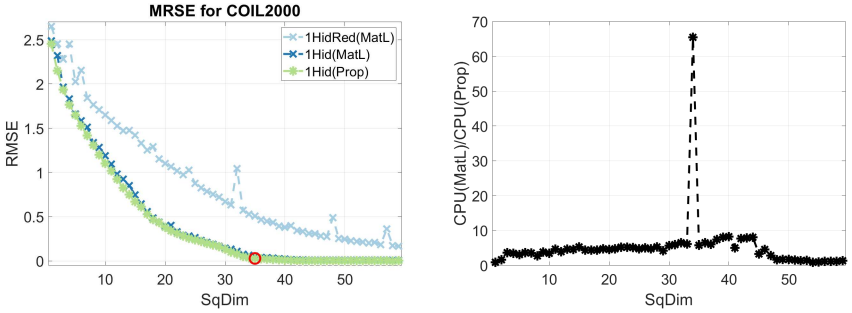

Figure 79: Comparison of forms and realizations of AE for COIL2000.

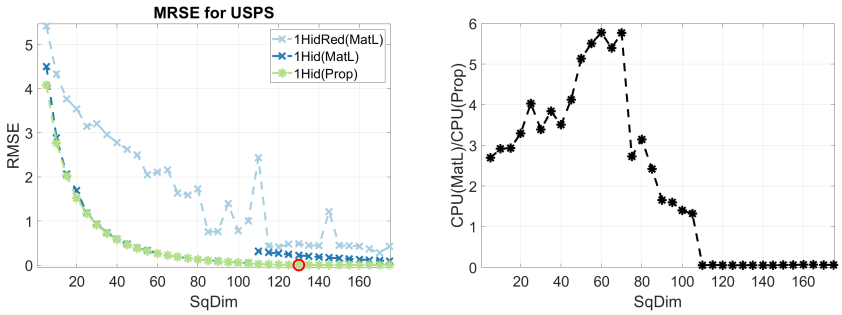

Figure 80: Comparison of forms and realizations of AE for USPS.

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.