Recovery of Linear Components: Reduced Complexity Autoencoder Designs

Federico Zocco and Seán McLoone, Senior Member, IEEE

Abstract—Reducing dimensionality is a key preprocessing step in many data analysis applications to address the negative effects of the curse of dimensionality and collinearity on model performance and computational complexity, to denoise the data or to reduce storage requirements. Moreover, in many applications it is desirable to reduce the input dimensions by choosing a subset of variables that best represents the entire set without any a priori information available. Unsupervised variable selection techniques provide a solution to this second problem. An autoencoder, if properly regularized, can solve both unsupervised dimensionality reduction and variable selection, but the training of large neural networks can be prohibitive in time sensitive applications. We present an approach called Recovery of Linear Components (RLC), that is designed to be a middle ground between linear and non-linear dimensionality reduction techniques, reducing autoencoder training times while enhancing performance over purely linear methods. With the aid of synthetic and real world case studies, we show that the RLC, when compared with an autoencoder of similar complexity, shows higher accuracy, similar robustness to overfitting, and faster training times. Additionally, at the cost of a relatively small increase in computational complexity, RLC is shown to outperform the current state-of-the-art for a semiconductor manufacturing wafer measurement site optimization application.

Index Terms—Dimensionality reduction, variable selection, feature selection, deep learning, semiconductor manufacturing, wafer profile reconstruction

I. INTRODUCTION

UNSUPERVISED variable selection and dimensionality reduction are fundamental aspects of data analysis, either as objectives in their own right, for example for sensor selection [1], factor analysis [2], and data visualisation applications [3], or as preprocessing steps to address the curse of dimensionality and collinearity issues that arise with high dimension data. Principal Component Analysis (PCA) is a well-established linear technique for unsupervised dimensionality reduction. This employs a linear transformation that maps the original variables in the data onto a new set of orthogonal components that are ordered such that they are successively orientated in the directions of maximum variance in the data [4]. A reduced dimension model is then obtained by retaining the first k components. Other linear transformation techniques include Fisher’s linear discriminant analysis (LDA) [5], canonical correlation analysis (CCA) [6], and multidimensional scaling (MDS) [7]. For a comprehensive review of linear dimensionality reduction the interested reader is referred to [8].

Rather than transforming to a new set of variables, it is often desirable to reduce dimensionality by selecting a subset of the original variables (or features) as this enhances the transparency and interpretability of models [9], and in some applications it is a necessity, for example in sensor selection and measurement plan optimisation problems [10], [11]. Since determining the best subset of variables from a candidate set, with respect to a given performance metric, is an NP-hard combinatorial optimisation problem, practical implementations of unsupervised variable selection usually involve approximate solutions such as those provided by greedy search algorithms [12], [13], [14], [15], [16].

Linear methods enjoy relatively low computational complexity, but are not able to fully exploit the redundancy in the data when the underlying relationships between variables are non-linear [17], [18], [19]. Therefore, nonlinear techniques have been developed to address both dimensionality reduction [20], [21] and variable selection [22], [23], [24], [25], [26], [27], [28], with neural network based algorithms [29], [30], [31], [32], [33], [34] proving particularly popular.

The first four of the aforementioned neural network based approaches employ an autoencoder architecture, as depicted in Fig. 1. This consists of two sub-networks in cascade: an encoding network, \( t = M_k^e(x) \), that maps a \( v \)-dimensional input vector \( x \) to a \( k < v \) dimensional intermediate vector \( t \); and a decoding network, \( \hat{x} = M_k^d(t) \), that attempts to reconstruct \( x \) from \( t \). \( M_k^e(x) \) and \( M_k^d(t) \) are in general nonlinear functions whose parameters are determined by training the overall network to produce the identity mapping, i.e. \( M_k^e(M_k^d(x)) \rightarrow x \). Dimensionality reduction is then achieved by virtue of the bottleneck layer of size \( k \).

These neural network based techniques have the advantage of being able to exploit recent advances in deep learning to improve their accuracy, but this comes at the expense of substantially greater computational complexity and training times [35], [36]. Consequently, in this paper we propose an unsupervised dimensionality reduction and feature selection methodology, referred to as Recovery of Linear Components (RLC), that is designed to be a middle ground between autoencoders and linear dimensionality reduction techniques. The methodology involves two stages; first, linear techniques are used to produce an ordered set of \( k_{lin} \) linearly transformed components from the original \( v \) dimensional dataset; then, the enhanced representational capabilities of neural networks are exploited to predict these \( k_{lin} \) components, from a subset of the first \( k \) components. The advantage of this two-stage approach is that it requires less complex networks compared to autoencoders, thereby accelerating the training phase.

The main contributions of this paper are the development
of the RLC methodology and training algorithm, and the benchmarking of its performance on a range of case studies. In particular:

- we show experimentally that the RLC, when compared with a conventional nonlinear autoencoder of similar complexity, shows higher accuracy, similar robustness to overfitting, and faster training times;
- for the specific case of a semiconductor manufacturing wafer surface measurement plan optimisation application which motivated this work, we show that RLC outperforms the state-of-the-art, allowing fewer measurements to be taken while still accurately recovering the full wafer profile [37], [12].

The remainder of the paper is organized as follows. Section II provides a brief description of the dimensionality reduction, variable selection and autoencoder algorithms that underpin the work. The RLC methodology and training algorithm are introduced in Section III. Experimental results are presented in sections IV and V for unsupervised variable selection and unsupervised dimensionality reduction, respectively. Finally, conclusions are provided in Section VI.

II. PRELIMINARIES

The following notation and conventions are adopted throughout the paper. Matrices and vectors are denoted by bold capital and lowercase letters, respectively, while scalars are denoted by non-bold lower case letters. The dimensionality reduction/variable selection problem is with respect to a dataset consisting of \( m \) measurements of \( v \) variables collected in a matrix \( \mathbf{X} \in \mathbb{R}^{m \times v} \), hereinafter assumed to be scaled to have zero-mean columns. The objective is to find a lower dimensional representation of \( \mathbf{X} \) that adequately captures the information contained in \( \mathbf{X} \). Denoting this representation as \( \mathbf{T}_k \in \mathbb{R}^{m \times k} \), the encoding process can be expressed as \( \mathbf{T}_k = \mathcal{M}^e_k(\mathbf{X}) \) and the decoding process (i.e. the reconstruction of \( \mathbf{X} \) from \( \mathbf{T}_k \)) as \( \hat{\mathbf{X}} = \mathcal{M}^d_k(\mathbf{T}_k) \).

The adequacy of \( \mathbf{T}_k \) as a representation of \( \mathbf{X} \) can be quantified in terms of the percentage variance explained \( V_{EX}() \) between the original matrix \( \mathbf{X} \) and its reconstruction \( \hat{\mathbf{X}} \), that is

\[
V_{EX}(\mathbf{X}; \hat{\mathbf{X}}) = 100 \left( 1 - \frac{||\mathbf{X} - \hat{\mathbf{X}}||^2_F}{||\mathbf{X}||^2_F} \right),
\]

where \( || \cdot ||_F \) denotes the Frobenius norm. Alternatively the lossiness of \( \mathbf{T}_k \) can be measured in terms of the mean squared reconstruction error \( M_{SE}(\cdot) \) defined as

\[
M_{SE}(\mathbf{X}; \hat{\mathbf{X}}) = \frac{1}{mv} ||\mathbf{X} - \hat{\mathbf{X}}||^2_F
\]

This is complementary to \( V_{EX}(\cdot) \) since

\[
M_{SE}(\mathbf{X}; \hat{\mathbf{X}}) = \alpha (100 - V_{EX}), \quad \text{with} \quad \alpha = \frac{||\mathbf{X}||^2_F}{100mv},
\]

hence, as an optimization objective, maximizing \( V_{EX}(\cdot) \) is equivalent to minimizing \( M_{SE}(\cdot) \).

The dimensionality reduction techniques that are the foundation of RLC can be classified as either linear or nonlinear. In the former both the encoding and decoding processes are linear transformations, while in the latter they are nonlinear.

A. Linear techniques

**Dimensionality reduction:** In linear dimensionality reduction the encoding function is given by

\[
\mathbf{t} = \mathcal{M}^e_k(\mathbf{x}) = \mathbf{X} \mathbf{P}_k,
\]

The corresponding decoding transformation is

\[
\hat{\mathbf{X}} = \mathcal{M}^d_k(\mathbf{T}_k) = \mathbf{T}_k \mathbf{B},
\]

where

\[
\mathbf{B} = (\mathbf{T}_k^\top \mathbf{T}_k)^{-1} \mathbf{T}_k^\top \mathbf{X}
\]

defines the least squares error linear reconstruction of \( \mathbf{X} \) from \( \mathbf{T}_k \). The encoding transformation matrix \( \mathbf{P}_k \) maximising the variance explained, that is

\[
\mathbf{P}_k = \arg\max_{\mathbf{P}_k} V_{EX}(\mathbf{X}; \hat{\mathbf{X}}),
\]

is then given by the first \( k \) principal components (PCs) of the principal component analysis (PCA) of \( \mathbf{X} \) [4]. These may be computed as the eigenvectors of the sample covariance matrix

\[
\mathbf{C} = \frac{1}{m-1} \mathbf{X}^\top \mathbf{X}
\]

corresponding to its \( k \) largest eigenvalues [38]. Under these conditions, the decoding transformation reduces to \( \mathbf{B} = \mathbf{P}_k^\top \).

**Variable selection:** When performing dimensionality reduction via variable selection \( \mathbf{T}_k \) is constrained to be a subset of the columns of \( \mathbf{X} \), or equivalently the columns of the encoding transformation matrix \( \mathbf{P}_k \) are constrained to be a subset of the standard basis vectors of \( \mathbf{X} \). A standard basis vector \( \mathbf{e}_i \) is a vector with its \( i \)th element equal to 1, and all other elements equal to zero, hence the identity matrix \( \mathbf{I}_v \) can be written as \( \mathbf{I}_v = [\mathbf{e}_1 \mathbf{e}_2 \ldots \mathbf{e}_v] \). Using this notation the variable selection optimisation problem can be compactly expressed as

\[
\mathbf{P}_k = \arg\max_{\mathbf{P}_k \subseteq \mathcal{I}_v} V_{EX}(\mathbf{X}; \hat{\mathbf{X}})
\]
While finding the optimum subset of $k$ variables from a candidate set of $v$ variables is an NP-hard combinatorial optimisation problem, and therefore computationally intractable in general, satisfactory approximate solutions can be found using greedy search methods such as Forward Selection Component Analysis (FSCA) [12]. FSCA performs greedy selection using $V_{EX}(\cdot)$ as a variable ranking criterion. More precisely, at each iteration it adds to the set of currently selected variables the one that maximizes the variance explained. Since (9) is a constrained version of the PCA optimisation problem (7), it follows that the number of PCs required to achieve a specific value of $V_{EX}(\cdot)$ is a lower bound of the number on variables needed to achieve the same reconstruction accuracy [12].

To reduce the gap between the FSCA solution and the optimum solution, should one exist, [12] propose several refinements to the basic FSCA algorithm. Among these, the two approaches that provided the best accuracy-complexity compromise were the Single-Pass Backward Refinement (SPBR) and the Multi-Pass Backward Refinement (MPBR) algorithms. Since these algorithms can be viewed as refinements of the encoding map $M_k^E(X)$ in order to improve its performance, they are complimentary to the methodology proposed herein as it can be interpreted as a refinement of FSCA that focuses on improving the decoding map $M_k^D(T_k)$.

B. Neural network based nonlinear techniques

Dimensionality reduction: The autoencoder, as depicted in Fig. 1, is a well established approach to achieving unsupervised nonlinear dimensionality reduction. It consists of a multilayer neural network model with a bottleneck layer whose structure and parameters are determined by solving a non-convex optimization problem [29]. A classical single hidden layer artificial neural network (ANN) generates a nonlinear mapping between the desired target $Y \in \mathbb{R}^{n_y \times n_y}$ and the input $X$ of the form

$$\hat{Y}^T = f_2(W_2 f_1(W_1X^T + B_1) + B_2),$$

(10)

where, $n_y$ and $n_h$ are the number of neurons in the output and hidden layer, respectively, $W_1 \in \mathbb{R}^{n_h \times n_y}$, $W_2 \in \mathbb{R}^{n_y \times n_h}$ are the weight matrices between the input and the hidden layer and hidden layer and the output, respectively, $B_1 \in \mathbb{R}^{n_h \times m}$, $B_2 \in \mathbb{R}^{n_y \times m}$ are the corresponding bias matrices, $f_1(\cdot)$ is the activation function of the $l$-th layer and $Y^T \in \mathbb{R}^{n_y \times m}$ is the network output (i.e. the estimate of the desired target $Y$ transposed). When (10) is configured as a single hidden layer autoencoder, $Y = X$ and $n_h = k$. If linear activation functions $f_l(\cdot)$ are used, the autoencoder is linear and yields equivalent $V_{EX}(\cdot)$ performance to PCA. However, the practical value of an autoencoder is its ability to produce more general nonlinear encoding-decoding transformations through the use of nonlinear activation functions such as the logistic sigmoid, the hyperbolic tangent, or the rectified linear unit (ReLU).

Denoting the output of the autoencoder as

$$N(X; \theta) \triangleq [f_2(W_2 f_1(W_1X^T + B_1) + B_2)]^T,$$

(11)

where for compactness of notation we introduce

$$\theta \triangleq \{W_1, B_1, W_2, B_2\},$$

(12)

the parameters of the optimum autoencoder $N^*(X; \theta^*)$ can be computed as

$$\theta^* = \arg \min_\theta J(X; \theta)$$

(13)

with $J(X; \theta)$ usually chosen as the mean square error cost function $M_{SE}(X; \hat{X}(\theta))$, that is

$$J(X; \theta) = \frac{1}{mv} ||X - N(X; \theta)||_F^2.$$  

(14)

This non-convex optimisation problem is typically solved using a gradient descent algorithm [39], a quasi-Newton algorithm [40], or the Levenberg-Marquardt algorithm [41].

A single hidden layer autoencoder has limited nonlinear compression capabilities since representational capabilities are a function of the number of hidden neurons, which is required to be small to achieve dimensionality reduction. To achieve a high level of input space compression, i.e. $k \ll v$, a more complex autoencoder structure has to be designed. Thus, further hidden layers are added to obtain a deep structure called a stacked autoencoder (SAE) [42]. Assuming the general case of an ANN having $L$ layers, the autoencoder equation (11) becomes

$$N(X; \theta) \triangleq [f_L(W_L \cdots f_1(W_1X^T + B_1) + B_2) + B_1] \cdots + B_1 + B_L]$$

(15)

However, increasing the number of layers makes training the autoencoder much more challenging due to the much greater computational complexity, the vanishing gradient problem and the potential to over fit the data when using deep neural networks [36]. Many different strategies have been proposed to address these challenges. Among the most effective are: early stopping where training is stopped when the error on a validation set starts to increase [43]; introducing different types of operation in the hidden layers such as convolution or pooling to achieve weight sharing and network size reduction [44], [30]; Dropout [45] and its extension DropConnect [46] where neurons are randomly dropped out with a fixed probability during the training phase as a form of regularisation; and greedy layer-wise pre-training [47], [48] of networks to improve regularization and training performance through good weight initialization.

Various approaches to achieving greedy layer-wise pre-training have been proposed including using a deep belief net (DBN) [49], a stacked denoising autoencoder (SDAE) [50], a k-sparse autoencoder [51], and a sparse autoencoder [52]. In our work, a sparse autoencoder is used, hence, during pre-training, the basic cost function (14) is augmented with regularisation and sparsity terms to become

$$J(X; \theta) = \frac{1}{mv} ||X - N(X; \theta)||_F^2 + \lambda \sum_{l=1}^{L-1} ||W_l||_F^2 + \gamma \sum_{i=1}^{k} KL(\rho | \hat{\rho}_i).$$

(16)
Here, $\lambda$ and $\gamma$ are the weightings applied to the weight decay (i.e., $L_2$ or ridge regression) and sparsity terms, respectively. The sparsity penalty $KL(\rho|\bar{\rho}_i)$, defined as

$$KL(\rho|\bar{\rho}_i) = \rho \log \frac{\rho}{\bar{\rho}_i} + (1 - \rho) \log \frac{1 - \rho}{1 - \bar{\rho}_i},$$

is the Kullback-Leibler (KL) divergence between two Bernoulli random variables with mean $\rho$ and $\bar{\rho}_i$, respectively, where the random variable mean $\bar{\rho}_i$ is computed as the mean of the output of the $i$th neuron in the bottleneck layer averaged over the training data, and $\rho$ is the target value for this parameter. This is chosen to be close to zero (e.g., 0.05) to encourage inactivation of the hidden layer neurons, and hence sparsity.

**Variable selection:** Practically, nonlinear variable selection is not computationally tractable, even using greedy search methods, due to the huge overhead associated with training the nonlinear models for each variable combination evaluated. Consequently, implicit approaches are preferred in the literature when using neural network models, whereby the training cost function $J(X; \theta)$ is modified to include a term that induces sparsity in the connection matrix between the inputs and the first hidden layer (i.e., $W_1$). For example, in [34, 53] the training phase is split into two phases with the first phase learning the input-output mapping and the second phase focusing on shrinking to zero some of the input layer weights by solving an optimization problem based on the nonnegative garrote parameter. The $L_1$ regularization term, also known as the least absolute shrinkage and selection operator (LASSO), is added to the cost function (14) in [33] to achieve input selection in a Multilayer Perceptron neural network, while [31] and [32] integrated a row-sparse regularization term and a graph regularization trace-ratio criterion, respectively, within the training of an autoencoder to indirectly obtain nonlinear variable selection. Even when using implicit approaches, the computational burden associated with nonlinear variable selection can be prohibitive in time-constrained applications such as dynamic spatial sampling for silicon wafer process monitoring [11], especially when training large neural models. This has motivated our investigation of a methodology that requires the training of less complex nonlinear models, and can be thought as a middle ground between linear and nonlinear variable selection techniques.

### III. Proposed Algorithm

The methodology we propose is called Recovery of Linear Components (RLC). The key idea of our approach is to retain a linear encoding block, but employ a nonlinear model for the decoding block to enhance representational capabilities in the reconstruction phase. Specifically, assuming that the desired reconstruction accuracy of $X$ can be achieved using $k_{lin}$ linear components, then $\hat{k}$ of these are discarded to further reduce the dimensionality to $k < k_{lin}$. Then a nonlinear (neural network) model is used to recover the discarded linear components from the subset that have been retained. In this way, the nonlinear model is only needed to capture the nonlinear relationships between the retained and discarded linear components leading to a much lower complexity model than required for a full $v$ to $v$ autoencoder. In particular, the sum of the input and output dimensions of the RLC neural network model is $k + \hat{k} = k_{lin} < v$, compared to $2v$ for a standard autoencoder. The methodology can be used for both unsupervised dimensionality reduction and unsupervised variable selection if combined with either PCA or FSCA, respectively, to implement the linear encoding step. Hereinafter, the RLC algorithm will be referred to as PCA-RLC or FSCA-RLC whenever it is used in combination with PCA or FSCA, respectively.

The implementation of RLC is described in Algorithm 1. The inputs to the algorithm are the data matrix $X \in \mathbb{R}^{m \times v}$, the threshold $\tau$ for the desired level of variance explained, the string $Task \in \{DR, VS\}$ indicating whether dimensionality reduction (DR) or variable selection (VS) is being requested, the number of linear components to be selected $k$, and the neural network hyperparameters (number of layers, number of units per layer, maximum number of training epochs, and regularization parameters).

The algorithm begins by calling the function $\text{Define}\_k_{lin}$ which is described in Algorithm 2. This returns $k_{lin}$ as the number of components needed to achieve the desired reconstruction accuracy $\tau$ with the selected linear technique (PCA or FSCA). The goal of RLC is then to achieve this level of accuracy using only $k$ components, where $k < k_{lin}$, allowing $\hat{k} = k_{lin} - k$ components to be discarded. The Nonlinear Iterative Partial Least Squares (NIPALS) algorithm

![Algorithm 1 Recovery of Linear Components (RLC)](image_url)
which computes principal components sequentially in order of significance, can be used to efficiently compute the first \( k_{lin} \) principal components if \( k_{lin} << v \), otherwise they can be computed using singular value decomposition (SVD) [55] or by the covariance method described in the previous section.

The outputs of the RLC algorithm are the linear encoding matrix \( P_{k_{lin}} \in \mathbb{R}^{v \times k_{lin}} \), the linear decoding matrix \( \beta \), and the trained neural network model \( N^* \) whose parameters are obtained by minimizing

\[
J_{RLC}(T_k; \theta) = \| T_k - N(T_k; \theta) \|_F^2.
\]  
(18)

The matrix \( T_k \in \mathbb{R}^{m \times k} \) is the reduced dimensionality dataset consisting of either the first \( k \) PCs or FSCA selected variables, and \( T_k^+ \in \mathbb{R}^{k \times m} \) is its pseudo-inverse. In the algorithm pseudocode, the text on the right between \( \langle \rangle \) highlights the distinction between the training and test phases. The steps defining the models belong to the training phase whereas steps using the models belong to the test phase.

For benchmarking purposes we implemented two additional algorithms to allow comparisons with more complete deep neural network autoencoder implementations. The first of these, presented in Algorithm 3, tackles the unsupervised dimensionality reduction problem using PCA and a stacked autoencoder (SAE) in cascade, and is called PCA-SAE. PCA reduces the dimension of the input space from \( v \) to \( k_{lin} \), then the encoder block of the SAE reduces the dimensionality to \( k \). The decoder block followed by linear regression are applied to recover \( X \). The training of the SAE is performed in two stages: first a sparse autoencoder based greedy layer-wise pre-training is performed, then the output layer is added and the full network is fine-tuned using the Levenberg-Marquardt training algorithm with early stopping. The objective function for the fine-tuning stage is

\[
J_{SAE}(T_{k_{lin}}; \theta) = \| T_{k_{lin}} - N(T_{k_{lin}}; \theta) \|_F^2.
\]  
(19)

The second algorithm, Algorithm 4, tackles the unsupervised variable selection problem. In common with PCA-SAE it employs a deep neural network model and requires a two stage training process, but differs from PCA-SAE by virtue of the fact that the encoding map \( \mathcal{M}_b^k(X) \) is fully linear as in the RLC algorithm, and only the decoder block is nonlinear. The objective function when training the stacked decoder (SDE) block is

\[
J_{SDE}(X; \theta) = \| X - N(T_k; \theta) \|_F^2.
\]  
(20)

The algorithm, referred to as FSCA-SDE, has been developed to benchmark the RLC algorithm against a technique that shares the same linear encoding map, but exploits the accuracy of a deep neural network in the decoding map \( \mathcal{M}_b^k(T_k) \).

The RLC, PCA-SAE and FSCA-SDE architectures can be interpreted as large neural network models consisting of two or three sub-neural models that are independently trained and connected in series. This is illustrated graphically in Fig. 2.

IV. RESULTS FOR UNSUPERVISED VARIABLE SELECTION

In this section, in addition to FSCA, FSCA-RLC and FSCA-SDE, the SPBR and MPBR algorithms presented in

Algorithm 2 Define \( k_{lin} \)

1: Input: \( X, \tau, Task \)
2: Output: \( k_{lin} \)
3: \( k = 0, \tilde{V}_{EX} = 0 \)
4: while \( \tilde{V}_{EX} < \tau \) do
5: \( k = k + 1 \)
6: switch \( Task \) do
7: case DR
8: \( \tilde{V}_{EX} = PCA(X, \tilde{k}) \)
9: case VS
10: \( \tilde{V}_{EX} = FSCA(X, \tilde{k}) \)
11: end switch
12: end while
13: \( k_{lin} = k \)

Algorithm 3 PCA-Stacked Autoencoder (PCA-SAE)

1: Input: \( X, \tau, k \), neural model hyperparameters
2: Output: \( P_{k_{lin}}, N^*, \beta \)
3: Task = ‘DR’
4: \( k_{lin} = Define_k_{lin}(X, \tau, Task) \)
5: \( P_{k_{lin}} = PCA(X, k_{lin}) \) \( \langle \text{defines } P_{k_{lin}} \rangle \)
6: \( T_{k_{lin}} = X P_{k_{lin}} \) \( \langle \text{uses } P_{k_{lin}} \rangle \)
7: if \( k_{lin} \leq k \) then
8: \( \hat{X} = T_{k_{lin}} P_{k_{lin}}^T \)
9: else
10: Greedy layer-wise SAE pre-training via sparse autoencoder (16) with \( \lambda = 0.001, \gamma = 1, \rho = 0.05 \)
11: Include the output layer and fine-tune the full SAE \( N(T_{k_{lin}}; \theta) \) to minimize (19) \( \langle \text{defines } N^* \rangle \)
12: \( \hat{X} = \hat{T}_{k_{lin}} \beta \) \( \langle \text{uses } \beta \rangle \)
13: \( \beta = \hat{T}_{k_{lin}}^+ X \) \( \langle \text{defines } \beta \rangle \)
14: end if

Algorithm 4 FSCA-Stacked Decoder (FSCA-SDE)

1: Input: \( X, \tau, k \), neural model hyperparameters
2: Output: \( P_{k_{lin}}, N^*, \beta \)
3: Task = ‘VS’
4: \( k_{lin} = Define_k_{lin}(X, \tau, Task) \)
5: \( P_{k_{lin}} = FSCA(X, k_{lin}) \) \( \langle \text{defines } P_{k_{lin}} \rangle \)
6: \( T_{k_{lin}} = X P_{k_{lin}} \) \( \langle \text{uses } P_{k_{lin}} \rangle \)
7: if \( k_{lin} \leq k \) then
8: \( \beta = T_{k_{lin}}^+ X \) \( \langle \text{defines } \beta \rangle \)
9: \( \hat{X} = T_{k_{lin}} \beta \) \( \langle \text{uses } \beta \rangle \)
10: else
11: Greedy layer-wise SDE pre-training via sparse autoencoder (16) with \( \lambda = 0.001, \gamma = 1, \rho = 0.05 \)
12: Include the output decoder and fine-tune the full SDE \( N(T_k; \theta) \) to minimize (20) \( \langle \text{defines } N^* \rangle \)
13: \( \hat{X} = N^*(T_k; \theta^*) \) \( \langle \text{uses } N^* \rangle \)
14: end if
Hence, there are three groups of variables \( \{x_1, \ldots, x_6\} \), \( \{x_7, x_8, x_9\} \) and \( \{x_{10}\} \), and at least one variable must be selected from each group to enable extraction of the generator variables. Matrix \( X^{NL} \in \mathbb{R}^{m \times 10} \) is formed from \( m \) measurements of these variables. Using a randomly generated linear map \( \Psi \in \mathbb{R}^{10 \times (v-10)} \) where \( \Psi_{ij} \sim \mathcal{N}(0, 1) \), and noise matrix \( N \in \mathbb{R}^{m \times (v-10)} \), where \( N_{ij} \sim \mathcal{N}(0, \sigma^2) \), \( v - 10 \) linearly related variables are generated as \( X^L = X^{NL} \Psi + N \).

The final dataset is then constituted as \( X = [X^{NL}, X^L] + 70 \cdot 1_{m1^T} \in \mathbb{R}^{m \times v} \), where \( 1_{dim} \) denotes a column vector of ones of length \( dim \).

The variable selection algorithms have been applied to this dataset with \( v = 50 \) and \( m = 500 \) for two different noise levels, \( \sigma^2 = 0 \) and \( \sigma^2 = 0.01 \), and with the experimental parameters as described in Table I. The average and standard deviation (over 100 Monte Carlo simulations) of the variance explained by each algorithm as a function of \( k \) is reported in Fig. 3. FSCA-SDE and FSCA-RLC are the best performing methods with almost equivalent variance explained profiles in terms of mean and standard deviation. SPBR and MPBR perform similarly to FSCA with the exception of \( k = 3 \) (\( \sigma^2 = 0 \)) and \( k = 4 \) (\( \sigma^2 = 0.01 \)) where they are marginally superior. All methods are relatively robust to variations in the dataset, as suggested by the low standard deviation for each \( k \) and their convergence to the same \( V_{EX} \) when \( k > 5 \). In the absence of noise the nonlinear methods are able to achieve \( V_{EX} > 98\% \) with 3 variables (\( k = 3 \)), whereas 5 variables are required to achieve the same level of accuracy with the linear methods. The frequency with which each variable is selected in the linear encoding step when \( k = 10 \) is shown in Fig. 4. This reveals that the first 10 variables are selected with increasing frequency as we move from FSCA to SPBR to MPBR, and are almost exclusively chosen when using MPBR. The focusing of selection on the first 10 variables in more pronounced with the noisy version of the dataset. This is a consequence of the reduced coupling between the first 10 nonlinear variables and the linearly related variables with the addition of noise. While the variables selected by each method vary considerably between methods and between simulation runs they yield similar levels of variance explained, as evident from Fig. 3.

The algorithm computational times, relative to FSCA, are reported in Table II for several values of \( k \) (\( \sigma^2 = 0.01 \)). FSCA is used as the baseline as it has the lowest computational complexity and is executed as the first step in each of the other methods; SPBR is the quickest method, followed by MPBR, and then FSCA-RLC. FSCA-RLC is 30 times more computationally intensive to compute than MPBR for \( k = 2 \) but becomes more efficient with increasing \( k \) and only requires twice the computation time of MPBR for \( k = 8 \). In contrast, FSCA-SDE is more than two orders of magnitude more computationally intensive than FSCA-RLC.

### A. Synthetic dataset

The synthetic dataset, denoted \( X^{\text{synthetic}} \), is defined as follows. Given generator variables \( \{g_i\}_{i=1,2,3} \sim \mathcal{N}(0, 1) \) and noise variables \( \{n_i\}_{i=1,2,\ldots,10} \sim \mathcal{N}(0, \sigma^2) \), ten nonlinearly related variables are defined as
\[
\begin{align*}
x_1 &= \sin(g_1) + n_1 \\
x_2 &= \cos(g_1) + n_2 \\
x_3 &= x_1^2 + n_3 \\
x_4 &= \text{sign}(g_1) \cdot \tanh(g_1) + n_4 \\
x_5 &= \sqrt{x_1^2 + x_2^2} + n_5 \\
x_6 &= g_1 \cdot x_1^2 + x_2^2 + n_6 \\
x_7 &= \cos^2(10g_2) + n_7 \\
x_8 &= \frac{|x_1 + x_2|}{\sqrt{x_1^2 + x_2^2}} + n_8 \\
x_9 &= \frac{g_2}{\sqrt{x_1 + x_2}} + n_9 \\
x_{10} &= g_1^2 \cdot \frac{g_2}{\sqrt{x_1 + x_2}} \cdot e^{\sin(g_1) \cdot g_3} \cdot \cos^2(g_2) \cdot \frac{\sin(g_3)}{44} + n_{10}
\end{align*}
\]

1https://github.com/fedezocco/RecOfLinComp-MATLAB

---

### B. Real world benchmark datasets

To assess the performance of the variable selection algorithms more generally they are further evaluated for six real world datasets, namely, data on weekly purchases of products
(Xbusiness [56]), chemical sensor array data (Xchemistry [57], [58]), the extended Yale face database B (YaleB [59], [60]), the USPS handwritten digit dataset (USPS [61], [62]), and two industrial case studies on semiconductor manufacturing (Xwafer [37] and Xsemicon [63]). Further details on these datasets and the experimental settings employed are provided in Table I. The computation times (relative to FSCA) for each algorithm are reported in Table II while their performance in terms of variance explained as a function of $k$ is presented in Fig. 5. Whenever results have been omitted for an algorithm it is because it has proven to be too computationally expensive to compute compared to the other algorithms.

Overall, the most effective variable selection method is FSCA-SDE followed closely by FSCA-RLC, but the former is at least 100 times more computationally complex to train than the latter, hence it was deemed intractable for the largest datasets. Consistent with the finding of [12], MPBR has an accuracy comparable to SPBR and a computation complexity that increases more rapidly with $k$. For the lowest values of $k$ SPBR is the fastest algorithm to compute, but it is less accurate than FSCA-RLC. As $k$ increases, the difference in variance explained between algorithms decreases and for $k \geq 40$ FSCA-RLC becomes faster to compute than SPBR, as observed with YaleB and USPS. This can be explained considering that SPBR has $O(2^{n_{th}}k^3 + 8nk^3)$ complexity and hence $O(k^3)$ complexity with respect to $k$, whereas the RLC neural network has a complexity of $O(mbe)$ where $e$ is the number of epochs required to satisfy early stopping and $\theta$ is the number of weights in the network [20]. In these experiments $\theta = kh + hk + k = hkk_{in} + h + k_{in} - k$, hence the size of the network decreases as $k$ increases, as does the computational complexity per iteration (hypothesizing that $e$ does not increase if the network gets smaller).

**Extended analysis of Xwafer:** The development of RLC was initially motivated by a semiconductor manufacturing spatial sampling design problem, as exemplified by the Xwafer dataset [37], [64]. Here, each column of the data matrix represents the measurements of a process parameter, such as etch depth or deposition layer thickness, at a specific wafer site (location on the wafer surface) for a batch of wafers. The objective is to identify the minimum number of measurement sites needed to accurately monitor the full wafer surface, or equivalently, to find the minimum subset of columns of the matrix that can achieve a desired level of accuracy in terms of reproducing the full matrix. An important requirement for practical application is to have a relatively low computation time for the algorithm performing site selection and surface reconstruction. Therefore, complex deep learning techniques are not suitable in this context.

The site selection problem has already been addressed in [37], [64] and the state-of-the-art is defined by the SPBR and MPBR proposed in [12], where for a target surface reconstruction accuracy of 99% or greater they yield a minimum measurement plan of 6 sites in contrast to 7 sites with FSCA. To assess the performance of FSCA-RLC and FSCA-SAE for this problem Table III focuses on the results for $k = 5$ and $k = 6$ since these are the minimum values of $k$ required to achieve the desired reconstruction accuracy with at least one of the algorithms. FSCA-RLC yields the best performance on average and exceeds the 99% threshold more frequently than the other methods, with FSCA-SDE a close second. SPBR is the least computationally complex of the 4 algorithms and, along with MPBR, exhibits the smallest standard deviation in performance over the 100 Monte Carlo simulations. The situation for $k = 5$ is further clarified with the boxplot in Fig.

### Table I

| Dataset | Size | Description |
|---------|------|-------------|
| Xsynthetic | $n \times m$ | Synthetic dataset |
| Xwafer | $n \times m$ | Semiconductor manufacturing dataset |
| YaleB | $n \times m$ | Face recognition dataset |
| USPS | $n \times m$ | Handwritten digit dataset |

### Table II

| Algorithm | Computation Time (x FSCA) |
|-----------|---------------------------|
| FSCA      | 100x                       |
| SPBR      | 10x                        |
| MPBR      | 1000x                      |
| FSCA-SDE  | 10000x                     |
| FSCA-RLC  | 50000x                     |

### Footnotes

1. Downloaded from: https://archive.ics.uci.edu/ml/datasets/Sales_Transactions_Dataset_Weekly2 (when you paste the link in the search engine, manually add the 3 underscores).
2. Downloaded from: http://archive.ics.uci.edu/ml/datasets/Seismic+Data+3. Downloaded from: http://archive.ics.uci.edu/ml/datasets/Sensor Array+Different+Concentrations (only the batch number 3 was considered).
3. Downloaded from: http://archive.ics.uci.edu/ml/datasets/Marketing+Campaign+for+Hypothetical+Electronic+Retailer

---

**Fig. 3.** Performance of the variable selection algorithms under investigation as a function of $k$ for Xsynthetic: (a) $\sigma^2 = 0$, and (b) $\sigma^2 = 0.01$. The dots and error bars indicate the mean and standard deviation, respectively, of the variance explained for 100 Monte Carlo simulations.

**Fig. 4.** Variable selection frequency (averaged over 100 Monte Carlo simulations) with FSCA, SPBR and MPBR when $k = 10$ for Xsynthetic for $\sigma^2 = 0$ and $\sigma^2 = 0.01$.2
TABLE I

| Dataset (m x v) | Source | m_{tr} (%) | No. MC sims | τ | Method | Size of $\mathcal{N}$ | Epochs limit |
|----------------|--------|------------|--------------|---|--------|-----------------------|--------------|
| Xsynthetic (500 x 50) | Sec. IV-A | 70 | 100 | 99 | FSCA-SDE | [k, 11, 21, v] | 1000 |
| Xwafer (316 x 50) | [37] | 70 | 100 | 99.9 | FSCA-SDE | [k, 15, 25, v] | 1000 |
| Xbusiness (811 x 52) | [56] | 70 | 100 | 97 | FSCA-SDE | [k, 11, 21, v] | 1000 |
| Xchemistry (1586 x 128) | [57], [58] | 40 | 30 | 99 | FSCA-SDE | [k, 13, 18, v] | 20 |
| Xsemicon (2194 x 2046) | [63] | 50 | 50 | 97 | FSCA-SDE | [k, 3, k] | 1000 |
| YaleB (2414 x 1024) | [59], [60] | 45 | 60 | 97 | FSCA-SDE | [k, 5, k] | 8 |
| USPS (9298 x 256) | [61], [62] | 15 | 70 | 97 | FSCA-SDE | [k, 5, k] | 20 |

Fig. 5. Variance explained as a function of k. Each line connects the mean values, while the vertical spread is the standard deviation. Table I provides details of the simulation setting employed. Missing methods in (c), (d) and (e) have been omitted as they required an excessive training time to be competitive (due to the large value of v and $k_{init}$).

TABLE II

| Method | Xsynthetic Mean(std) | Xchemistry Mean(std) | Xbusiness Mean(std) | Xsemicon Mean(std) | YaleB Mean(std) | USPS Mean(std) |
|--------|----------------------|----------------------|--------------------|--------------------|----------------|----------------|
| SPBR   | $2 \begin{pmatrix} 4.4(3.8) \\ 5.7(5.3) \\ 9.7(11.7) \end{pmatrix}$ | $2 \begin{pmatrix} 5.8(2.9) \\ 16.5(2.0) \\ 33.2(8.2) \end{pmatrix}$ | $2 \begin{pmatrix} 4.5(2.5) \\ 8.1(8.6) \\ 9.7(13.1) \end{pmatrix}$ | $2 \begin{pmatrix} 2.5(2.3) \\ 4.4(2.2) \\ 5.1(1.8) \end{pmatrix}$ | $2 \begin{pmatrix} 10.0(2.4) \\ 37.1(20.0) \\ 54.7(30.9) \end{pmatrix}$ | $2 \begin{pmatrix} 6.0(5.0) \\ 34.1(2.3) \end{pmatrix}$ |
| MPBR   | $2 \begin{pmatrix} 6.2(2.4) \\ 13.2(10.1) \\ 27.0(77.3) \end{pmatrix}$ | $2 \begin{pmatrix} 7.4(12.0) \\ 48.6(75.9) \\ 100.8(130.5) \end{pmatrix}$ | $2 \begin{pmatrix} 4.9(3.7) \\ 13.4(47.3) \\ 17.8(112.8) \end{pmatrix}$ | $2 \begin{pmatrix} 4.0(3.0) \\ 12.6(213.9) \\ 16.8(120.4) \end{pmatrix}$ | $2 \begin{pmatrix} - \\ - \end{pmatrix}$ | $2 \begin{pmatrix} 9.6(12.1) \end{pmatrix}$ |
| FSCA-SDE | $\frac{1}{10^4}$ | $2 \begin{pmatrix} 3.6(8.6) \\ 1.9(2.7) \\ 1.5(3.6) \end{pmatrix}$ | $2 \begin{pmatrix} 15.4(3.6) \\ 4.4(0.9) \\ 2.2(0.5) \end{pmatrix}$ | $2 \begin{pmatrix} 3.6(2.7) \\ 1.4(3.2) \\ 1.9(3.3) \end{pmatrix}$ | $2 \begin{pmatrix} - \end{pmatrix}$ | $2 \begin{pmatrix} - \end{pmatrix}$ |
| FSCA-RLC | $2 \begin{pmatrix} 216.2(34.3) \\ 109.9(332.3) \\ 49.6(46.7) \end{pmatrix}$ | $2 \begin{pmatrix} 211.5(857.3) \\ 40.8(576.2) \\ 9.0(12.3) \end{pmatrix}$ | $2 \begin{pmatrix} 193.6(29.0) \\ 81.1(64.6) \\ 59.7(135.6) \end{pmatrix}$ | $2 \begin{pmatrix} 5.0(7.2) \\ 2.1(6.4) \\ 1.4(2.4) \end{pmatrix}$ | $2 \begin{pmatrix} 64.9(36.0) \\ 2.9(5.1) \end{pmatrix}$ | $2 \begin{pmatrix} 224.0(48.4) \end{pmatrix}$ |
In this section three unsupervised dimensionality reduction methods are compared: PCA, PCA-SAE and PCA-RLC. The experimental setup and software tools employed are as introduced in Section IV. The MATLAB code developed for the experiments is available on GitHub. Four datasets are used as case studies, namely, Xsynthetic, Xwafer, Xchemistry and USPS. For each one three different PCA-SAE and PCA-RLC training scenarios are investigated:

**Scenario A**: In addition to employing the settings detailed in Table IV, the algorithms are implemented in accordance with Algorithms 1 and 3 with different randomly generated training and test datasets used for each Monte Carlo simulation;

**Scenario B**: Here the same training and test datasets are used in all simulations. Hence, any variation in \(V_{EX}\) will be a consequence of the neural network training process only. In addition, greedy layer-wise pre-training is not performed with PCA-SAE.

**Scenario C**: This is an extension of Scenario B where a network size equivalence condition (23) is imposed on PCA-SAE and PCA-RLC, as described later.

Fig. 7 shows the variance explained as a function of \(k\) for the four different datasets and the three training scenarios. Overall Fig. 7(a) shows that PCA-SAE is the most accurate method followed by PCA-RLC and PCA. While in Xsynthetic PCA-RLC has almost equivalent performance to PCA-SAE, in the other case studies PCA-SAE significantly outperforms PCA-RLC at lower \(k\) values. With the USPS dataset PCA-SAE struggles at the higher values of \(k\) and is the worst performing of the three methods for \(k > 10\). The variance in performance over the Monte Carlo simulations is relatively small for each algorithm and value of \(k\), which confirms the achievement of two positive outcomes: first, the training set contains enough information to permit us to define a model able to generalize on the test set, and second, the algorithms are robust to overfitting. Note also that the differences in variance explained between PCA-SAE and PCA-RLC is greater than the corresponding differences between FSCA-SDE and FSCA-RLC. This is simply a reflection of the fact that PCA-SAE benefits from having the additional representational power of a nonlinear encoding map whereas the other methods (including FSCA-SDE) have an encoding map that is entirely linear.

The most striking feature of the results for Scenario B (Fig. 7(b)) is the impact on the performance of PCA-SAE of not employing greedy pre-training. This autoencoder is now the worst performing method, particularly for lower values of \(k\), with both a lower mean and much higher dispersion in \(V_{EX}\) for each value of \(k\), highlighting the importance of performing pre-training to regularize deep neural networks. Since the training and test datasets remain the same for all simulations in this scenario, the observed dispersion is caused exclusively by the sub-optimal neural network training process. In contrast to PCA-SAE, the dispersion in \(V_{EX}\) decreases for PCA-RLC relative to Scenario A and is zero for PCA. PCA-SAE shows contradictory behaviour for higher values of \(k\) in the USPS case study as it achieves superior results to the other methods, whereas the opposite was the case for Scenario A.

Overall PCA-RLC yields stable training performance at each \(k\) for all case studies and does so without the use of any pre-training. This can largely be attributed to the lower complexity neural network models that arise with the RLC topology compared to PCA-SAE, as detailed in Table IV. The PCA-RLC neural network \(\mathcal{N}\) has fewer neurons and only a single-hidden layer, leading to a less challenging training problem than with the deeper PCA-SAE architecture. Therefore, as a final evaluation, Scenario C compares the performance of the two approaches when they are constrained to have the same number of parameters and layers, as expressed in the following equivalence condition (23):

\[
\theta_{SAE} = k(k + k_l + k_l + k_l) = 2k - k_l + k_l + k_l
\]

\[
\theta_{RLC} = k + h + h + k = h(k + k_l + k_l) - k
\]

\[
\theta_{SAE} = \theta_{RLC} \Rightarrow h = 2k
\]

Equation (21) defines the number of parameters of a single-hidden layer \(\mathcal{N}\) PCA-SAE architecture, while equation (22) is the equivalent expression for a PCA-RLC with \(h\) hidden neurons. In both topologies the neural network hidden layers...
are implemented with hyperbolic tangent activation functions, while the output layers are linear.

The results for Scenario C are presented in Fig. 7(c). Since PCA-SAE is not employing a deep network, as was the case in Scenario B, it has much more stable training performance even without pre-training, but its representational capabilities are greatly diminished with the result that it can only achieve the same level of explained variance as linear PCA. Under these conditions PCA-RLC is substantially superior to PCA-SAE while also having much shorter training times (see Table V).

To gain some insight into why training times are much shorter for PCA-RLC than for PCA-SAE in Scenario C, even though both networks have the same number of weights, the number of epochs required to train each network in each simulation and the mean computational time over the first few epochs are plotted in Figures 8 and 9, respectively, for the four case studies. Two observations can be made with regard to these results: first, recalling that early stopping is employed as part of the training process, the total number of epochs required to train a PCA-RLC averages between 15 and 50 and seldom exceeds 100, whereas early stopping rarely interrupts the training of PCA-SAE which often reaches the 1000 limit (i.e. the number of components k is relatively small). It is also robust to overfitting and substantially faster to train than SAEs that can achieve similar or better performance. In particular, for the specific case of the semiconductor manufacturing wafer measurement plan optimisation problem that motivated this work, FSCA-RLC is able to achieve state-of-the-art performance with a modest increase in computational complexity, and is two orders of magnitude faster to train than conventional SAEs.

The FSCA and PCA implementations of RLC (PCA-RLC and FSCA-RLC), representative of variable selection and dimensionality reduction formulations, respectively, have been presented and evaluated on a synthetic dataset and several real world case studies. The results confirm that RLC is able to achieve significantly higher levels of variance explained than linear techniques when high levels of compression are required (i.e. the number of components k is relatively small).

VI. Conclusions

Recovery of Linear Components (RLC) has been proposed as a novel methodology for unsupervised dimensionality reduction and variable selection, whereby starting with a set of klin ordered linear components, a neural network is used to recover the full set of components from a subset of the first k components. Crucially, by employing a linear encoding block and a nonlinear decoding block, RLC provides a compromise between fully nonlinear autoencoders and linear techniques such as PCA and FSCA, delivering improved representational capabilities over linear methods, while requiring significantly lower complexity autoencoder designs than conventional SAEs.

ACKNOWLEDGMENTS

The first author gratefully acknowledges Irish Manufacturing Research (IMR) for the financial support provided for this work.

REFERENCES

[1] J. Ranieri, A. Chebira, and M. Vetterli, “Near-optimal sensor placement for linear inverse problems,” IEEE Trans. Signal Process., vol. 62, no. 5, pp. 1135–1146, 2014.
[2] H. H. Harman, Modern factor analysis. University of Chicago press, 1976.
[3] M. Becker, J. Lippel, A. Stuhlsatz, and T. Ziebel, “Robust dimensionality reduction for data visualization with deep neural networks,” Graphical Models, vol. 108, article 101060, 2020.
[4] I. T. Jolliffe, “Principal Component Analysis. Second Edition,” Springer Series in Statistics, 2002. [Online]. Available: http://link.springer.com/10.1007/b98835
[5] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” Annals of Eugenics, vol. 7, no. 2, pp. 179–188, 1936.
[6] H. Hotelling, “Relations between two sets of variates,” Biometrika, vol. 28, no. 3/4, pp. 321–377, 1936.
Fig. 7. Variance explained by each dimensionality reduction technique as a function of $k$ for four different datasets with: (a) Scenario A; (b) Scenario B, and; (c) Scenario C. The lines connect the mean values at each $k$ and the vertical bars denote the standard deviations.

Fig. 8. Number of epochs required in each Monte Carlo simulation to train $\mathcal{N}$ in PCA-SAE and PCA-RLC for Scenario C. Early stopping is employed for regularization and the maximum number of epochs allowed is as specified in Table IV.

Fig. 9. The mean computation times (averaged over the Monte Carlo simulations) for the first few epochs of the training process for PCA-SAE and PCA-RLC when operating in accordance with Scenario C.

Table V

| Dataset | PCA-SAE $\times 10^3$ | PCA-RLC |
|---------|----------------------|---------|
| $k$ | $\mathcal{N}$ | $\mathcal{N}$ |
| $k$ | $\mathcal{N}$ | $\mathcal{N}$ |
| $k$ | $\mathcal{N}$ | $\mathcal{N}$ |
| $k$ | $\mathcal{N}$ | $\mathcal{N}$ |

The fastest algorithm for each dataset and each $k$ is highlighted in bold.
[56] S. C. Tan and J. P. San Lau, “Time series clustering: A superior alternative for market basket analysis,” in Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013). Springer, Singapore, 2014, pp. 241–248.

[57] A. Vergara, S. Vembu, T. Ayhan, M. A. Ryan, M. L. Homer, and R. Huerta, “Chemical gas sensor drift compensation using classifier ensembles,” Sensors and Actuators B: Chemical, vol. 166, pp. 320–329, 2012.

[58] I. Rodriguez-Lujan, J. Fonollosa, A. Vergara, M. Homer, and R. Huerta, “On the calibration of sensor arrays for pattern recognition using the minimal number of experiments,” Chemometrics and Intelligent Laboratory Systems, vol. 130, pp. 123–134, 2014.

[59] A. Georgiades, P. Belhumeur, and D. Kriegman, “From few to many: Illumination cone models for face recognition under variable lighting and pose,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 23, no. 6, pp. 643–660, 2001.

[60] D. Cai, X. He, Y. Hu, J. Han, and T. Huang, “Learning a spatially smooth subspace for face recognition,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition Machine Learning (CVPR’07), 2007.

[61] D. Cai, X. He, J. Han, and T. S. Huang, “Graph regularized nonnegative matrix factorization for data representation,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 33, no. 8, pp. 1548–1560, 2010.

[62] D. Cai, X. He, and J. Han, “Speed up kernel discriminant analysis,” The VLDB Journal, vol. 20, no. 1, pp. 21–33, 2011.

[63] L. Puggini and S. McLoone, “Extreme learning machines for virtual metrology and etch rate prediction,” in 2015 26th Irish Signals and Systems Conference (ISSC). IEEE, 2015, pp. 1–6.

[64] S. McLoone, F. Zocco, M. Maggipinto, and G. A. Susto, “On optimising spatial sampling plans for wafer profile reconstruction,” IFAC-PapersOnLine, vol. 51, no. 10, pp. 115–120, 2018.