Deep PLS: A Lightweight Deep Learning Model for Interpretable and Efficient Data Analytics

Xiangyin Kong, and Zhiqiang Ge, *Senior Member, IEEE*

***Abstract*— The salient progress of deep learning is accompanied by nonnegligible deficiencies, such as: 1) interpretability problem; 2) requirement for large data amounts; 3) hard to design and tune parameters; and 4) heavy computation complexity. Despite the remarkable achievements of neural networks-based deep models in many fields, the practical applications of deep learning are still limited by these shortcomings. This article proposes a new concept called the lightweight deep model (LDM). LDM absorbs the useful ideas of deep learning and overcomes their shortcomings to a certain extent. We explore the idea of LDM from the perspective of partial least squares (PLS) by constructing a deep PLS (DPLS) model. The feasibility and merits of DPLS are proved theoretically, after that, DPLS is further generalized to a more common form (GDPLS) by adding a nonlinear mapping layer between two cascaded PLS layers in the model structure. The superiority of DPLS and GDPLS is demonstrated through four practical cases involving two regression problems and two classification tasks, in which our model not only achieves competitive performance compared with existing neural networks-based deep models but also is proven to be a more interpretable and efficient method, and we know exactly how it improves performance, how it gives correct results. Note that our proposed model can only be regarded as an alternative to fully connected neural networks at present and cannot completely replace the mature deep vision or language models.**

***Index Terms*— Deep learning, latent variable models, machine learning, model interpretability, partial least squares (PLSs).**

I. Introduction

DEEP learning has been one of the hottest spots in the machine learning and artificial intelligence communities for the past decade [1]–[3]. It has made remarkable achievements in many fields, such as deep convolutional neural networks (CNNs) in image classification [4], [5], long short-term memory (LSTM) networks [6] in machine translation [7], generative adversarial network (GAN) in data generation [8], and deep reinforcement learning (DRL) in games [9]. However, albeit the significant achievements of deep learning in many fields, it also brings some new problems, specifically as follows.

1) Interpretability problem: Deep neural network (DNN) is just like a "black box" and has no gratifying interpretability. We do not really know the whole operation process inside of DNN and how it gives correct results, which usually means there are inevitable concerns about reliability in the practical application of deep learning.
2) Requirement for large data amounts: Deep learning needs a lot of training data samples, when the training samples are small (this problem is indeed existing in many practical fields), this modeling method usually leads to intolerably poor performance.
3) Hard to design and tune parameters: DNN usually consists of many model hyperparameters, including many network structure hyperparameters and optimizer hyperparameters. These parameters are very tricky to assign and adjust.
4) Heavy computation complexity: Tremendous weights and biases of DNN are updated iteratively during the model training phase through the backpropagation procedure, which leads to a very complicated training process, resulting in heavy computation complexity and long training time.

In fact, the first issue—interpretability problem is probably the most discussed question in recent years. Interpretability is concerned with model transparency, and lack of interpretability is one of the reasons why deep learning has been criticized, and research on model interpretability is currently one of the hottest frontiers in the deep learning community [10]. The interpretability can be categorized into ante-hoc and posthoc interpretabilities: ante-hoc interpretability refers to train self-explainable models, while posthoc interpretability means developing special techniques to interpret trained models. Problems 2)–4) are all related to the modeling efficiency. These problems are rooted in the mechanism of deep learning and seem to be contradictory to the performance of deep models. Compared with the traditional machine learning methods, one of the reasons for such great success of deep learning is the high model complexity. However, like a double-edged sword, the above-mentioned four problems are also related to this issue. High complexity extremely improves the ability of representation and feature extraction of deep models, but it also brings a lot of defects, such as bad interpretability, high computational burden, and so on.

Current deep models are almost all based on artificial neural networks. Are there any other better ways to build a

Manuscript received March 20, 2021; revised July 28, 2021, November 8, 2021, and February 9, 2022; accepted February 21, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 92167106, Grant 62103362, Grant 62003300, and Grant 61833014 and in part by the Natural Science Foundation of Zhejiang Province under Grant LR18F030001. *(Corresponding author: Zhiqiang Ge.)*

Xiangyin Kong is with the State Key Laboratory of Industrial Control Technology, College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China (e-mail: xiangyinkong@zju.edu.cn).

Zhiqiang Ge is with the State Key Laboratory of Industrial Control Technology, College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China, and also with the Peng Cheng Laboratory, Shenzhen 518000, China (e-mail: gezhiqiang@zju.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3154090.

Digital Object Identifier 10.1109/TNNLS.2022.3154090

Fig. 1. Connotation of LDM. In the upper part of the picture, current deep learning method tried to lift the box from "low performance" to "high performance." However, in this process, "deep learning" encountered various resistances, such as lack of data and so on. In order to deal with these resistances, "deep learning" needs some outside help, for example, advanced hardware. After solving these problems through external help, "deep learning" successfully lifted the box up, but it had turned the original white box into a black box, and simultaneously could not explain how the box got up. On the other hand, the lower part of the picture tells us that LDM not only has no problems, such as lack of data but also can clearly show researcher how it improves performance.

deep model? As mentioned, model complexity seems to be crucial to performance improvement, yet be contradictory to model interpretability and efficiency. Inspired by the successes and defects of deep learning, in the tradeoff between model interpretability and model performance, we provide our own insights in this article. It should be noted that model complexity is necessary for model performance, and the unique hierarchical and cascaded model structure of deep learning is a good way to increase complexity. If the unique ideas of deep learning can be applied to the traditional machine learning models—which hold succinct and clear principles, simple and fast to solve, convenient and limpid to train—can this combination not only improves the performance of traditional models but also simultaneously maintains the advantages of these models themselves (such as good interpretability, convenient training procedures, etc.)?

Based on this kind of pondering, in this article, we propose a new concept called the lightweight deep model (LDM). LDM holds the basic idea—deep, which means it utilizes hierarchical and cascaded structures to increase model complexity to improve the ability of representation and feature extraction. But the fundamental modeling tools are not artificial neurons; instead, LDM makes use of simpler approaches, such as the concise latent variable machine learning models [34]. In other words, LDM is a bit like deepening the traditional simple models through some ideas of deep learning. Since the traditional machine learning methods are usually transparent with good interpretability and efficient modeling procedures, by effectively deepening them to deep forms, we believe the LDM can enhance the performance of the corresponding traditional model (even achieve competitive performance compared with current mainstream deep models), while keeping the original merits of the traditional model that current deep models do not have. There are many traditional machine learning models, and the successful development of deep learning also provides us with many unique useful ideas, hence there are many ways to construct an LDM. However, no matter what the specific model structure is, the connotation of LDM is always achieving competitive performance compared with current deep learning models in appropriate scenarios, while overcoming the shortcomings of current deep models to the maximum. The connotation is vividly shown in Fig. 1.

As mentioned earlier, there may be many approaches to build a successful LDM. Specifically, under the connotation of LDM, in this article, we explored this concept from the perspective of the partial least squares (PLSs) [11] model. PLS has also been explained as "projection to latent structures," which is a class of techniques for modeling the association between observed variables and predicted variables by means of latent variables. PLS has particular predominance when dealing with the problems that the sample capacity is small, variable dimension is high, or badly multiple correlations exist, while in these situations, the ordinary least squares method usually presents poor modeling results. PLS is mainly used for regression and can also be used for classification, dimension reduction, feature extraction, and so on. Generally speaking, PLS is a mature and powerful latent variable modeling method, which makes prominent contributions in many fields [12], [13].

In this article, we combine the virtues of deep learning and PLS, first propose the deep PLS (DPLS) model. DPLS is a supervised model, and the basic form of DPLS is to use PLS to construct the hierarchical and cascaded model structure, then the obtained abstract representation in the highest layer is used to make a prediction. We not only prove the feasibility and merits of the DPLS model theoretically but also generalize it

into a more common form by adding a nonlinear mapping layer between two cascaded PLS layers. After establishing the framework of the DPLS model, four case studies with different properties are conducted to evaluate the performance of the proposed method. After that, model interpretability of DPLS is presented through the case studies, from which we can find that the DPLS model is a transparent deep model, and we know exactly how it improves performance, how it gives correct results. The results and analyses in this article demonstrated that DPLS is a more interpretable and efficient modeling method compared to DNN. However, here we want to emphasize that, our purpose in proposing LDM or DPLS is not to completely surpass the existing neural networks-based deep learning and replace it (actually the existing neural networks-based deep models are still the best choice in many fields such as computer vision and natural language processing), but to achieve comparable or even better performance in some appropriate scenarios while at the same time providing unique advantages that existing deep learning does not have.

In general, the motivation of this article can be summarized as *exploring nonneural-network manners to build deep models.* Another popular research that shares a similar motivation is gcForest [14]. gcForest is a decision tree ensemble method that uses stacked random forests to generate a deep forest ensemble. The performance of gcForest is highly competitive to DNN in many tasks, but the most criticized issues about gcForest may be its interpretability and model complexity. The complexity of stacked random forests is very high, whose training may be even more time-consuming than DNN. And [14] also did not theoretically interpret how gcForest improves the performance compared with single-layer random forests. To some extent, the proposed DPLS can be viewed as simultaneously performing deep matrix decomposition on observed variables matrix and target variables matrix. This is related to another line of deep models, i.e., deep matrix factorization (DMF) [15], [16], which is popular in multi-view learning, image processing, and multiscale clustering. DMF utilizes a hierarchical structure to learn information of different attributes. Although stacked matrix decomposition is used in both DPLS and DMF, the difference between them is also quite obvious. DFM is essentially a feature extractor, and the extracted features are fed into a preset classifier or regressor to make final predictions. While DPLS is essentially a learner which can directly make classification or regression. Besides, DMF uses backpropagation and gradient descent to iteratively train the model, which is computation-expensive and time-cost, while the training procedures of DPLS do not involve backpropagation and gradient descent, thus resulting in a quite fast training process. In addition, DMF sometimes has nonnegative constraints on the factor matrices, which may be unsuitable for some real scenarios.

The rest of this article is organized as follows. Section II starts with a short overview of PLS, then the framework of the proposed DPLS model and its generalization form are presented. Four different case studies are conducted in Section III to evaluate the superiority of the proposed methods, including two regression problems and one classification problem. Finally, conclusions are drawn in Section IV.

II. Methods

A. Overview of PLS Model

PLS seeks two different projection vectors from observed variables and predicted variables, respectively, so that the covariance between the projected data reaches the maximum. The covariance not only contains various information of the two groups of variables but also measures the degree of correlation between the two projected vectors, PLS assumes the vectors with maximum covariance can best describe the association between observed variables and predicted variables.

Let $\boldsymbol{X} \in \mathbb{R}^{n\times m}$ denotes the dataset of observed variables, while $\boldsymbol{Y} \in \mathbb{R}^{n\times d}$ is the predicted variables dataset, where n is the number of samples, m is the number of observed variables and d denotes the number of predicted variables. Here we assume $\boldsymbol{X}$ and $\boldsymbol{Y}$ have both already been centralized. The PLS model can be formulated as an optimization task [11]

$$\begin{aligned} &\max \langle \boldsymbol{X}\boldsymbol{p}_1, \boldsymbol{Y}\boldsymbol{q}_1 \rangle \\ &\text{s.t. } \|\boldsymbol{p}_1\| = 1, \quad \|\boldsymbol{q}_1\| = 1 \end{aligned} \tag{1}$$

where $\langle\cdot\rangle$ represents the inner product, and $\boldsymbol{p}_1 \in \mathbb{R}^m$, $\boldsymbol{q}_1 \in \mathbb{R}^d$. Denote $\boldsymbol{u}_1 = \boldsymbol{X}\boldsymbol{p}_1$ and $\boldsymbol{v}_1 = \boldsymbol{Y}\boldsymbol{q}_1$, here $\boldsymbol{u}_1$ and $\boldsymbol{v}_1$ are the first pair of latent vectors extracted by PLS, and their covariance $\text{cov}(\boldsymbol{u}_1, \boldsymbol{v}_1) = \langle \boldsymbol{X}\boldsymbol{p}_1, \boldsymbol{Y}\boldsymbol{q}_1 \rangle$ is the optimization objective.[1]

Equation (1) is an optimization task with equality constraints. By introducing two Lagrange multipliers $\lambda/2$ and $\mu/2$, we can obtain the following Lagrangian function:

$$\mathcal{L} = \boldsymbol{p}_1^T \boldsymbol{X}^T \boldsymbol{Y}\boldsymbol{q}_1 - \frac{\lambda}{2}\left(\boldsymbol{p}_1^T \boldsymbol{p}_1 - 1\right) - \frac{\mu}{2}\left(\boldsymbol{q}_1^T \boldsymbol{q}_1 - 1\right). \tag{2}$$

Calculating the partial derivatives $((\partial\mathcal{L})/(\partial\boldsymbol{p}_1))$ and $((\partial\mathcal{L})/(\partial\boldsymbol{q}_1))$, and let them equal to 0, we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{p}_1} = \boldsymbol{X}^T \boldsymbol{Y}\boldsymbol{q}_1 - \lambda \boldsymbol{p}_1 = 0 \tag{3}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_1} = \boldsymbol{Y}^T \boldsymbol{X}\boldsymbol{p}_1 - \mu \boldsymbol{q}_1 = 0. \tag{4}$$

From (3) and (4), we can deduce that $\lambda = \mu$. Substituting (4) into (3) and (3) into (4), respectively, we can obtain the following eigenvalue decomposition problems:

$$\boldsymbol{X}^T \boldsymbol{Y}\boldsymbol{Y}^T \boldsymbol{X}\boldsymbol{p}_1 = \lambda^2 \boldsymbol{p}_1 \tag{5}$$

$$\boldsymbol{Y}^T \boldsymbol{X}\boldsymbol{X}^T \boldsymbol{Y}\boldsymbol{q}_1 = \lambda^2 \boldsymbol{q}_1 \tag{6}$$

where λ^2 is the largest eigenvalue of matrices $\boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{X}$ and $\boldsymbol{Y}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{Y}$. $\boldsymbol{p}_1$ is the corresponding eigenvector of matrix $\boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{X}$ and eigenvalue λ^2, and $\boldsymbol{q}_1$ is the corresponding eigenvector of matrix $\boldsymbol{Y}^T\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{Y}$ and eigenvalue λ^2. So far, (1) has been solved.

As can be seen, problem (1) can be solved through matrix decomposition [i.e., (5) and (6)]. But unlike other matrix decomposition-based models, such as principal component

[1]The covariance can be positive or negative. The sign of the covariance is not important here, to avoid ambiguity, the covariance $\text{cov}(\cdot)$ in this article represents the magnitude.

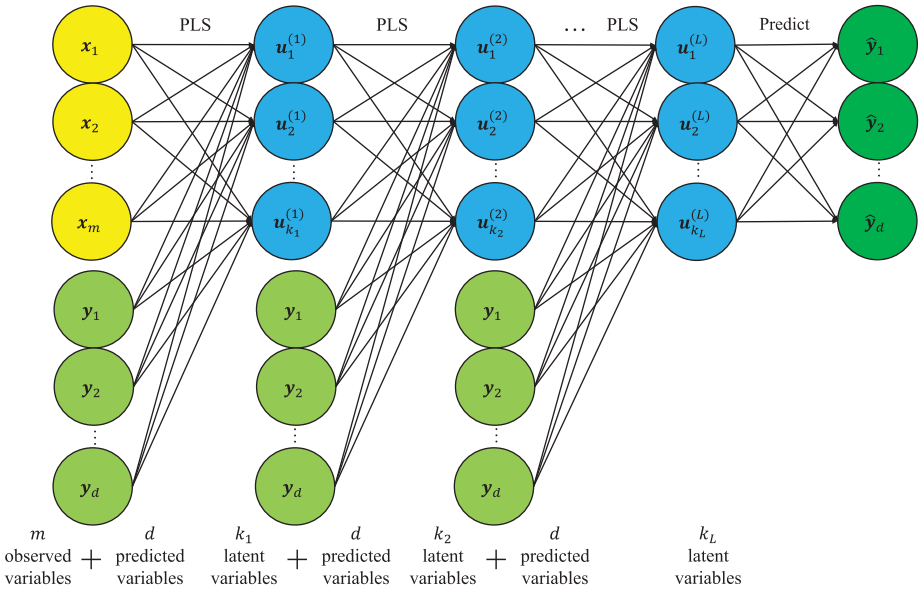


Fig. 2. Model architecture of DPLS. In the training phase, DPLS will perform matrix decomposition on the whole training set, and the letters in the circles all represent vectors. In the testing phase, the test data will be passed forward through the entire model architecture. If the test data contains multiple samples, the letters in the circles also represent vectors. If the test data is a single sample, the letters in the circles are scalars.

analysis (PCA) or nonnegative matrix factorization (NMF), whose algorithm is decomposing only on $\boldsymbol{X}$, the matrix factorization of PLS is performed through simultaneously decomposing $\boldsymbol{X}$ and $\boldsymbol{Y}$ (specifically, $\boldsymbol{X}^T\boldsymbol{Y}$). Such a factorization manner is named cross decomposition.

It is easy to find that $\boldsymbol{u}_1$ has a direct relationship with $\boldsymbol{X}$, at the same time, PLS assumes $\boldsymbol{u}_1$ is a good predictor of $\boldsymbol{Y}$, which actually hypothesizes that $\boldsymbol{u}_1$ has a linear relationship with $\boldsymbol{v}_1$ [11]. After obtaining $\boldsymbol{u}_1$ and $\boldsymbol{v}_1$, we perform rank-one deflation on $\boldsymbol{X}$ [11]

$$\boldsymbol{X} = \boldsymbol{X} - \frac{\boldsymbol{u}_1\boldsymbol{u}_1^T}{\|\boldsymbol{u}_1\|^2}\boldsymbol{X}. \tag{7}$$

The new $\boldsymbol{X}$ in (7) is again substituted into optimization problem (1), while the new solutions of (1) bring us the second pair of latent vectors $\boldsymbol{u}_2$ and $\boldsymbol{v}_2$. Repeat the above-mentioned process k times (k can be determined through cross validation [11]), we get two groups of latent vectors $\boldsymbol{u}_i (i = 1, 2, \ldots, k)$ and $\boldsymbol{v}_i (i = 1, 2, \ldots, k)$ since the PLS model hypothesizes $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ have linear relationship, and here, we only need to record $\boldsymbol{u}_i$. It is worth noting that k should be no greater than the rank of $\boldsymbol{X}$. In general, k vectors $\boldsymbol{u}_i (i = 1, 2, \ldots, k)$ are the latent variables extracted by PLS model, and the association between $\boldsymbol{X}$ and $\boldsymbol{Y}$ can be modeled through $\boldsymbol{u}_i$. Optimization task (1) can be directly solved through matrix decomposition. On the other hand, if one wants to avoid decomposition computations, PLS can also be solved by iteration algorithm. The most popular algorithm for PLS is the NIPALS [17], which computes the latent variables one by one through iteration.

B. Deep PLS Model

One of the most important characteristics of deep learning is the hierarchical and cascaded model structure. Currently, this kind of structure is usually realized through artificial neural networks, each layer of the model structure is connected through lots of neurons. In fact, neurons in each layer can be regarded as the features extracted by this layer of the network, and from this point, we can also use other feature extraction methods to implement the hierarchical and cascaded model structure. As described in Section II-A, the latent variables $\boldsymbol{u}_i$ in PLS are the features extracted through performing cross decomposition on $\boldsymbol{X}$ and $\boldsymbol{Y}$. The extracted features $\boldsymbol{u}_i$ can be forward passed to the next PLS model. That is, the next PLS model performs cross decomposition on $\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k]$ and $\boldsymbol{Y}$. In this way, we can successfully build a deep model structure through PLS. The latent variables outputted by this layer are used as the input of the next layer, so that different layers are hierarchically connected by PLS. This novel hierarchical and cascaded deep model structure is named DPLS, and the architecture of DPLS with L layers is shown in Fig. 2.

The connecting lines between each layer in Fig. 2 are a little different from those in DNN models. Connecting lines in DNN usually represent the linear connection weights between two neurons, while in Fig. 2, these lines only mean a kind of association (e.g., the value of $\boldsymbol{u}_1$ depends on $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m]$ and $[\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_d]$). During the training stage, observed variables (or latent variables) and predicted variables are crosses modeled through PLS. The main steps of cross modeling are finding the projection matrix which maps $\boldsymbol{X}$ to $\boldsymbol{U}$ and the prediction matrix which predicts $\boldsymbol{Y}$ through $\boldsymbol{U}$. In the DPLS model, the final estimated values of $\boldsymbol{Y}$ are predicted only by the latent variables of the highest layer, i.e., the prediction layer at the right end of Fig. 2. However, the latent variables $\boldsymbol{U}^{(l)}$ ($l = 1, 2, \ldots, L$ represents the number of layers) outputted from each PLS layer can also be used to make prediction. In the case studies of Section III, the results predicted by each hidden layer of DPLS are compared with show how deep structure improves the modeling performance.

Except for the experimental results, the merits of DPLS can also be directly evaluated through the theoretical level. As described in Section II-A, the principle of PLS is to maximize the covariance between $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$. Here, we will theoretically prove that the deep structure of DPLS is helpful in increasing the covariance, and the corresponding theorem is described in *Theorem 1*. Before the proof of *Theorem 1* is presented, we first give an auxiliary *Lemma 1*.

Lemma 1: For a given $n \times n$ Hermitian matrix $\boldsymbol{A}$, all its diagonal entries a_{ii} are between the matrix's smallest eigenvalue $\lambda_{\min}(\boldsymbol{A})$ and the largest eigenvalue $\lambda_{\max}(\boldsymbol{A})$

$$\lambda_{\min}(\boldsymbol{A}) \le a_{ii} \le \lambda_{\max}(\boldsymbol{A}) \quad (i = 1, \ldots, n). \tag{8}$$

Proof: Lemma 1 can be easily demonstrated through the Rayleigh quotient. For an arbitrary nonzero vector $\boldsymbol{x} \in \mathbb{C}^n$, the Rayleigh quotient of Hermitian matrix $\boldsymbol{A}$ is defined as

$$R(\boldsymbol{A}, \boldsymbol{x}) = \frac{\boldsymbol{x}^H \boldsymbol{A} \boldsymbol{x}}{\boldsymbol{x}^H \boldsymbol{x}}. \tag{9}$$

According to the properties of Rayleigh quotient, we have

$$\lambda_{\min}(\boldsymbol{A}) \le R(\boldsymbol{A}, \boldsymbol{x}) \le \lambda_{\max}(\boldsymbol{A}) \tag{10}$$

denote $\boldsymbol{e}_i \in \mathbb{C}^n$ as a unit vector whose ith element is 1, then

$$R(\boldsymbol{A}, \boldsymbol{e}_i) = \boldsymbol{e}_i^H \boldsymbol{A} \boldsymbol{e}_i = a_{ii} \quad (i = 1, \ldots, n). \tag{11}$$

By substituting (11) into (10), *Lemma 1* is, thus, proved. Moreover, the inequality (8) is tight iff a_{ii} is equaled to the eigenvalue $\lambda_{\max}(\boldsymbol{A})$ (or $\lambda_{\min}(\boldsymbol{A})$), and $\boldsymbol{e}_i$ is the corresponding eigenvector of $\boldsymbol{A}$ and $\lambda_{\max}(\boldsymbol{A})$ (or $\lambda_{\min}(\boldsymbol{A})$).

The merits of the proposed deep structure in DPLS are discussed in the succeeding theorem.

Theorem 1: The covariance between the first pair of latent variables gradually increases with the increase of layers in the DPLS model, that is,

$$\operatorname{cov}\left(\boldsymbol{u}_1^{(1)}, \boldsymbol{v}_1^{(1)}\right) \le \operatorname{cov}\left(\boldsymbol{u}_1^{(2)}, \boldsymbol{v}_1^{(2)}\right) \le \cdots \le \operatorname{cov}\left(\boldsymbol{u}_1^{(L)}, \boldsymbol{v}_1^{(L)}\right) \tag{12}$$

where the superscript represents the layer number.

Proof: According to the principle of DPLS model, we only need to prove the first inequality for the above-mentioned continuous inequalities, i.e., to prove that $\operatorname{cov}(\boldsymbol{u}_1^{(1)}, \boldsymbol{v}_1^{(1)}) \le \operatorname{cov}(\boldsymbol{u}_1^{(2)}, \boldsymbol{v}_1^{(2)})$.

According to (5) and (6), for DPLS, we have

$$\boldsymbol{X}^T \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{X} \boldsymbol{p}_1^{(1)} = \left(\lambda^{(1)}\right)^2 \boldsymbol{p}_1^{(1)} \tag{13}$$

$$\boldsymbol{Y}^T \boldsymbol{X} \boldsymbol{X}^T \boldsymbol{Y} \boldsymbol{q}_1^{(1)} = \left(\lambda^{(1)}\right)^2 \boldsymbol{q}_1^{(1)} \tag{14}$$

where $(\lambda^{(1)})^2$ is the largest eigenvalue of $\boldsymbol{X}^T \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{X}$, and $\lambda^{(1)}/n$ is the desired covariance between $\boldsymbol{u}_1^{(1)}$ and $\boldsymbol{v}_1^{(1)}$, which are computed by

$$\boldsymbol{u}_1^{(1)} = \boldsymbol{X} \boldsymbol{p}_1^{(1)}, \quad \boldsymbol{v}_1^{(1)} = \boldsymbol{Y} \boldsymbol{q}_1^{(1)}. \tag{15}$$

Denote $\boldsymbol{u}_1^{(2)}$ and $\boldsymbol{v}_1^{(2)}$ as the first pair of latent variables in the second layer and $\operatorname{cov}(\boldsymbol{u}_1^{(2)}, \boldsymbol{v}_1^{(2)}) = \lambda^{(2)}/n$. The magnitude of the covariance is measured by its absolute value, so to prove $\operatorname{cov}(\boldsymbol{u}_1^{(1)}, \boldsymbol{v}_1^{(1)}) \le \operatorname{cov}(\boldsymbol{u}_1^{(2)}, \boldsymbol{v}_1^{(2)})$, we just need to prove $|\lambda^{(1)}| \le |\lambda^{(2)}| \Longleftrightarrow (\lambda^{(1)})^2 \le (\lambda^{(2)})^2$.

Left multiplying (13) by $\boldsymbol{p}_1^{(1)^T}$, we have that

$$\boldsymbol{p}_1^{(1)^T} \boldsymbol{X}^T \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{X} \boldsymbol{p}_1^{(1)} = \left(\lambda^{(1)}\right)^2 \boldsymbol{p}_1^{(1)^T} \boldsymbol{p}_1^{(1)}. \tag{16}$$

According to (1), $\boldsymbol{p}_1^{(1)^T} \boldsymbol{p}_1^{(1)}$ should be equal to 1. Hence, (16) means that

$$\boldsymbol{u}_1^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_1^{(1)} = \left(\lambda^{(1)}\right)^2. \tag{17}$$

After extracting the first latent vector $\boldsymbol{u}_1^{(1)}$, it needs to be subtracted from the original data matrix, i.e., performing (7) to obtain a new $\boldsymbol{X}$. Substituting the new $\boldsymbol{X}$ into (13) and (14), we can gain the second pair of latent variables $\boldsymbol{u}_2^{(1)}$ and $\boldsymbol{v}_2^{(1)}$. After solving the decomposition problem (13) and (14) k times, we can form a latent vectors matrix $\boldsymbol{U}^{(1)} = [\boldsymbol{u}_1^{(1)}, \boldsymbol{u}_2^{(1)}, \ldots, \boldsymbol{u}_k^{(1)}] \in \mathbb{R}^{n \times k}$. At this point, the first layer of DPLS is completed. Treating $\boldsymbol{U}^{(1)}$ as the observed variables matrix of the next layer and performing PLS algorithm on $\boldsymbol{U}^{(1)}$ and $\boldsymbol{Y}$, we can, thus, construct the second layer.

Through the above-mentioned steps, we not only detailedly present how DPLS is constructed but also refine the proof of *Theorem 1* to a more concise form. Similar to (13), in the second layer of DPLS, we have that

$$\boldsymbol{U}^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{U}^{(1)} \boldsymbol{p}_1^{(2)} = \left(\lambda^{(2)}\right)^2 \boldsymbol{p}_1^{(2)} \tag{18}$$

where $(\lambda^{(2)})^2$ is the largest eigenvalue of $\boldsymbol{U}^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{U}^{(1)}$. Since $\boldsymbol{U}^{(1)} = \left[\boldsymbol{u}_1^{(1)}, \boldsymbol{u}_2^{(1)}, \ldots, \boldsymbol{u}_k^{(1)}\right]$, we have

$$\begin{aligned} &\boldsymbol{U}^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{U}^{(1)} \\ &\quad = \begin{bmatrix} \boldsymbol{u}_1^{(1)^T} \\ \boldsymbol{u}_2^{(1)^T} \\ \vdots \\ \boldsymbol{u}_k^{(1)^T} \end{bmatrix} \boldsymbol{Y} \boldsymbol{Y}^T \left[\boldsymbol{u}_1^{(1)} \boldsymbol{u}_2^{(1)} \cdots \boldsymbol{u}_k^{(1)}\right] \\ &\quad = \begin{bmatrix} \boldsymbol{u}_1^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_1^{(1)}, \ldots, \boldsymbol{u}_1^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_k^{(1)} \\ \vdots \\ \boldsymbol{u}_k^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_1^{(1)}, \ldots, \boldsymbol{u}_k^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_k^{(1)} \end{bmatrix}. \end{aligned} \tag{19}$$

It is easy to see that $\boldsymbol{U}^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{U}^{(1)}$ is a Hermitian matrix. Note that $(\lambda^{(1)})^2 = \boldsymbol{u}_1^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_1^{(1)}$ is a diagonal entry of $\boldsymbol{U}^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{U}^{(1)}$, and $(\lambda^{(2)})^2$ is the largest eigenvalue of the same matrix. According to *Lemma 1*, we have $(\lambda^{(1)})^2 \le (\lambda^{(2)})^2$. This completes the proof. ■

Recall *Lemma 1*, we can derive that the inequality $\operatorname{cov}(\boldsymbol{u}_1^{(1)}, \boldsymbol{v}_1^{(1)}) \le \operatorname{cov}(\boldsymbol{u}_1^{(2)}, \boldsymbol{v}_1^{(2)})$ is tight iff $(\lambda^{(1)})^2 = \boldsymbol{u}_1^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_1^{(1)}$ is the largest eigenvalue of $\boldsymbol{U}^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{U}^{(1)}$, and $\boldsymbol{e}_1 = [1, 0, \ldots, 0]^T$ is the corresponding eigenvector, which reads as

$$\begin{bmatrix} \boldsymbol{u}_1^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_1^{(1)}, \ldots, \boldsymbol{u}_1^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_k^{(1)} \\ \vdots \\ \boldsymbol{u}_k^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_1^{(1)}, \ldots, \boldsymbol{u}_k^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_k^{(1)} \end{bmatrix} \begin{bmatrix} 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} \left(\lambda^{(1)}\right)^2 \\ \vdots \\ 0 \end{bmatrix}. \tag{20}$$

Equation (20) implies that

$$\forall j \in \{2, 3, \ldots, k\}, \boldsymbol{u}_j^{(1)^T} \boldsymbol{Y} \boldsymbol{Y}^T \boldsymbol{u}_1^{(1)} = 0. \tag{21}$$

Note that (21) is a strong constraint for all the latent variables in the first layer. Although this constraint cannot be mathematically proved to be true or false, through tremendous experiments, we find that (21) is not true in most cases. That is, in most actual applications, the inequality $\text{cov}(\boldsymbol{u}_1^{(1)}, \boldsymbol{v}_1^{(1)}) \leq \text{cov}(\boldsymbol{u}_1^{(2)}, \boldsymbol{v}_1^{(2)})$ cannot achieve equality, which means *Theorem 1* can be generally relaxed into

$$\text{cov}\left(\boldsymbol{u}_1^{(1)}, \boldsymbol{v}_1^{(1)}\right) < \text{cov}\left(\boldsymbol{u}_1^{(2)}, \boldsymbol{v}_1^{(2)}\right) < \cdots < \text{cov}\left(\boldsymbol{u}_1^{(L)}, \boldsymbol{v}_1^{(L)}\right). \quad (22)$$

It is worth noting that, after each pair of latent variables has been extracted, this component must be subtracted from the data matrix [i.e., (7)]. This step has a great influence on the data matrix, resulting in the covariance of the second pair and the subsequent pair of latent variables does not follow a specific change rule. The first latent variable is of great importance since it is first extracted from the original data, it usually carries the most data information. Besides, the covariance between the first pair of latent variables is usually an order of magnitude larger than that between the later latent variables, and its proportion of total covariances usually reaches 70% or more. Therefore, the above theorem—which shows the relationship between the first pair of latent variables at different layers—contributes great significance to the model structure of DPLS.

Due to the clear principle of PLS, the model itself has antehoc interpretability. In addition, because of the transparency of the PLS model, we can easily construct some methods to understand the internal mechanism of the PLS model [18], thus providing posthoc interpretability. The DPLS model inherits this advantage of the PLS model by analyzing the DPLS model layer by layer through these methods, and we can clearly see how DPLS improves the performance. In the following, we will provide two methods with regard to data visualization and variable importance assessment, respectively.

Data visualization is a beneficial step in data modeling and analyzing. The visualization of high-dimensional data usually involves dimension reduction. Extracting the information which we are interested in through reducing data to the low dimension, we can visualize data in two or three dimensions. For the current DNN models, there usually exists no direct visualization approaches, we need to rely on specific complex external visualization approaches to visualize data [19]. However, due to the model mechanism of DPLS, dimension reduction and data visualization can be easily achieved. The latent vector $\boldsymbol{u}_i$ is like the principal component in PCA, thus, after projecting data to latent vector space, the values of the first three latent vectors $\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3$ can be used to visualize the data. Moreover, in the DPLS model, for each layer l, there is a visualization result presented by $\boldsymbol{u}_1^{(l)}, \boldsymbol{u}_2^{(l)}, \boldsymbol{u}_3^{(l)} (l = 1, 2, \ldots, L)$, these visualization results can be compared to show the feature extraction ability of different layers of DPLS. The advantage of convenient visualization of DPLS is especially meaningful in classification tasks. Because for a well-performing PLS-based classification model, in the first 2-D or 3-D latent vector space, data points of different categories may already have clear separating hyperplanes. The above-mentioned analysis will be presented in Section III-E through a practical case.

Research on model interpretability of DNN is currently one of the hottest frontiers. The lack of interpretability of DNN has been reflected in many aspects, one of which is that researchers are not able to know which features (including input variables and features of abstract layers) have important influences on decision-making. In recent years, many interpretation methods have been proposed to solve the above-mentioned problem of DNN, including feature inversion [20], [21], class activation mapping (CAM) [22], gradient-weighted CAM (Grad-CAM) [23], and so on. However, the shortcomings of these methods are also very obvious. Such as [21] can only be used for instance-level interpretation; while [20] is focused on model-level interpretation, but the inversion result is relatively rough and difficult to understand. All these methods are complex and computation-costly, and sometimes the interpretation methods for DNN may even be more complicated than the original DNN model.

When dealing with the above question that which features (including input variables and features of abstract layers) have important influences on decision-making, the proposed DPLS model has incomparable advantages over the DNN model, which essentially comes from the clear principle and transparency of the PLS model itself. In the PLS model, there is an indicator called variable importance in projection (VIP) [11]. VIP measures the contribution of each variable with respect to the output results, the VIP of the ath $(a = 1, 2, \ldots, m)$ variable is defined as follows:

$$\text{VIP}_a = \sqrt{m \frac{\sum_{i=1}^{k} \text{Rd}(\boldsymbol{y}; \boldsymbol{u}_i) p_{ai}^2}{\text{Rd}(\boldsymbol{y}; \boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k)}} \quad (23)$$

where p_{ai} is the element in the ath row and ith column of the matrix $\boldsymbol{P} = [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k]$, and the vector $\boldsymbol{p}_i (i = 1, 2, \ldots, k)$ comes from (1), which can be computed one by one through (5) and (7). p_{ai} is used to measure the contribution of $\boldsymbol{x}_a$ to the construction of latent variable $\boldsymbol{u}_i$. $\text{Rd}(\boldsymbol{y}; \boldsymbol{u}_i)$ represents the interpretable power of $\boldsymbol{u}_i$ to $\boldsymbol{y}$, while $\text{Rd}(\boldsymbol{y}; \boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k)$ represents the cumulative interpretable power of $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k$ to $\boldsymbol{y}$, and they are computed by

$$\begin{aligned} \text{Rd}(\boldsymbol{y}; \boldsymbol{u}_i) &= r^2(\boldsymbol{y}, \boldsymbol{u}_i) \\ \text{Rd}(\boldsymbol{y}; \boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k) &= \sum_{i=1}^{k} \text{Rd}(\boldsymbol{y}; \boldsymbol{u}_i) \end{aligned} \quad (24)$$

where $r(\boldsymbol{y}, \boldsymbol{u}_i)$ is the Pearson correlation coefficient between the predicted variable $\boldsymbol{y}$ and the latent variable $\boldsymbol{u}_i$, which is calculated by

$$r(\boldsymbol{y}, \boldsymbol{u}_i) = \frac{\text{cov}(\boldsymbol{y}, \boldsymbol{u}_i)}{\sigma_{\boldsymbol{y}} \sigma_{\boldsymbol{u}_i}} \quad (25)$$

where $\text{cov}(\cdot)$ is the covariance. $\sigma_{\boldsymbol{y}}$ is the standard deviation of $\boldsymbol{y}$, and $\sigma_{\boldsymbol{u}_i}$ is the standard deviation of $\boldsymbol{u}_i$.

The VIP index can be used in any layer of DPLS. According to the VIP values, we can not only understand the influence of each input variable but also evaluate the importance of each feature which lies in the middle abstract layer. The above-mentioned variable and feature importance analyses will be presented in Section III-E through a practical regression case.

C. Generalization of Deep PLS Model

One may notice that, though we utilize the PLS algorithm layer by layer to extract deep features, all of these processes are linear transformations. Assuredness, the DPLS model proposed in Section II-B is based on linear transformation, when dealing with linear application scenarios, the above DPLS model may achieve good modeling results, but when the problem presents strong nonlinearity, the performance of DPLS may not be satisfactory. Therefore, the adaptability of the DPLS model must be improved, in this Section, we proposed a generalization form of the above DPLS model, so that it can address the nonlinear factors in specific problems.

The first step is to modify the PLS algorithm so that it can adapt to nonlinear scenarios. The existing nonlinear PLS algorithms are mainly kernel methods [24], which hold the virtues of stability and accuracy. However, when the number of samples is relatively large, the computation of kernel PLS will increase to an intolerable level, besides, since there is no explicit nonlinear mapping in the kernel methods, the construction of layer by layer is the hierarchical and cascaded model structure may be inconvenient. Based on these considerations, we propose a new nonlinear PLS algorithm that directly applies specific nonlinear mapping to the data matrix, so that the nonlinear factors can be introduced to PLS; meanwhile, the hierarchical structure can also be easily established.

Denote ϕ as the nonlinear mapping function which maps $\boldsymbol{X}$ to a high-dimensional nonlinear space $\boldsymbol{H} = \phi(\boldsymbol{X})$. After obtaining $\boldsymbol{H}$, the PLS decomposition [(13) and (14)] can be performed on $\boldsymbol{H}$ and $\boldsymbol{Y}$. Here, we develop two manners to solve the decomposition problem. The first one is based on the NIPALS algorithm [17] whose essence is leveraging the power method [25] to calculate eigenvectors through iterations. On the other hand, (13) and (14) imply $\boldsymbol{p}_1^{(1)}$ is the left singular vector of $\boldsymbol{X}^T\boldsymbol{Y}$, and $\boldsymbol{q}_1^{(1)}$ is the right singular vector. Hence, the second manner is directly obtaining $\boldsymbol{p}_1^{(1)}$ and $\boldsymbol{q}_1^{(1)}$ through singular value decomposition (SVD). To distinguish, we define a parameter $solver$ which equals to "iter" or "svd." When $solver$ is "iter," the iterative method is used to compute $\boldsymbol{p}_1^{(1)}$ and $\boldsymbol{q}_1^{(1)}$. Otherwise, the SVD algorithm is directly used to solve the PLS decomposition. Whether to use "iter" or "svd" depends on the size of the dataset, which will be elaborated in Section II-D. Denote k as the number of extracted latent variables, I as the maximum number of iterations when $solver$ is "iter," and ϵ as the iteration convergence threshold. The proposed nonlinear PLS is listed in **Algorithm 1**.

Algorithm 1 uses the first "**if,…, else,…,** " statement to realize two different solving procedures corresponding to $solver$ equals to "iter" or "svd." The second "**if,…,** " statement is leveraged to control the inner loop. If an updated $\boldsymbol{p}$ is close enough to the old vector $\boldsymbol{p}_0$, the convergence is considered to be achieved, then the inner loop will be broken; otherwise, the maximum number of iterations I will be arrived. Note that the iteration will be executed only once when there is only one predicted variable ($d == 1$), which means the inner loop is no longer needed if Y is a vector [11]. The $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{u}, \boldsymbol{v}$ in **Algorithm 1** have the same meaning with the namesakes in (13), (14), and (15). $\boldsymbol{r}_j$ is the loading vector of $\boldsymbol{H}$ with respect to latent variable $\boldsymbol{u}_j$, while $\boldsymbol{c}_j$ is the loading vector of $\boldsymbol{Y}$ with respect to $\boldsymbol{u}_j$. $\boldsymbol{P}^*$ [11] is the projection matrix which projects the input data $\boldsymbol{H}$ to the latent space $\mathbb{R}^{n\times k}$, and $\boldsymbol{C}$ corresponds to the coefficient matrix of predicted variables $\boldsymbol{Y}$ with respect to the projected latent features. After collecting these matrices from **Algorithm 1**, for a new testing sample $\boldsymbol{x}_t$, its targets can be predicted by

$$\hat{\boldsymbol{y}}_t = \phi(\boldsymbol{x}_t)^T \boldsymbol{P}^* \boldsymbol{C}^T \tag{26}$$

where $\phi(\boldsymbol{x}_t) \in \mathbb{R}^h$.

The nonlinear mapping ϕ serves as a connection bridge between $\boldsymbol{X}$ and $\boldsymbol{Y}$. Instead of directly performing the original PLS algorithm on $\boldsymbol{X}$ and $\boldsymbol{Y}$, **Algorithm 1** first maps $\boldsymbol{X}$ to a nonlinear space $\boldsymbol{H}$; therefore, the final extracted latent variables $\boldsymbol{U}$ are not linear but nonlinear features. Just like the model structure of DPLS in Section II-B, latent variables $\boldsymbol{U}$ are continuously forward passed to the next nonlinear PLS layer.

Algorithm 1: Nonlinear PLS Algorithm

Input: $\boldsymbol{X} \in \mathbb{R}^{n\times m}$; $\boldsymbol{Y} \in \mathbb{R}^{n\times d}$; ϕ; k; $solver$; I; ϵ
Output: $\boldsymbol{U}$: latent variables matrix; $\boldsymbol{P}^*$: projection matrix; $\boldsymbol{C}$: coefficient matrix

Centering $\boldsymbol{X}$ and $\boldsymbol{Y}$
$\boldsymbol{H} = \phi(\boldsymbol{X}) \in \mathbb{R}^{n\times s}$, then centering $\boldsymbol{H}$
for $j = 1$ *to* k **do**
  if $solver$=='iter' **then**
    $\boldsymbol{v} \leftarrow$ one of the $\boldsymbol{Y}$ columns
    $\boldsymbol{p}_0 \leftarrow 10^6 * [1, 1, \ldots, 1]^T$
    for $i = 1$ *to* I **do**
      $\boldsymbol{p} = \boldsymbol{H}^T\boldsymbol{v}/\boldsymbol{v}^T\boldsymbol{v}$, normalize $\boldsymbol{p}$ to $\|\boldsymbol{p}\| = 1$
      $\boldsymbol{u} = \boldsymbol{H}\boldsymbol{p}$
      $\boldsymbol{q} = \boldsymbol{Y}^T\boldsymbol{u}/\boldsymbol{u}^T\boldsymbol{u}$
      $\boldsymbol{v} = \boldsymbol{Y}\boldsymbol{q}/\boldsymbol{q}^T\boldsymbol{q}$
      if $\|\boldsymbol{p} - \boldsymbol{p}_0\|^2 < \epsilon \vee d == 1$ **then**
        break
      end
      $\boldsymbol{p}_0 = \boldsymbol{p}$
    end
  else
    $\boldsymbol{M}, \boldsymbol{S}, \boldsymbol{N}^T = SVD(\boldsymbol{H}^T\boldsymbol{Y})$
    $\boldsymbol{p} =$ the first column of $\boldsymbol{M}$
    $\boldsymbol{q} =$ the first row of $\boldsymbol{N}^T$
  end
  $\boldsymbol{p}_j = \boldsymbol{p}$
  $\boldsymbol{u}_j = \boldsymbol{H}\boldsymbol{p}_j$
  $\boldsymbol{r}_j = \boldsymbol{H}^T\boldsymbol{u}_j/\boldsymbol{u}_j^T\boldsymbol{u}_j$
  $\boldsymbol{c}_j = \boldsymbol{Y}^T\boldsymbol{u}_j/\boldsymbol{u}_j^T\boldsymbol{u}_j$
  $\boldsymbol{H} = \boldsymbol{H} - \boldsymbol{u}_j\boldsymbol{r}_j^T$
end
$\boldsymbol{P} = [\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k] \in \mathbb{R}^{s\times k}$
$\boldsymbol{R} = [\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_k] \in \mathbb{R}^{s\times k}$
$\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k] \in \mathbb{R}^{n\times k}$
$\boldsymbol{P}^* = \boldsymbol{P}(\boldsymbol{R}^T\boldsymbol{P})^{-1} \in \mathbb{R}^{s\times k}$
$\boldsymbol{C} = [\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots, \boldsymbol{c}_k] \in \mathbb{R}^{d\times k}$

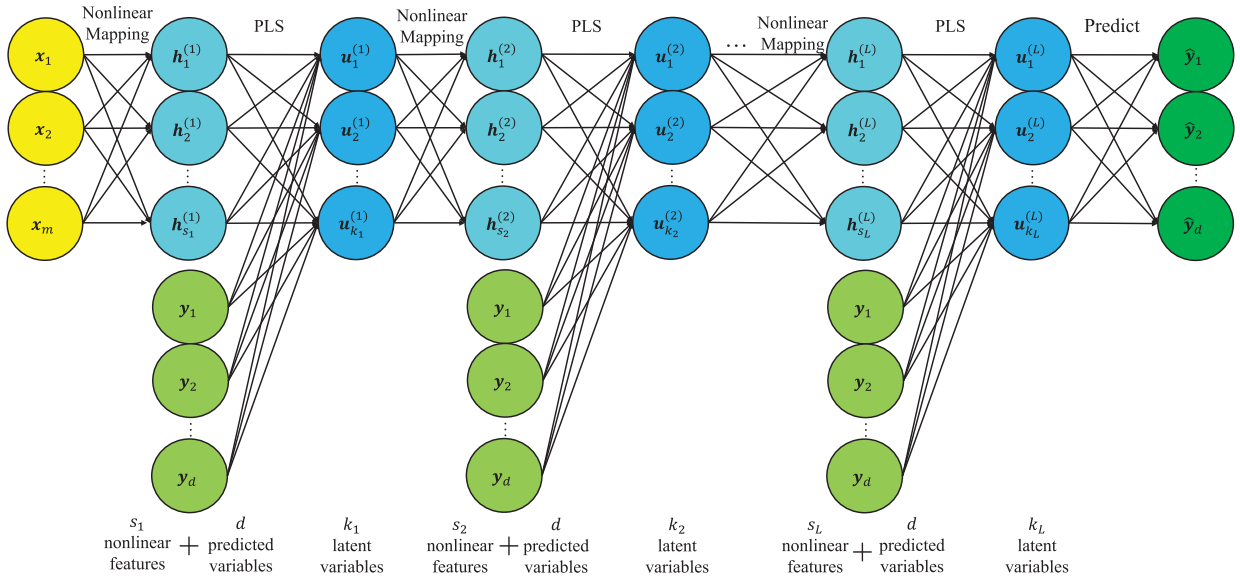


Fig. 3. Model architecture of GDPLS. The meaning of the letters in the circles is similar to that in Fig. 2.

Compared with Fig. 2, after adding the above nonlinear mapping layers, the architecture of the generalized DPLS (GDPLS) model is shown in Fig. 3. Note that when $\phi(x)$ is the identity function, i.e., $\phi(x) \equiv x$, the GDPLS will degenerate to the basic DPLS model.

The nonlinear mapping function ϕ is a key step in **Algorithm 1** and Fig. 3. In this article, the method we used to implement nonlinear mapping is the so-called Nyström technique [26], which is a general approach for low-rank approximations of kernels. The Nyström method can approximate various explicit nonlinear mappings of the corresponding implicit kernel functions, such as radial basis function (rbf) kernel, polynomial kernel, and Laplacian kernel. Unlike neural networks whose nonlinear activation function must be differentiable, a special advantage of the proposed framework is that the nonlinear mapping function can be either differentiable or nondifferentiable. Since the proposed framework does not rely on backpropagation and gradient descent to train the model, the optimization of DPLS and GDPLS need not calculate the differentials. Such an advantage means the proposed framework can try various mapping functions no matter they are differentiable or not, actually, the Nyström method we use is able to approximate nondifferentiable kernel functions.

As mentioned earlier, going deep is a good way to increase model complexity; therefore, Section II-B utilizes the hierarchical and cascaded model structure to build DPLS. On the other hand, the nonlinear PLS is the other manner to boost model complexity. Introducing nonlinear mapping ϕ further expands the complexity of basic DPLS, making the representation ability of GDPLS more powerful than DPLS. Various nonlinear mapping functions also bring diversity and flexibility to the GDPLS model. Different GDPLS models can be established by selecting different mapping functions, especially, different nonlinear mapping functions that can be applied to different layers of the same GDPLS model. By doing so, the model diversity of GDPLS is enhanced, which may lead to performance improvement. However, in this article, we would like to put focus on the deep model structure. In order to emphasize the role of deep structure, it is necessary to control the influence of other factors as much as possible. Therefore, for each GDPLS model structure of this article, the nonlinear mapping function of each layer is set to the same. In this situation, if the model performance increases as the number of layers increases, we can basically conclude that the proposed deep PLS structure is effective.

One question worth exploring is that does *Theorem 1* still hold for GDPLS? According to the *proof*, the key steps to make *Theorem 1* tenable are *Lemma 1*, (18) and (19). More specifically, the first latent variable $\boldsymbol{u}_1^{(1)}$ in the first layer must be one of the variables that fed into the next PLS decomposition process in the second layer. In the DPLS model structure (Fig. 2), $\boldsymbol{u}_1^{(1)}$ happens to be the first column of the input data matrix ($\boldsymbol{U}^{(1)}$) of the next PLS decomposition process, thus the required condition is naturally met. According to Fig. 3, in the GDPLS, due to the mapping operation, the extracted latent vector $\boldsymbol{u}_1^{(1)}$ will not be directly fed into the next PLS decomposition process. Instead, the input data matrix of the second PLS process is the mapped feature matrix $\boldsymbol{H}^{(2)}$. Therefore, the model structure of GDPLS does not meet the required condition, which means *Theorem 1* may not be tenable for GDPLS. However, we can still make *Theorem 1* hold for any form of GDPLS through a simple trick. That is, after obtaining the mapped feature matrix $\boldsymbol{H}^{(2)}$, the latent variable $\boldsymbol{u}_1^{(1)}$ in the previous layer is concatenated to be one of the columns of $\boldsymbol{H}^{(2)}$ (e.g. $\boldsymbol{H}^{(2)} = [\boldsymbol{u}_1^{(1)}, \boldsymbol{h}_1^{(2)}, \boldsymbol{h}_2^{(2)}, \ldots, \boldsymbol{h}_{s2}^{(2)}]$). Then, the required condition will be met. Note that simply adding a previous feature $\boldsymbol{u}_1^{(1)}$ to $\boldsymbol{H}^{(2)}$ will not make much difference. Follow the above-mentioned operation, for $l = 1, 2, \ldots, L-1$, each $\boldsymbol{u}_1^{(l)}$ is concatenated to the mapped feature matrix $\boldsymbol{H}_1^{(l+1)}$ of the next layer. And through such procedures, *Theorem 1* will be tenable for any form of GDPLS.

Remark 1: Since the modeling procedures of DPLS only involve linear PLS cross decomposition, *Theorem 1* has great significance to DPLS. Although the transformations are all linear, *Theorem 1* proves that the hierarchical and cascaded PLS layers are beneficial to extract more representative latent variables. However, though we make *Theorem 1* still hold for GDPLS through a simple trick, the importance of *Theorem 1* to GDPLS may not be as significant as it to DPLS. Because in the GDPLS, a more important part is the nonlinear mapping ϕ. It has a more powerful influence on the performance of GDPLS for addressing nonlinear problems.

D. Computational Complexity

The computational complexity of nonlinear PLS mainly comes from the nonlinear mapping and the PLS decomposition. Denote the mapped matrix as $\boldsymbol{H} \in \mathbb{R}^{n\times s}$, then the Nyström mapping method will have a complexity of $\mathcal{O}(ns^2)$ [26]. As for the PLS decomposition, according to **Algorithm 1**, if the $solver$ is "iter" and the actual number of iterations is I_a, the complexity of PLS decomposition will be $\mathcal{O}(knsI_a)$. If the $solver$ is "svd," the complexity is $\mathcal{O}(knsd)$. Note that if there is only one predicted variable, then we will have $I_a = d = 1$, which means the complexity of these two solving manners is the same. But the "svd" solving manner will introduce unnecessary memory consumption to store all column vectors in matrix $\boldsymbol{M}$, so when there is only one predicted variable, we recommend using $solver$ = "iter". If $d > 1$, the required actual number of iterations I_a is usually greater than d, so in this situation, the form of $solver$ is recommended to be "svd."

The total complexity of **Algorithm 1** is $\mathcal{O}(ns^2 + knsd)$. For a GDPLS with total L_1 layers, the training procedures are equivalent to performing **Algorithm 1** for L_1 times; thus, the total complexity of GDPLS is $\mathcal{O}(ns(s + kd)L_1)$. For neural networks, suppose the training datasets are $\boldsymbol{X} \in \mathbb{R}^{n\times m}$ and $\boldsymbol{Y} \in \mathbb{R}^{n\times d}$, and the networks have L_2 hidden layers with each layer containing $h_l (l = 1, \ldots, L_2)$ neurons. Denote the epochs as e, the complexity of training the above-mentioned networks will be $\mathcal{O}(enmd\prod_{l=1}^{L_2} h_l)$. In real applications, the term $e\prod_{l=1}^{L_2} h_l$ contributes a lot to the computations. In general, due to the term $e\prod_{l=1}^{L_2} h_l$, we have $\mathcal{O}(enmd\prod_{l=1}^{L_2} h_l) > \mathcal{O}(ns(s + kd)L_1)$.

In addition, the complexity of PLS and DPLS are $\mathcal{O}(knmd)$ and $\mathcal{O}(knmdL_3)$ (L_3 means the number of layers), respectively. Another popular model used in this article—support vector machines (SVM) has a complexity between $\mathcal{O}(n^2m)$ and $\mathcal{O}(n^3m)$ [27]. In general, the complexity of the above-mentioned methods have a relationship of $\mathcal{O}_{\text{DNN}} > \mathcal{O}_{\text{SVM}} > \mathcal{O}_{\text{GDPLS}} > \mathcal{O}_{\text{DPLS}} > \mathcal{O}_{\text{PLS}}$.[2]

III. Results

In this Section, we evaluate the effectiveness of the proposed methods through four case studies, including two regression problems and two classification tasks. By comparing GDPLS (or DPLS) with other current mainstream deep learning methods, the superior performance and high modeling efficiency of our models are demonstrated. After this, the fourth part will try to show how the proposed methods improve model performance through the results of case studies, i.e., to analyze the interpretability of DPLS and GDPLS models.

In order to evaluate the performance, for regression problems, we examine the following two indicators: root-mean-squared error (RMSE) and the coefficient of determination R^2, respectively, which are defined as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n_t}\sum_{i=1}^{n_t}(y_i - \hat{y}_i)^2}, \quad R^2 = 1 - \frac{\sum_{i=1}^{n_t}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n_t}(y_i - \bar{y}_i)^2} \tag{27}$$

where n_t is the number of testing samples, $\hat{y}$ is the predicted output value, and $\bar{y}$ is the mean of output values of all testing samples. A small RMSE and a large R^2 usually represent a good performance model. On the other hand, in classifying, the classification accuracy is used to measure the performance.

A. NIR Spectra of Corn Samples

The first case is the near-infrared (NIR) spectral data, which has the characteristics of *small samples* and *high-dimensional features*. The spectral dataset is the Cargill corn dataset whose descriptions and download can be acquired from http://www.eigenvector.com/data/Corn/. After removing the outliers, this dataset consists of 78 samples, 700 features, and one predictive target (each sample's starch content). The dataset is randomly split into 39 training samples and 39 testing samples.

The relationship between spectral features and target is considered to be approximately linear according to previous research [28]. Therefore, we built a four hidden layers DPLS model (without nonlinear mapping layers). The training procedures can be found in Fig. 2, and the model structure is [700, **35**, **15**, **9**, **6**, 1]. The bold numbers represent the number of latent variables in each hidden layer of DPLS, while the nonbold numbers are the input and output. The reason why the number of latent variables in the first hidden layer jumps directly to 35 is that: in the PLS algorithm, the number of latent variables should be no larger than rank [input matrix] = 39. The latent variables in each layer of the DPLS model are similar to the neurons in each hidden layer of DNN.

The performance of stacked autoencoders (SAE) and DNN are also provided. SAE is a classical pretraining and fine-tuning neural networks method. The performance of traditional PLS and its variant interval PLS (IPLS) [28]—which is a mature method in the field of spectral calibration—are also examined. The hidden layer numbers of SAE and DNN are both 3, and the specific model structures are: SAE: [700, **300**, **150**, **80**, 1] and DNN: [700, **100**, **80**, **30**, 1]. The bold value is the number of neurons in each hidden layer.

The RMSE and R^2 obtained by the above methods in the testing dataset are listed in Table I. The number in brackets of each method indicates the number of latent variables (for PLS-based methods) or the number of neurons (for neural networks-based methods) in the final hidden layer. Generally speaking,

[2]The real running time may not follow this inequality when the sample size n is quite large or small. For a large enough n, the running time of SVM may be greater than DNN. If n is quite small, the running time of SVM may be less than GDPLS and DPLS.

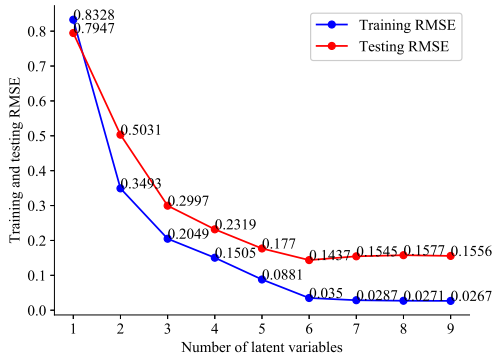


Fig. 4. Relationship between training and testing RMSE and the number of latent variables in the last hidden layer.

neural network-based methods require a certain number of neurons to learn useful information, and a relatively large number of neurons usually help to improve the fitting ability. Therefore, the neurons of SAE and DNN are much more than the latent variables of PLS-based methods. Actually, we have tried different model structures and trained DNN and SAE a lot of times, and the results of DNN and SAE reported in Table I are the best results among dozens of experiments. However, the performance of DNN and SAE is still significantly worse than other methods. We think the poor performance of DNN and SAE is because the training procedures of neural networks need to rely on a large number of samples. While in this test case, the training samples are quite small, which leads neural networks hard to learn useful and intrinsic information from data. In other words, the existing deep learning methods are not suitable for the problems with a small sample size. Contrarily, the PLS-based methods perform well in such situations. Especially, with only six latent variables for predicting, the DPLS model achieves the best results.

The last hidden layer of DPLS undertakes the task of forecasting output. The number of latent variables of this layer has an important influence on the final prediction performance. The relationship between training and testing RMSE and the number of latent variables in the last hidden layer is shown in Fig. 4. After the number of latent variables reaches 6, the training RMSE only decreases slightly, which indicates that increasing the model complexity (i.e. the number of latent variables) will not improve the model performance, contrarily, it may bring overfitting problems. On the other hand, looking at the testing RMSE, when the number of latent variables is greater than 6, the prediction accuracy even decreases slightly, which indicates the model may have already fallen into the state of overfitting. In fact, though the performance of six variables is the best, the fitting concern has already slightly appeared at five variables. Because from five variables to six variables, the drop degree of training RMSE is obviously greater than testing RMSE.

In each hidden layer of DPLS, a different amount of latent variables is extracted; finally, the latent variables of the last hidden layer are used for predicting. However, the latent variables in other layers can also be used to forecast output.

TABLE I
RESULTS FOR CORN SPECTRAL DATASET

Method	RMSE	R^2	Time/s
PLS (6)	0.3802	0.7636	0.13
IPLS (6)	0.2415	0.9046	1.45
SAE (80)	0.3684	0.7782	12.62
DNN (30)	0.4757	0.6300	8.04
DPLS (6)	0.1437	0.9662	0.54

Specifically, the testing RMSE achieved by these four hidden layers of DPLS are

$$0.1647(35) \rightarrow 0.1613(15) \rightarrow 0.1556(9) \rightarrow 0.1437(6).$$

The number in the bracket indicates the number of latent variables in this layer. Even when the number of latent variables used in each layer is gradually reduced, the prediction accuracy is improved instead, which shows that the proposed DPLS model can extract more inherent data features. DPLS can extract intrinsic data latent structure while simultaneously greatly compressing data dimension ($700 \rightarrow 6$).

We not only evaluated the performance of each model but also investigated the running time of them, which is listed in the last column of Table I. The running time is hardly a problem for traditional machine learning methods, so PLS and IPLS both perform well on this indicator. On the other hand, although SAE, DNN, and DPLS are all deep models, the running time of DPLS is much shorter than SAE and DNN. The running time depends on the model complexity, and the latter mainly comes from the model parameters and the model optimization strategy. According to their model structure, we can find that the model parameter scale of SAE, DNN, and DPLS is in the same order of magnitude. However, the difficulty of training their parameters varies a lot, which is closely related to their optimization strategy. The model parameters in DPLS are obtained through solving (1), and the optimization process will be performed only one time. And once obtained, they will be fixed. For SAE and DNN, their optimization needs to repeatedly solve the gradients and backpropagate them during the training procedures. Forward propagation and backpropagation will be repeated many times, and the model parameters of neural networks need to be iteratively updated in each backpropagation process, which is very time-consuming. In addition, gradient vanishing and gradient exploding may occur in this process. In a nutshell, although DPLS, SAE, and DNN have similar parameter scales, the parameters of DPLS can be fixed after only one optimization process, and the parameters in SAE and DNN will be fixed only when all training epochs are finished. The gradient-based optimization strategies of SAE and DNN make their parameters harder to obtain than those of DPLS, which also causes their running time much longer than DPLS.

A large amount of data usually serve as a necessary condition for training successful neural networks. This has a close relationship with the gradient-based optimization strategy of neural networks. Neural networks start off with a poor initial state and then the gradient-based algorithm is used to optimize the networks through iterations. This process requires a lot of

data, especially when the number of parameters is big. Instead, the optimization procedures of DPLS do not require iterative updates. Besides, the base learner of DPLS—PLS is especially good at handling problems with small sample capacity [11]. The above-mentioned merits of DPLS make its dependency on data amount not as severe as neural networks. The results, in this case, show that when dealing with a small dataset, the performance of DPLS is much superior to neural networks.

B. KPI Prediction in Industrial Process

The second experiment is an industrial key performance index (KPI) prediction problem, which is also known as *soft sensor* [29]–[31]. The industrial process researched here is the primary reformer comes from the hydrogen manufacturing units in the ammonia synthesis process. For more detailed descriptions of this task, please refer to [30]. The KPI of this industrial unit is the oxygen concentration, and the object is to predict it as accurately as possible. To develop the regression prediction model, 13 easy-measured process variables of this unit, such as flows, pressures, and temperatures, are chosen as inputs, while the oxygen content is the output [30]. A total of 2000 samples are collected for evaluation, 1000 of them are used as a training set, and the other 1000 are used as testing samples.

Considering that the real industrial processes are mostly nonlinear, the model used here is GDPLS. We established three hidden layers of GDPLS, and according to the principle and training procedures of GDPLS, there are also three nonlinear mapping layers corresponding to three different hidden layers. It is worth noting that in order to emphasize the role of deep structure and control the influence of other factors as much as possible, the parameters of the three nonlinear mapping layers are completely the same. Specific as follows, the Nyström method has a kernel function of Laplacian kernel whose form is $\text{laplacian}(x, y) = \exp(-\gamma \|x - y\|_1)$. The function hyperparameter γ is set to 1.0, and the dimensions of the mapped nonlinear spaces are all 500. Based on this, the model structure of GDPLS can be described as follows:

$$13 \Rightarrow \textit{500} \rightarrow \mathbf{10} \Rightarrow \textit{500} \rightarrow \mathbf{7} \Rightarrow \textit{500} \rightarrow \mathbf{4} \rightarrow 1$$

"$\Rightarrow$" symbol denotes the nonlinear mapping operations in Fig. 3, while "$\rightarrow$" represents PLS cross decomposition operations. Italic numbers represent the dimensions of the mapped nonlinear spaces, while bold values are the number of latent variables extracted by PLS in each hidden layer. In general, [13, 1] are the input and output, [**10**, **7**, **4**] are three hidden layers, and [*500*, *500*, *500*] are the transitional nonlinear feature layers.

SAE and DNN are again used for comparisons, the model structures of them are both [13, **10**, **7**, **4**, 1]. Another popular nonlinear regression method—support vector regression (SVR) is also introduced, the kernel of which is radial basis function. Two linear methods, PLS and DPLS, are also included to demonstrate the necessity of building a nonlinear GDPLS model, and the model structure of DPLS is the same as the above GDPLS's structure but without nonlinear layers.

The results are shown in Table II. The proposed GDPLS model surpasses traditional machine learning methods (PLS and SVR) a lot and has achieved competitive performance comparable to neural networks-based SAE and DNN. Meanwhile, the running time of GDPLS is much shorter than SAE and DNN. The different results of DPLS and GDPLS show that the nonlinear mapping layers make GDPLS suitable for nonlinear applications. Again, it is important to note that each hidden layer of GDPLS can construct a regression model. To provide insight on how the deep model structure improves prediction accuracy, the testing RMSE and R^2 of the three hidden layers of GDPLS are listed as follows:

$$\text{RMSE} : 1.1136 \rightarrow 0.7774 \rightarrow 0.7247$$
$$R^2 : 0.6218 \rightarrow 0.8157 \rightarrow 0.8399.$$

TABLE II
RESULTS FOR OXYGEN CONTENT PREDICTION DATASET

Method	RMSE	R^2	Time/s
PLS	1.446	0.3621	0.14
DPLS	1.277	0.5031	0.45
SVR	1.115	0.6208	0.24
SAE	0.7383	0.8338	38.32
DNN	0.7466	0.8300	22.59
GDPLS	0.7247	0.8399	1.33

C. Myocardial Infarction Identification and Classification

PLS is mainly used for regression problems, but it can also be utilized for classification. When used for classification, one should first encode the discrete class attributes numerically. The commonly used numerical encoding methods of discrete classes are dummy encoding, one-hot encoding, and so on.

Here, we use GDPLS to handle a classification problem. The object is to identify myocardial infarction (MI) according to the electrocardiogram (ECG) signals [32], [33]. The task is a binary classification with one class normal state and the other MI state. The ECG dataset [33] is a $14\,552 \times 187$ matrix, with each row representing an example. The total number of samples is 14 552, including 4046 normal samples and 10 506 MI samples. Each sample has 187 features, and all of the features are floating numbers. Since there are only two classes, the encoding of them is quite simple—the normal samples are labeled as -1 while the MI samples are marked as $+1$. We randomly selected 80% samples as training datasets and the rest 20% are used as testing datasets.

The model structure of GDPLS built for this problem is as follows:

$$187 \Rightarrow \textit{1500} \rightarrow \mathbf{100} \Rightarrow \textit{1500} \rightarrow \mathbf{60} \Rightarrow \textit{1500} \rightarrow \mathbf{10} \rightarrow 1.$$

The kernel functions of the Nyström method in all hidden layers are Laplacian kernel with a hyperparameter $\gamma = 0.05$. The model structures of DPLS, SAE, and DNN are all [187, **100**, **60**, **10**, 1]. SAE and DNN both have a dropout rate of 30% in all hidden layers.

The hidden layer numbers of the above-mentioned four deep models are all 3, and the numbers of latent variables (or neurons) are also the same. The results of PLS and support vector classification (SVC) with rbf kernel are also introduced.

TABLE III
RESULTS FOR MI CLASSIFICATION DATASET

Method	Accuracy (%)	Time/s
PLS	81.07	2.06
DPLS	81.31	6.27
SVC	92.37	12.26
SAE	96.32	234.83
DNN	95.91	156.61
GDPLS	96.15	55.41

The classification accuracy and running time are listed in Table III. The effectiveness of GDPLS is still available in classification cases, and the results indicate that GDPLS is a more precise modeling method compared with traditional methods, a more efficient modeling method compared with neural networks-based methods. Besides, since GDPLS performs much better than DPLS, we can tentatively conclude that the relationship between ECG signals and MI is nonlinear. Inside the GDPLS, the classification accuracy obtained by each hidden layer is

$$92.89\% \rightarrow 95.64\% \rightarrow 96.15\%.$$

D. MNIST Dataset

The results on the Mixed National Institute of Standards and Technology database (MNIST) dataset [4] are also evaluated in this article. The dataset contains 60 000 images of size 28×28 for training and 10 000 images for testing. The popular methods for recognizing MNIST are the CNNs-based methods. As the convolution operators can detect spatial features, which is quite beneficial for image classification. Directly being compared with CNNs is unfair for the GDPLS model, because the convolutional layers in CNNs serve as preprocessing steps of learning spatial features, whereas GDPLS has no ability to extract such spatial relationships. In general, to achieve high accuracy, feature engineering is usually needed for GDPLS to handle image-style data or sequence data. The raw inputs should first be transformed into features that reflect the spatial or sequential relationships; then, the extracted features are fed into the GDPLS model. Such preprocessing steps for GDPLS may involve convolution operations, time sliding windows, multigrain scanning [14], and so on. Such feature engineering works for GDPLS are left to future studies. For fair comparisons, we abandoned convolution processing for all methods, which means that each 28×28 input image is simply flattened into a sample with 784 features. Previous studies [4] have shown that high prediction accuracy still can be achieved for the MNIST dataset under such processing.

Ten classes of MNIST are encoded through one-hot encoding. The model structure of DNN is from [4], which has two hidden layers of [784, **300**, **100**, 10]. SAE has the same model structure as DNN, but has a dropout rate of 20% in all hidden layers. The GDPLS also has two hidden layers, specifically as

$$784 \Rightarrow \mathit{3000} \rightarrow \mathbf{100} \Rightarrow \mathit{3000} \rightarrow \mathbf{150} \rightarrow 10$$

TABLE IV
RESULTS FOR MNIST DATASET

Method	Accuracy (%)	Time/s	Time/s (GPU)
PLS	85.55	3.29	\
DPLS	86.45	15.89	0.82
SVC	97.92	291.25	\
SAE	98.23	707.44	151.64
DNN	96.95	479.68	133.44
GDPLS	98.11	195.05	20.95

TABLE V
RESULTS FOR CROPPED MNIST DATASET (1000 TRAINING SAMPLES)

Method	Accuracy (%)	Time/s
PLS	76.47	0.36
DPLS	80.38	0.41
SVC	90.83	3.10
SAE	88.96	40.55
DNN	86.60	23.74
GDPLS	92.67	3.17

in which the kernel function of the Nyström method is rbf kernel. The performance of PLS, DPLS with a structure of [784, **100**, **20**, 10] and SVC (rbf kernel) are also reported.

All results are listed in Table IV,[3] in which the accuracy of DNN also comes from [4]. Though the training efficiency of PLS and DPLS is quite high, the accuracy of these two linear methods is intolerably poor. Among all nonlinear models, the accuracy of GDPLS is better than DNN and SVC, competitive to SAE. But the running time of GDPLS is much superior over other nonlinear models, especially the SAE. Since the sample size of the MNIST dataset is relatively large, to accelerate training, we also implement the GPU versions of four deep models. The running time of DPLS, GDPLS, DNN, and SAE on GPU is also listed in Table IV. Note that the GPU implementation can indeed greatly speed up the training for all models, but the efficiency improvement of GDPLS is much greater than that of DNN and SAE. This is because GDPLS uses all samples at one time to train the model, while DNN and SAE iteratively use mini-batch samples at each epoch to train, and loading data in such a mini-batch and iterative style will cost a lot of time.

To emphasize the influence of data amount on different models, we also conducted an experiment on a cropped MNIST dataset. The complete training dataset of MNIST has 60 000 samples, we selected the first 1000 samples to construct a new training set, and the original testing set with 10 000 samples are retained to evaluate performance. The training dataset is balanced with each category having about 100 samples. The model structure of all methods is the same as before, except DNN and SAE have a higher dropout rate with 30% to mitigate overfitting. The results on the cropped MNIST dataset are listed in Table V. When the amount of training samples is significantly reduced, the accuracy of SAE and DNN also decreases severely, while the performance of the proposed GDPLS dropped the least. Meanwhile, the GDPLS

[3]Note that the accuracy in Table IV is not state-of-the-art (SOTA) due to no spatial features have been used. Some CNNs-based models can achieve accuracy over 99% in the MNIST dataset.

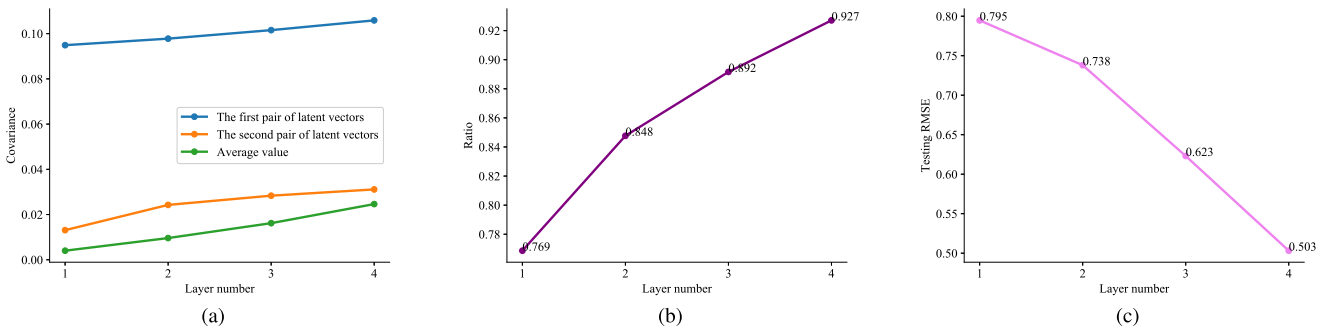


Fig. 5. Some indicators of the latent variables. (a) Relationship between covariance and layer number. (b) Relationship between layer number and ratio of the covariance of the first and second pair of latent variables to the total covariance. (c) Relationship between layer number and testing RMSE predicted by only the first two latent variables.

achieved the highest accuracy on the cropped dataset, which again demonstrates that GDPLS has obvious superiority over neural networks when the sample size is relatively small.

E. Model Interpretability

Different from the neural networks-based methods, another advantage of our model is its good interpretability. The former is like a "black box," we do not know how it gives correct results, and trying to interpret it requires novel and complex external approaches [20], [22]. However, due to the transparency of our model, this Section will base on the above case studies to show readers how the proposed method improves performance, and how it gives correct results.

As discussed, the core principle of PLS is to maximize the covariance between the projected vectors, and the DPLS has a merit that the covariance between the first pair of latent variables gradually increases with the increase of layers. To evaluate this theorem, under the corn spectral dataset, the changing relationship between layer number and the covariance of the first and second pair of latent vectors, and the average covariance of all pairs of latent vectors are shown in Fig. 5(a). This plot tells us that not only the covariance between the first pair of latent variables increases layer by layer as expected; the second pair also satisfies this rule. Most importantly, the average covariance of all pairs of latent vectors is also increased. The three indicators indicate that our method has a certain effect improvement theoretically.

Since the first and second pairs of latent variables confirm the changing rule, we will show some more metrics related to them. The relationship between layer number and ratio of the covariance of the first and second pair of latent variables to the total covariance is listed in Fig. 5(b). Besides, in the prediction stage, each layer only uses the first two latent variables to predict, the relationship between layer number and the corresponding testing RMSE is shown in Fig. 5(c). Fig. 5(b) shows that the ratio of the covariance of the first two pairs of latent variables to the total covariance is gradually increasing, which indicates that with the increase of layer number, the noise information is irrelevant to the final prediction is gradually eliminated. Fig. 5(c) shows that the RMSE has been significantly reduced in each layer when only the first two latent variables are used, which indicates the feature extraction is getting better and better with the increase of layers.

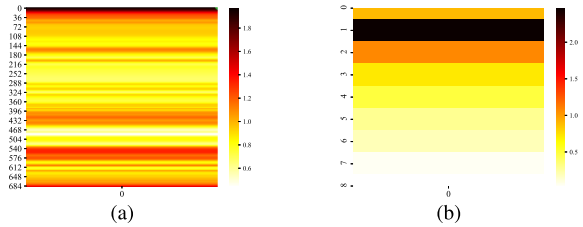
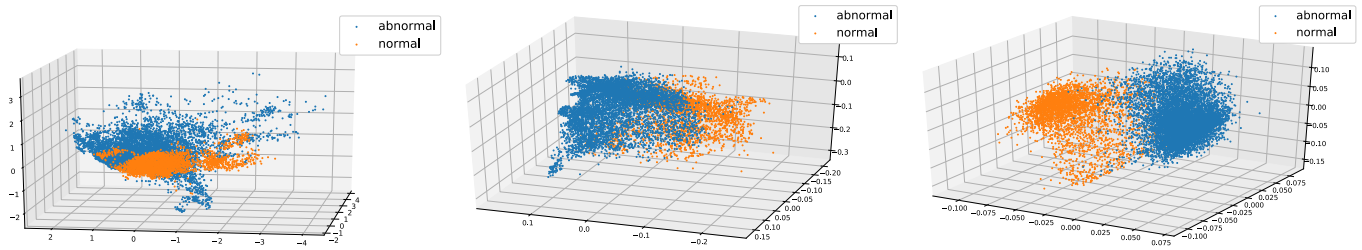


Fig. 6. VIP values of different layers presented by heat map. (a) VIP values of the inputs of the first layer (700 variables). (b) VIP values of the inputs of the fourth hidden layer (nine variables).

The VIP index presented in Section II-B provides us a tool for measuring variable and feature importance. In the corn spectra case study, the DPLS has a model structure of $700 \rightarrow \mathbf{35} \rightarrow \mathbf{15} \rightarrow \mathbf{9} \rightarrow \mathbf{6} \rightarrow 1$. In the form of a heat map, the variable importance of the inputs of the first layer (700 variables) and the inputs of the fourth hidden layer (nine variables) are presented in Fig. 6. The larger the VIP value, the darker the color in the heat map. Fig. 6(a) measures the importance of all variables in the original data; therefore, raw variables can be filtered according to this heat map. Specifically, since these are spectral wavelength features, we can retain several wavelength ranges with darker colors and remove the lighter ones. It can be expected that after such an operation, the prediction accuracy of the model may be further improved. Fig. 6(b) shows the importance of features in the fourth hidden layer. The importance of the last two or three variables is relatively low; therefore, we can consider further extracting latent variables by PLS for dimension reduction. This explains, to a certain extent, why we need to reduce the number of latent variables from 9 to 6 at the last hidden layer.

In addition, in Section II-B, we mentioned that data visualization can be conveniently realized in deep PLS methods through projecting data into the first two or three latent vectors. And this convenient visualization is especially meaningful in classification tasks. Because for a good PLS-based classification model, in the first 2-D or 3-D projection latent vector space, data points of different categories may already have clear separating hyperplanes. Here, this advantage will be evaluated through the MI classification case. Specifically, $\boldsymbol{u}_1^{(l)}, \boldsymbol{u}_2^{(l)}, \boldsymbol{u}_3^{(l)} (l = 1, 2, 3)$ are used as coordinate axes for realizing visualization in 3-D space. For PLS, nonlinear PLS



(a) (b) (c)

Fig. 7. Classification visualization results of three models. (a) PLS: Accuracy = 81.07%. (b) Nonlinear PLS: Accuracy = 92.89%. (c) GDPLS: Accuracy = 96.15%.

(implemented by **Algorithm 1**) and GDPLS, Fig. 7 selects the best perspectives to show the classification visualization results of the three models. The orange points indicate the normal condition and the blue points represent the abnormal condition (MI). The two classes of points in PLS [Fig. 7(a)] are almost completely mixed together, and they are distributed across each other in space, which may be the reason for the poor classification performance of the PLS model. For nonlinear PLS [Fig. 7(b)], the distribution of the two classes of points in space becomes more dispersed. Some of the points are located in two different directions in the space, but there are still many points mixed together. While in the GDPLS [Fig. 7(c)], the two classes of points are almost completely separated from each other. In fact, the Nonlinear PLS model here is the first layer of GDPLS. From PLS to the first layer of GDPLS then to the last layer of GDPLS, Fig. 7 informs us why the model performance is improved gradually.

IV. Conclusion

For quite some time, the rapid and salient progress of deep learning is accompanied by deficiencies that cannot be ignored. Taking some notable shortcomings of deep learning as the point of penetration, this article proposed a new concept called LDM. LDM aims to achieve competitive performance compared with current deep learning models in appropriate application scenarios, simultaneously overcoming the shortcomings of current deep models. In this work, we explored LDM from the perspective of the PLS model, thus proposing a novel deep learning method—GDPLS. The superiority of GDPLS is evaluated through four practical cases. The results show that the proposed GDPLS is a more interpretable and efficient data analytics method compared with neural networks-based deep models. The novelty of GDPLS can be summarized as follows.

1) Good interpretability: GDPLS is a deep model with transparency and simplicity. Through the analysis of the internal parts of GDPLS, we know exactly how it improves performance and how it gives correct answers.
2) Adaptability to data amounts: GDPLS not only achieves excellent results in the problems with sufficient data amounts but also performs well when the sample size is small. In fact, the performance of GDPLS on the small dataset is far superior to neural networks.
3) Model parameters easy to train: The parameters of GDPLS can be obtained after only one optimization process, but the parameters in neural networks will be fixed only when all training epochs are finished.
4) Low computation complexity and short running time: Due to the concise principle and simple training procedures of GDPLS, this method has a low computation complexity and its running time is quite short.

In general, GDPLS can achieve satisfactory performance, while simultaneously maintaining good interpretability and high modeling efficiency. We believe that this work may open up a new field for the research on deep learning, and deep models are no longer the antithesis of interpretability and efficiency. Under the concept presented in this article, one can explore new ways to construct transparent deep models with good interpretability and high efficiency, at the same time, the performance of which is also excellent. This work uses PLS and hierarchical model structure to successfully visualize the concept of LDM. In addition to it, this concept can definitely be probed through other approaches, such as combining other unique ideas of deep learning with else latent variable models.

On the other hand, the proposed deep PLS framework can also be further developed. It is worth noting that our model can only be regarded as an alternative to fully connected neural networks at present, and it cannot completely replace the existing large vision or language models because of having no steps to systematically address unstructured data. Therefore, a useful improvement is to integrate spatial or temporal information processing steps into the current model. Besides, as mentioned in Section II-C, in order to boost the diversity and flexibility of our model, different nonlinear mapping functions and different function parameters can be utilized instead of fixing all nonlinear mapping layers to the totally same setting. Moreover, a special advantage of the proposed framework is that we can use either differentiable or nondifferentiable mapping functions. In the future, we will consider using more diverse mapping functions in experiments. And we think that introducing variously differentiable and nondifferentiable activation functions in the modeling process, instead of just using Relu or sigmoid may help us to extract more representative nonlinear features. The deep PLS framework now serves as an ensemble of feature extraction and label prediction; however, it can also be used as a standalone feature extractor. The latent variables output from each PLS layer can be treated as features transferring into other models for different purposes.

REFERENCES

[1] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, Nov. 2019.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, Feb. 2015.

[3] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 604–624, Feb. 2021.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.

[8] I. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.

[9] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[10] E. Tjoa and C. Guan, "A survey on explainable artificial intelligence (XAI): Toward medical XAI," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 11, pp. 4793–4813, 2021, doi: 10.1109/TNNLS.2020.3027314.

[11] S. Wold, M. Sjöström, and L. Eriksson, "PLS-regression: A basic tool of chemometrics," *Chemometrics Intell. Lab. Syst.*, vol. 58, no. 2, pp. 109–130, 2001.

[12] D. Ballabio and V. Consonni, "Classification tools in chemistry. Part 1: Linear models. PLS-DA," *Anal. Methods*, vol. 5, no. 16, pp. 3790–3798, 2013.

[13] Z. Ge, Z. Song, L. Zhao, and F. Gao, "Two-level PLS model for quality prediction of multiphase batch processes," *Chemom. Intell. Lab. Syst.*, vol. 130, pp. 29–36, Jan. 2014.

[14] Z.-H. Zhou and J. Feng, "Deep forest," 2017, *arXiv:1702.08835*.

[15] G. Trigeorgis, K. Bousmalis, S. Zafeiriou, and B. W. Schuller, "A deep matrix factorization method for learning attribute representations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 3, pp. 417–429, Mar. 2017.

[16] W. Zhao, C. Xu, Z. Guan, and Y. Liu, "Multiview concept learning via deep matrix factorization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 814–825, Feb. 2021.

[17] H. Wold, "Soft modelling by latent variables: The non-linear iterative partial least squares (NIPALS) approach," *J. Appl. Probab.*, vol. 12, no. S1, pp. 117–142, 1975.

[18] L. Eriksson, T. Byrne, E. Johansson, J. Trygg, and C. Vikström, *Multi- and Megavariate Data Analysis Basic Principles and Applications*, vol. 1. USA: Umetrics Academy, 2013.

[19] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, pp. 2579–2605, Nov. 2008.

[20] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2013, *arXiv:1312.6034*.

[21] M. Du, N. Liu, Q. Song, and X. Hu, "Towards explanation of DNN-based prediction with guided feature inversion," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 1358–1367.

[22] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2921–2929.

[23] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual explanations from deep networks via gradient-based localization," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 618–626.

[24] R. Rosipal and L. J. Trejo, "Kernel partial least squares regression in reproducing kernel Hilbert space," *J. Mach. Learn. Res.*, vol. 2, pp. 97–123, Mar. 2001.

[25] J. A. Wegelin, "A survey of partial least squares (PLS) methods, with emphasis on the two-block case," Dept. Statist., Univ. Washington, Seattle, WA, USA, Tech. Rep. 371, 2000.

[26] C. K. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, pp. 682–688.

[27] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, p. 27, 2011. [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm

[28] W. Ni, S. D. Brown, and R. Man, "Stacked partial least squares regression analysis for spectral calibration and prediction," *J. Chemometrics*, vol. 23, no. 10, pp. 505–517, Oct. 2009.

[29] Q. Sun and Z. Ge, "A survey on deep learning for data-driven soft sensors," *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, pp. 5853–5866, Sep. 2021.

[30] L. Yao and Z. Ge, "Moving window adaptive soft sensor for state shifting process based on weighted supervised latent factor analysis," *Control Eng. Pract.*, vol. 61, pp. 72–80, Apr. 2017.

[31] X. Kong and Z. Ge, "Adversarial attacks on neural-network-based soft sensors: Directly attack output," *IEEE Trans. Ind. Informat.*, vol. 18, no. 4, pp. 2443–2451, 2022, doi: 10.1109/TII.2021.3093386.

[32] A. L. Goldberger *et al.*, "PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[33] M. Kachuee, S. Fazeli, and M. Sarrafzadeh, "ECG heartbeat classification: A deep transferable representation," in *Proc. IEEE Int. Conf. Healthcare Informat. (ICHI)*, Jun. 2018, pp. 443–444.

[34] X. Kong and Z. Ge, "Deep learning of latent variable models for industrial process monitoring," *IEEE Trans. Ind. Informat.*, p. 1, 2021, doi: 10.1109/TII.2021.3134251.

Xiangyin Kong received the B.Eng. degree in marine engine engineering from the School of Naval Architecture and Ocean Engineering, Huazhong University of Science and Technology, Wuhan, China, in 2019. He is currently pursuing the Ph.D. degree in control science and engineering with the State Key Laboratory of Industrial Control Technology, College of Control Science and Engineering, Zhejiang University, Hangzhou, China.

His research interests include machine learning, deep learning, model security, and data-driven industrial modeling.

Zhiqiang Ge (Senior Member, IEEE) received the B.Eng. and Ph.D. degrees in automation from the Department of Control Science and Engineering, Zhejiang University, Hangzhou, China, in 2004 and 2009, respectively.

He was a Research Associate with the Department of Chemical and Biomolecular Engineering, The Hong Kong University of Science Technology, from 2010 to 2011, and a Visiting Professor with the Department of Chemical and Materials Engineering, University of Alberta, Edmonton, AB, Canada, in 2013. He was an Alexander von Humboldt Research Fellow with the University of Duisburg-Essen, Duisburg, Germany, from 2014 to 2017, and the Japan Society for the Promotion of Science (JSPS) Invitation Fellow with Kyoto University, Kyoto, Japan, in 2018. He is currently a Full Professor with the State Key Laboratory of Industrial Control Technology, College of Control Science and Engineering, Zhejiang University, and the Peng Cheng Laboratory, Shenzhen, China. His research interests include industrial big data, process monitoring, soft sensor, data-driven modeling, machine intelligence, and knowledge automation.