Adaptive group Lasso neural network models for functions of few variables and time-dependent data

Lam Si Tung Ho · Nicholas Richardson · Giang Tran

Received: 23 September 2022 / Accepted: 27 July 2023
© The Author(s), under exclusive licence to Springer Nature Switzerland AG 2023

Abstract
Learning nonlinear functions from time-varying measurements is always difficult due to the high correlation among observations. This task is more challenging when the target function is high dimensional. In this work, we propose a new method to learn high dimensional functions which depends only on a few unknown coordinates from a set of time-varying measurements. More precisely, we approximate the target function by a neural network and enforce an adaptive group Lasso constraint on the suitable weight matrix to represent the low-dimensional property of the unknown function. Using the non-negative property of the Bregman distance, we show that the proposed optimization procedure achieves loss decay. Our empirical studies show that the proposed method outperforms recent state-of-the-art methods including the sparse dictionary matrix method, and neural networks with or without group Lasso penalty.

Keywords Neural networks · Function approximation · Sparse optimization · Time-dependent data

Mathematics Subject Classification 65P99 · 37M99 · 65D40

1 Introduction
Function approximation is a fundamental problem that lies at the interface between applied mathematics and machine learning. The main goal of this problem is to approximate an unknown underlying nonlinear function from a collection of inputs and
outputs. Typically, we assume the unknown function belongs to a certain function space such as the spaces of polynomials, trigonometric functions, or Fourier terms. The underlying nonlinear function can be written as a linear combination of those basis terms and the unknown coefficients can be found via solving a least square fitting problem. In another direction, we can approximate the underlying function as a composition of linear and simple nonlinear functions, namely neural networks [2, 14]. Recently, deep neural networks have been shown as one of the best methods for function approximation [12, 18, 28, 29, 35, 36, 44, 47].

The problem of function approximation becomes even more challenging for high-dimensional data. The general idea for handling this scenario is to add a regularization term such as the Tikhonov, $\ell_1$, or $\ell_0$-like penalties in order to capture active variables. For example, polynomial approximations with sparse coefficients have been proven theoretically and numerically to be one of the most effective dictionary methods in the high-dimensional setting [1, 4, 8, 22, 39, 42]. Neural networks, on the other hand, often utilize group penalties to force all weights associated with an inactive variable to zero together. For example, in [40], the authors propose to use group penalties for the weights in all layers of the neural network model to identify important features and prune the network simultaneously. Smoothing group $\ell_1$-penalties have been applied to the weights of the hidden layer in a single-hidden-layer neural network to identify useful features [50]. In [16], the authors propose to use adaptive group Lasso for analytic deep neural networks. Their simulations suggest that adaptive group Lasso is better than group Lasso in removing inactive variables. Adaptive group Lasso has also been used to learn directed graphs from time-dependent data [3] and to prune neural network architectures [19].

Another important aspect of the function approximation problem is the properties of data. The most common setting assumes that data are independent and identically distributed (iid). However, the independence assumption does not hold for many applications such as epidemiology, evolutionary biology, financial prediction, and signal processing. Therefore, much effort has been devoted to study function approximation for non-iid data. One popular type of non-iid data that has been extensively studied is mixing sequences [10, 17, 21, 22, 41, 48]. However, mixing property is not a natural assumption to make and verifying this property is often difficult. Time-dependent data is another popular type of non-iid data, such as data generated from a dynamical system. There are connections between dynamical systems and mixing properties but figuring them out is extremely difficult and worth a separate study. There is a great interest recently in learning the underlying governing equations from time-dependent data using both dictionary-based methods [4, 13, 25, 30, 38, 45] and the neural network methods [24, 37, 43]. Further details about system identification and its applications can be found in [26].

In this paper, we focus on the function approximation problem when data are generated from a high-dimensional dynamical system and the unknown smooth nonlinear function depends on few active variables. We propose to approximate the target function by a deep neural network with an adaptive group Lasso constraint to the weights of the first hidden layer. It has been proven recently that adaptive group Lasso can correctly identify active variables with high probability [16]. We also extend our method to the case when the target function depends on a few linear combinations of vari-
ables, which is related to active subspace methods [7, 9, 11, 20]. For optimizing the regularized loss function, we apply the popular proximal algorithm [33]. Using the non-negative property of the Bregman distance, we show that our proposed optimization procedure achieves loss decay. Specifically, if the learning rate is sufficiently small, the regularized loss always improves after each iteration and the estimated value of the parameters converges to the set of critical points of the regularized loss function. We conduct an extensive empirical study to compare the performance of the proposed method with two state-of-the-art methods: the neural network models (with/without group Lasso penalty) and dictionary matrix method with the ℓ₁-constraint to the coefficients. Our experiments demonstrate that neural networks with adaptive group Lasso outperform the other methods in learning various nonlinear functions where the input data are generated from a popular dynamical system, the Lorenz 96 system. These results illustrate the advantage of adaptive group Lasso over the regular group Lasso in selecting active variables for neural network models.

The paper is organized as follows. We explain the problem setting and present the proposed algorithm for learning functions of few active variables in Sect. 2. We also prove the loss decay property of the proposed optimization algorithm here. In Sect. 3, we show how to extend our method to the case when the target function depends on few linear combinations of variables and state the corresponding algorithm. Numerical results and comparisons are described in Sect. 4.

Our contributions:

- We propose a new framework to learn the governing function of a non-linear dynamical system in a high-dimensional setting. Our framework approximates the target function by a deep neural network model and imposes its sparsity constraint using adaptive group Lasso.
- We obtain the loss decay property for our proposed optimization procedure using the non-negative property of the Bregman distance.
- We design several simulation studies to demonstrate that our method outperforms existing state-of-the-art methods including the neural network models (with/without group Lasso penalty) and dictionary matrix method with the ℓ₁-constraint to the coefficients.

2 Problem setting and algorithm

2.1 Problem setting

Consider the problem of learning an unknown nonlinear smooth function \( f : \mathbb{R}^d \rightarrow \mathbb{R} \) from a set of input–output samples \( \{(x_i, y_i)\}_{i=1}^m \), where the input data \( \{x_i\}_{i=1}^m \) are (possibly noisy) time-dependent and the function \( f \) depends on a few active variables, \( f(x) = f(x | g) \) with \( g \subset \{1, 2, \ldots, d\} \) unknown. In our settings, we assume the input data are noisy observations of a high-dimensional dynamical system with the error governed by the noise level \( \sigma_x \),

\[
x_i = \tilde{x}(t_i) + \varepsilon_x M_x, \quad t_1 < t_2 < \cdots < t_m,
\]
where $\{\tilde{x}(t_i)\}_{i=1}^m$ are noiseless solutions of the dynamical system, $M_x$ is the maximum absolute value of the noiseless training inputs, and $\varepsilon_x \sim \mathcal{N}(0, \sigma^2_x I_d)$. The output data is also noisy with the error governed by the noise level $\sigma_y$:

$$y_i = f(\tilde{x}(t_i)) + \varepsilon_y M_y,$$

where $M_y$ is the maximum absolute value of the noiseless training outputs and $\varepsilon_y \sim \mathcal{N}(0, \sigma^2_y)$. Our goal is to find a good approximation of the nonlinear function $f$ and to accurately identify the active variables.

To achieve this goal, one of the popular methods is to approximate the nonlinear function $f$ by a feed-forward neural network:

$$f \approx F_\theta(x) = W_L(\sigma(W_{L-1}(\cdots \sigma(W_2(\sigma(W_1 x + b_1) + b_2) \cdots) + b_{L-1}) + b_L),$$

where $L$ is the number of layers of the neural network and $\theta = \{(W_l, b_l)\}_{l=1}^L$ are trainable weights and biases. Since the target function $f$ is smooth, we choose the activation function $\sigma$ to be the tanh. To identify these active variables, we impose the sparsity constraint on the weight of the first hidden layer $W_1 \in \mathbb{R}^{H_1 \times d}$, where $H_1$ is the number of nodes in the first hidden layer. More precisely, suppose $I_1 \subset \{1, \ldots, d\}$ be the index set of inactive variables. Then, the columns of $W_1$ associated with those indices in $I_1$ must be zero vectors. We enforce this constraint on $W_1$ by using a group Lasso penalty on $W_1$, where the entries of each column are grouped together. However, it has been shown empirically that applying the standard group Lasso for neural networks is not effective in producing sparse solutions [3, 16, 19]. On the other hand, it is shown theoretically in [16], using an adaptive (weighted) group Lasso for certain analytic deep networks is feature selection consistent, which guaranteed the sparsity of the learned weight matrix in the first hidden layer.

An illustration of an adaptive group Lasso neural network is presented in Fig. 1. In [16], the authors consider a simpler scenario where $f$ is an analytic deep neural network and the approximated function $F_\theta(x)$ has the same structure as $f$. Moreover, the inputs are iid and have no noise. Under this setting, they prove that the adaptive sparse group Lasso can select the correct set of active variables with high probability. As a consequence, we can recover the underlying function $f$ as the number of inputs increases. Although the result is only for a simple case, it suggests that the adaptive sparse group Lasso may also work for our setting. We will confirm this observation using simulation studies in Sect. 4.

Specifically, the adaptive sparse group Lasso aims to solve the following optimization problem:

$$\hat{\theta} \in \arg\min_{\theta} \mathcal{L}(\theta) + \mathcal{R}(W_1),$$

where

$$\mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^m |y_i - F_\theta(x_i)|^2,$$

$$\mathcal{R}(W_1) = \sum_{l=1}^L \lambda_l \sum_{k=1}^{H_l} \sum_{j=1}^d \lambda_{jk} W_{l,k}^j.$$
An illustration of an adaptive group Lasso neural network: all weights connecting each input variable with nodes in the first hidden layer are grouped together (weights in the same circle are in the same group). The colors of the eclipses represent whether the associated variable is active (blue) or inactive (black).

and

\[ R(W_1) = \lambda_m \sum_{j=1}^{d} \frac{1}{\| \tilde{W}_1[:,j] \|^2} \| W_1[:,j] \|. \]

Here, \( \lambda_m \) is a regularizer factor and \( W_1[:,j] \) denotes the \( j \)th column of \( W_1 \). Throughout this work, we use the Euclidean norm when we write \( \| \cdot \| \) without a trailing subscript. The initial estimator \( \tilde{W}_1 \) is the weight matrix of the first hidden layer of the neural network, which is obtained from solving the following non-linear least square problem:

\[ \tilde{\theta} \in \arg \min_{\theta} \mathcal{L}(\theta). \]

For convention, if \( \tilde{W}_1[:,j] = 0 \), we assign \( W_1[:,j] = 0 \). In practice, it is often that all columns of the initial estimator \( \tilde{W}_1 \) are non-zero vectors.

2.2 Algorithm

The optimization problem (1) can be solved using the proximal algorithm [33]:

\[ \theta^{(k+1)} = \text{prox}_{\tau_k R} \left[ \theta^{(k)} - \tau_k \nabla_{\theta} \mathcal{L}(\theta^{(k)}) \right] \]
where

\[
\text{prox}_{\tau k \mathcal{R}}(\bar{\theta}) = \arg\min_{\theta} \mathcal{R}(W_1) + \frac{1}{2\tau_k} \|\theta - \bar{\theta}\|^2_F,
\] (3)

and \(\bar{\theta} = \theta^{(k)} - \tau_k \nabla_{\theta} \mathcal{L}(\theta^{(k)})\). The closed form solution of (2) is:

\[
W^{(k+1)}_1[:, j] = \max \left( 0, \frac{\lambda_m \tau_k}{\|W_1[:, j]\|} \right) \frac{W_1[:, j]}{\|W_1[:, j]\|}, \quad \text{for } j = 1, \ldots, d,
\]

\[
W^{(k+1)}_l = W_l, \quad \text{for } l = 2, \ldots, L,
\]

\[
b^{(k+1)}_l = b_l, \quad \text{for } l = 1, \ldots, L
\]

where \(W\) and \(b\) are the weights and biases of \(\bar{\theta}\).

Finally, in the preprocessing step, the training dataset is standardized by dividing each variable with the sample standard deviations of the corresponding variables. That is,

\[
y_{i}^{\text{new}} := \frac{y_i}{\alpha} \quad \text{and} \quad x_{i,j}^{\text{new}} := \frac{x_{i,j}}{\sigma_j}, \quad \text{for } j = 1, \ldots, d,
\] (4)

where \(\alpha\) is the sample standard deviation of the output dataset \(\{y_i\}_{i=1}^m\), \(x_i = (x_{i,1}, \ldots, x_{i,d})^T\), and \(\sigma_j\) is the sample standard deviation of the \(j\)th variable of the input dataset \(\{x_{i,j}\}_{i=1}^m\). The method is summarized in Algorithm 1.

### 2.3 Loss decay property

Denote \(J(\theta) = \mathcal{R}(W_1), \mathcal{F}(\theta) = \mathcal{L}(\theta) + J(\theta)\), and

\[
\mathcal{L}(\theta; \mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{(x_i, y_i) \in \mathcal{B}} |y_i - F_\theta(x_i)|^2,
\]

where \(\mathcal{B}\) is a mini-batch of training data. Recall that a mini-batch is a subset of training data. Stochastic gradient descent uses mini-batches to approximate the gradient calculated from the entire data set. In this section, we provide convergence analysis of the proximal gradient step in Algorithm 1. Specifically, we prove that under some mild assumptions, the total loss \(\mathcal{F}(\theta)\) decays. We also illustrate the assumption verification for one-hidden layer networks.

Using the non-negative property of the Bregman distance, we derive a slightly stronger result than the one from [49].

**Theorem 1** Assume the following conditions:

1. (Lipschitz Smoothness). The function \(\mathcal{L}\) is \(C\)-smooth. That is, the function \(\mathcal{L}\) is continuously differentiable and there exists a constant \(C > 0\) such that

\[
\|\nabla \mathcal{L}(\theta) - \nabla \mathcal{L}(\phi)\| \leq C \|\theta - \phi\|. \quad \forall \theta, \phi \in \text{dom } \mathcal{L}.
\]
Algorithm 1

1: **Input:** Samples \( m \) input-output pairs \( \{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^d \times \mathbb{R} \), number of layers of the feed-forward neural network \( L \), number of nodes in each layer \( (H_1, \ldots, H_{L-1}) \), maximum numbers of epochs \( \text{epochMax}_1 \) and \( \text{epochMax}_2 \), regularization hyperparameter \( \lambda_m \), and step size \( \tau_k \).

2: **Preprocess:** Standardize training data by their sample standard deviations:

\[
y_i^{\text{new}} := \frac{y_i}{\alpha} \quad \text{and} \quad x_{i,j}^{\text{new}} := \frac{x_{i,j}}{\sigma_j}, \quad \text{for} \quad j = 1, \ldots, d,
\]

where \( \alpha \) and \( \sigma_j \) are sample standard deviations of \( \{y_i\}_{i=1}^m \) and \( \{x_{i,j}\}_{i=1}^m \), respectively.

3: **procedure** **Find the initial estimators** \( \tilde{\theta} \):

\[
\tilde{\theta} \in \arg\min_{\theta} \mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^m |y_i - F_\theta(x_i)|^2.
\]

4: **Initialize** \( \tilde{\theta}^{(0)} \)

5: **for** \( k < \text{iterMax}_1 \) **do**

6: \( \tilde{\theta}^{(k+1)} = \text{ADAM}(\tilde{\theta}^{(k)}) \)

7: **procedure** **Solve the Adaptive Group Lasso model:**

\[
\hat{\theta} \in \arg\min_{\theta} \mathcal{L}(\theta) + \mathcal{R}(W_1) = \frac{1}{m} \sum_{i=1}^m |y_i - F_{\hat{\theta}}(x_i)|^2 + \lambda_m \sum_{j=1}^d \frac{1}{\|W_1[:, j]\|^2} \|W_1[:, j]\|.
\]

8: **Initialize** \( \theta^{(0)} \)

9: **while** \( k < \text{iterMax}_2 \) **do**

10: \( \hat{\theta}^{(k)} = \theta^{(k)} - \tau_k \nabla_{\theta} \mathcal{L}(\hat{\theta}^{(k)}) \)

11: \( W_1^{(k+1)}[:, j] = \max \left( 0, \frac{\|\hat{W}_1[:, j]\| - \frac{\lambda_m \tau_k}{\|\hat{W}_1[:, j]\|^2} \|\hat{W}_1[:, j]\|}{\|\hat{W}_1[:, j]\|^2} \|\hat{W}_1[:, j]\| \right), \quad \text{for} \ j = 1, \ldots, d. \)

12: \( W_l^{(k+1)} = \hat{W}_l, \quad \text{for} \ l = 2, \ldots, L. \)

13: \( b_l^{(k+1)} = \hat{b}_l, \quad \text{for} \ l = 1, \ldots, L. \)

14: **Output:** \( \hat{\theta} = \theta^{(\text{epochMax}_1)}, \hat{f}(x) = F_{\hat{\theta}}(x); \) and the support set \( \hat{S} = \{ j : \|\hat{W}_1[:, j]\| > 0 \}. \)

2. (Bounded Variance). There exists a constant \( \mu > 0 \) such that for any trainable parameters \( \hat{\theta} \), we have

\[
\mathbb{E} \left[ \|\nabla \mathcal{L}(\theta; B) - \nabla \mathcal{L}(\hat{\theta})\|^2 \right] \leq \mu.
\]

3. The regularization function \( J(\theta) \) is convex and proper.

Then the stochastic proximal gradient algorithm with the step sizes \( \tau_k \leq \frac{1}{C} \) for all \( k \),

\[
g^{(k)} \leftarrow \nabla_{\hat{\theta}} \mathcal{L}(\hat{\theta}^{(k)}; B^{(k)}),
\]

\[
\overline{\theta}^{(k+1)} \leftarrow \theta^{(k)} - \tau_k g^{(k)},
\]

\[
\theta^{(k+1)} \leftarrow \text{prox}_{\tau_k J}(\overline{\theta}^{(k+1)}),
\]
achieves the total loss decay. That is,

\[
E \left[ F(\theta(k+1)) \right] + \left( \frac{1}{\tau_k} - C \right) E \left[ \| \theta(k+1) - \theta(k) \|^2 \right] \leq E \left[ F(\theta(k)) \right] + \frac{\mu}{2C}.
\]  

(6)

In particular, if we use the full batch, then \( \mu = 0 \) and we have the total loss decay:

\[
F(\theta(k+1)) + \left( \frac{1}{\tau_k} - C \right) \| \theta(k+1) - \theta(k) \|^2 \leq F(\theta(k)).
\]  

(7)

Proof See Appendix A. \qed

Using Theorem 1, we also achieve a similar estimation of the convergence of the stochastic proximal algorithm as in Theorem 1 [49].

Theorem 2 Assume that \( \tau_k \in [\gamma_1, \gamma_2] \), where \( 0 < \gamma_1 < \gamma_2 < \frac{1}{C} \) and all assumptions in Theorem 1 are satisfied. Then

\[
\frac{1}{K} \sum_{k=0}^{K-1} E \left[ \text{dist}(0, \partial F(\theta(k+1)))^2 \right] \leq \left( 3 + \frac{3B}{2C} \right) \mu + \frac{3B}{K} E[F(\theta(0)) - F(\theta(K))],
\]  

(8)

where \( B = \max_{t \in [\gamma_1, \gamma_2]} \left( \frac{t^2 + C^2}{t^2 - C} \right) \).

Proof See Appendix B. \qed

Finally, we want to verify that in certain situations, the function \( L(\theta) \) satisfies the Lipschitz smoothness assumption of Theorems 1 and 2. First, recall the definition of \( L_f \)-Lipschitz.

Definition 1 A function \( f : V \to W \) between normed vector spaces \((V, \| \cdot \|_V)\) and \((W, \| \cdot \|_W)\) is \( L_f \)-Lipschitz if we have

\[
\| f(a) - f(b) \|_W \leq L_f \| a - b \|_V \quad \forall a, b \in V,
\]  

(9)

where \( L_f \) is a finite, non-negative real number.

Theorem 3 Consider a one-hidden layer network,

\[
F_\theta(x) = W_2 \sigma(W_1 x + b_1) + b_2,
\]

where \( W_1 \in \mathbb{R}^{h \times d}, W_2 \in \mathbb{R}^{n \times h}, b_1 \in \mathbb{R}^h, b_2 \in \mathbb{R}^n, x \in \mathbb{R}^d, y \in \mathbb{R}^n \), and \( \sigma \) is a \( C_\sigma \)-smooth and \( C_\sigma \)-Lipschitz non-linear activation function. Then the associated mean squared error loss \( L(\theta) \) over all data points \( \{(x_i, y_i)\}_{i=1}^m \) is \( C \)-smooth with

\[
C = \max_{1 \leq i \leq m} 2(h + 1) \left( 1 + C_\sigma C_1 \sqrt{n} \| \hat{x} \|_2 \right) \sqrt{C_1^2 n(h + 1) \| \hat{x} \|_2^2 + 1}
\]
\[ = \mathcal{O}(nC_\sigma C_1^2 h^{3/2} \|\hat{x}\|_2^2) \] (10)

where \( C_1 \) is an upper bound for the magnitude of the entries in the outer layer and \( \|\hat{x}\|_2 = \sqrt{\|x\|_2^2 + 1} \).

**Proof** See Appendix C. \(\square\)

**Remark 1** Hyperbolic tangent, \( \tanh(x) \), and sigmoid, \( (1 + e^{-x})^{-1} \), are examples of one-Lipschitz and one-smooth non-linear activation functions since the magnitude of their first and second derivatives are bounded above by one.

### 3 Extension to \( f(x) = g(Ax) \)

Next, we propose an adaptation of our proposed method to the scenario where the underlying function \( f(x) \) depends only on a few (possibly unknown) linear combinations of variables:

\[
 f(x) = g \left( \sum_{j=1}^{d} a_{1,j} x_j, \sum_{j=1}^{d} a_{2,j} x_j, \ldots, \sum_{j=1}^{d} a_{k,j} x_j \right) = g(Ax),
\]

where the number of linear combinations \( k \), the matrix \( A = (a_{i,j}) \in \mathbb{R}^{k \times d} \), and the smooth function \( g : \mathbb{R}^k \to \mathbb{R} \) are unknown. We can rewrite the target function as follows:

\[
 f(x) = g(Ax) = \tilde{g}(W_1 x),
\]

where \( \tilde{g} : \mathbb{R}^{H_1} \to \mathbb{R} \) has only a few active variables and the set of column vectors of \( A \) is a subset of the set of column vectors of \( W_1 \in \mathbb{R}^{H_1 \times d} \). Let \( z = W_1 x \), then \( f(x) = \tilde{g}(z) \). Applying the procedure in Sect. 2, we approximate \( \tilde{g}(z) \) by

\[
 W_L(\sigma(W_{L_1}(\cdots(\sigma(W_2(\sigma(W_1x + b_1) + b_2) + b_3)\cdots) + b_{L_1}) + b_L).
\]

Since \( \tilde{g} \) has only a few active variables, we penalize the first layer of this neural network, which is equivalent to penalizing the second layer of the original network. An illustration of an adaptive group Lasso neural network that represents a function depending on a few linear combinations of variables is given in Fig. 2.

Adapting the method in Sect. 2, we approximate \( f \) by the following neural network

\[
 f \approx G_{\theta}(x) = W_L(\sigma(W_{L-1}(\cdots(\sigma(W_2(\sigma(W_1x + b_1) + b_2) + b_3)\cdots) + b_{L-1}) + b_L),
\]

where \( \sigma_1 \) is the identity mapping and \( \sigma \) is the tanh. Since \( f \) depends only on few linear combinations of \( x \), we would like to enforce the weights connecting nonactive variables with the second hidden-layer to zero.
The corresponding optimization problem is

\[
\hat{\theta} \in \arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} |y_i - G_\theta(x_i)|^2 + R(W_2).
\]

where

\[
R(W_2) = \lambda_{m} \sum_{j=1}^{d} \frac{1}{\|\hat{W}_2[:, j]\|^2} W_2[:, j],
\]

and the initial estimator \(\hat{W}_2\) is obtained from solving the following non-linear least square problem:

\[
\tilde{\theta} \in \arg\min_{\theta} \left( \frac{1}{m} \sum_{i=1}^{m} |y_i - G_\theta(x_i)|^2 \right).
\]

Again, we can use the proximal algorithm (see Eq. (2)) to solve the optimization problem (11). Specifically, the weights and biases in all layers except the second layer are updated by using a step of gradient descent. On the other hand, the weight matrix
in the second layer is updated using the proximal solution:

\[ W_2 = W_2^{(k)} - \tau_k \frac{\partial}{\partial W_2} \mathcal{L}(\theta^{(k)}), \]
\[ W_2^{(k+1)}[:, j] = \max \left( 0, \frac{\|W_2[:, j]\|}{\|W_2^{(k)}[:, j]\|} \right), \]
\[ \frac{\lambda_m \tau_k}{\|W_2^{(k)}[:, j]\|^2} \frac{\|W_2[:, j]\|}{\|W_2^{(k)}[:, j]\|}. \]

We summarize the method in Algorithm 2.

**Algorithm 2**

1: **Input:** Samples \( m \) input-output pairs \( \{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^d \times \mathbb{R} \), number of layers of the feed-forward neural network \( L \), number of nodes in each layer \( (H_1, \ldots, H_{L-1}) \), maximum numbers of epochs \( \text{epochMax1} \) and \( \text{epochMax2} \), regularization hyperparameter \( \lambda_m \), and step size \( \tau_k \).

2: **Preprocess:** Standardize training data by their sample standard deviations:
\[ y_i^{\text{new}} := \frac{y_i}{\alpha} \quad \text{and} \quad x_{i, j}^{\text{new}} := \frac{x_{i, j}}{\sigma_j}, \quad \text{for} \quad j = 1, \ldots, d, \]
where \( \alpha \) and \( \sigma_j \) are sample standard deviations of \( \{y_i\}_{i=1}^m \) and \( \{x_{i, j}\}_{i=1}^m \), respectively.

3: **procedure** FIND THE INITIAL ESTIMATORS \( \tilde{\theta} \):
\[ \tilde{\theta} \in \arg\min_{\theta} \mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^m |y_i - G_{\theta}(x_i)|^2. \]

4: **Initialize** \( \tilde{\theta}^{(0)} \)
5: **for** \( k < \text{epochMax1} \) **do**
6: \[ \tilde{\theta}^{(k+1)} = \text{ADAM}(\tilde{\theta}^{(k)}) \]
7: **procedure** SOLVE THE ADAPTIVE GROUP LASSO MODEL:
\[ \hat{\theta} \in \arg\min_{\theta} \mathcal{L}(\theta) + \mathcal{R}(W_2) \]
8: **Initialize** \( \theta^{(0)} \)
9: **while** \( k < \text{epochMax2} \) **do**
10: \[ \hat{\theta} = \theta^{(k)} - \tau_k \nabla_{\theta} \mathcal{L}(\theta^{(k)}) \]
11: \[ W_2^{(k+1)}[:, j] = \max \left( 0, \frac{\|W_2[:, j]\|}{\|W_2^{(k)}[:, j]\|} \right), \quad \text{for} \quad j = 1, \ldots, d. \]
12: \[ W_l^{(k+1)} = \tilde{W}_l, \quad \text{for} \quad l = 1, 3, 4, \ldots, L. \]
13: \[ b_l^{(k+1)} = \tilde{b}_l, \quad \text{for} \quad l = 1, 3, 4, \ldots, L. \]
14: **Output:** \( \hat{\theta} = \theta^{(\text{epochMax2})} \) and \( \hat{f}(x) = G_{\hat{\theta}}(x) \).

4 **Numerical results**

In this section, we demonstrate the performance of our proposed algorithm for input data generated from high-dimensional systems of ODEs. We apply our method with 3-hidden-layer neural networks where each hidden layer has \( H \) nodes. We measure
the performance of our proposed method using sensitivity, specificity, and relative test error. The sensitivity and specificity are defined as follows:

\[
\text{Sensitivity} = \frac{\text{true positive}}{\text{positive}},
\]

\[
\text{Specificity} = \frac{\text{true negative}}{\text{negative}},
\]

where \text{true positive} is the number of variables that are correctly selected, \text{true negative} is the number of variables that are correctly not selected, \text{positive} is the number of variables in the true model, and \text{negative} is the number of variables that are not in the true model. For a test set \( T \), the relative test error of an algorithm is defined as follows:

\[
\sqrt{\sum_{x \in T} \| f(x) - \hat{f}(x) \|^2} / \sum_{x \in T} \| f(x) \|^2,
\]

where \( \hat{f}(x) \) denotes the approximation of \( f(x) \) by the proposed algorithms.

We compare the sensitivity, the specificity, and the relative test error of our algorithms with those that are obtained from the standard neural network without regularization, the group Lasso neural network [31, 32], and the Legendre polynomial-based (up to degree two) dictionary matrix method with sparse constraints [4, 42]. To find the regularization hyperparameters of group Lasso and adaptive group Lasso, we use the popular Bayesian information criterion (BIC) [46]. We consider five possible choices for the regularization parameter \( \lambda_m \in \{0.05, 0.1, 0.2, 0.3, 0.4\} \). The value returns the smallest BIC will be our final choice. The remaining hyperparameters are fixed throughout the simulations. Specifically, the number of epoches \( \text{epochMax1} \) and \( \text{epochMax2} \) are set to 50000. We initialize \( \overline{\theta}(0) \) and \( \theta(0) \) from corresponding uniform distributions. The learning rate of Adam [23] (Line 6 in Algorithm 1 and Algorithm 2) and the step size \( \tau_k \) of the gradient descent step (Line 10 in Algorithm 1 and Algorithm 2) are both 0.005. Finally, all weights and coefficients of the final results whose absolute values are smaller than \( 10^{-4} \) are set to 0.

### 4.1 Data simulation

To simulate the input data, we first solve the Lorenz-96 system [27] numerically using \texttt{lsoda} method [34] with the time step size \( \Delta t = 0.01 \), from \( t = t_0 = 0 \) to \( t = T_{\text{final}} = 80 \) and obtain the numerical solution \( \{\tilde{x}(t_i)\}_{i=1}^m \):

\[
\frac{dx_j}{dt} = -x_{j-2} x_{j-1} + x_{j-1} x_{j+1} - x_j + F, \quad j = 1, \ldots, d.
\]

Here \( x_0 = x_d \) and \( x_{d+1} = x_1 \). In our simulations, we choose \( d = 40, F = 8, m = \frac{T_{\text{final}} - t_0}{\Delta t} = 8000, \) and \( t_i = i \Delta t, \) for \( i = 1, \ldots, m \). The initial conditions
are \( x_j(0) = 1 \) for all \( j \neq 20 \) and \( x_{20}(0) = 1.008 \). Then we add Gaussian noise with standard deviation \( \sigma_x \) to the numerical solution of the Lorenz 96 and obtain the input data \( \{ x_i \}_{i=1}^m \):

\[
x_i = \tilde{x}(t_i) + \varepsilon_x M_x,
\]

where \( M_x \) is the maximum absolute value of the noiseless training inputs \( t \in [0, 8] \), and \( \varepsilon_x \sim \mathcal{N}(0, \sigma_x^2 I_d) \). The simulated output data are

\[
y_i = f(\tilde{x}(t_i)) + \varepsilon_y M_y,
\]

where \( M_y \) is the maximum absolute value of the noiseless training outputs \( t \in [0, 8] \) and \( \varepsilon_y \sim \mathcal{N}(0, \sigma_y^2) \). The nonlinear function \( f \) will be specified in each experiment.

To validate those methods, we examine the learned functions on the validation set of 2000 samples \( \{ x_{test}^i, y_{test}^i \}_{i=1}^{2000} \), where \( \{ x_{test}^i \}_{i=1}^{2000} \) is the numerical solution of the Lorenz 96 system from \( t = 80 \) to \( t = 100 \) and \( y_{test}^i = f(x_{test}^i) \).

### 4.2 Effect of the size of neural networks

Here, we compare the performance of our proposed method with respect to \( H \), the number of nodes in each hidden layer. In this comparison, we assume that both inputs and outputs of the training data are noisy with \( \sigma_x = \sigma_y = 0.02 \). We vary the number of nodes \( H \) (\( H = 10, 20, 40, 80 \)) in each hidden layer while keeping the size of the training set to be 8000 as mentioned in Sect. 4.1. For each \( H \), we generate 100 different data sets with the same noise level and the same following target function:

\[
f(x) = -x_{23}x_{24} + x_{24}x_{26} - x_{25} + 8,
\]

which is the right hand side of the 25th equation of the Lorenz 96 system. The results are presented in Table 1. In all cases, the sensitivities are one, which means every neural network is able to select all active variables. Also, the specificities are all greater than 0.8, which means all neural networks successfully remove most of the inactive variables. Moreover, the relative test errors are almost the same of around 6\%, while the neural network with \( H = 20 \) provides the highest specificity of 92.7\%. Therefore, in the remaining of the paper, we fix the number of nodes in each hidden layer \( H \) to be 20.

### 4.3 Effect of noise levels

In this section, we test the performance of our proposed algorithm, the standard neural network without regularization, a group Lasso neural network method, and the sparse Legendre polynomial-based (up to degree two) dictionary method in approximating the following function

\[
f(x) = -x_8x_9 + x_9x_{11} - x_{10} + 8,
\]
Table 1 The average sensitivities, specificities, and relative test errors of Algorithm 1 with different neural networks in approximating the function $f(x) = -x_{23}x_{24} + x_{24}x_{26} - x_{25} + 8$

| $H$ | # parameters | Sensitivity | Specificity | Rel. test error |
|-----|--------------|-------------|-------------|-----------------|
| 10  | 681          | 1           | 0.807       | 0.051           |
| 20  | 1681         | 1           | 0.927       | 0.062           |
| 40  | 4961         | 1           | 0.87        | 0.067           |
| 80  | 16,321       | 1           | 0.79        | 0.079           |

Fig. 3 Learned curves (in red) of the standard neural network (top left), the group Lasso neural network (top right), the sparse Legendre polynomial-based (up to degree two) dictionary method (bottom left), and our method (bottom right) versus the ground truth curve (in blue) when there is no noise in both input and output data with various noise levels. This function is the governing equation of the 10th equation of the Lorenz 96.

When there is no noise in both input and output data, the Legendre polynomial-based (up to degree two) dictionary method and our method provide a better approximation than the other two methods. Specifically, the relative test errors are 0.33, 0.28, 0.061, and $4.2 \times 10^{-5}$ for the standard neural network without regularization, a group Lasso neural network method, the sparse Legendre polynomial-based (up to degree two) dictionary method, and our method, respectively. The learned curves are plotted in Fig. 3. In this scenario, the Legendre polynomial-based dictionary method performs the best due to the fact that there is no noise and the true underlying function is polynomial. On the other hand, when both input and output data are noisy with noise level $\sigma_x = \sigma_y = 0.04$, our method performs slightly better than the Legendre polynomial-based dictionary method. The learned curves are shown in Fig. 4. In this case, our method outperforms the other three methods with the smallest relative test error of 0.22 and the sparse Legendre polynomial-based dictionary method achieves the second smallest relative test error of 0.29.

We also perform a thorough simulation to study the effect of noise levels on these methods. In this simulation, we vary the noise levels $\sigma_x = \sigma_y = 0.02, 0.03, 0.04, 0.05$ and for each noise level, we generate 100 different data sets. The average relative test errors for all methods are presented in Fig. 5. We can see that our proposed method
Fig. 4 Learned curves (in red) of the standard neural network (top left), the group Lasso neural network (top right), the sparse Legendre polynomial-based dictionary method (bottom left), and our method (bottom right) versus the ground truth curve (in blue) when the noise levels in input and output are $\sigma_x = \sigma_y = 0.04$.

Fig. 5 Average relative test errors and average specificities of the standard neural network, the group Lasso neural network, the sparse Legendre polynomial-based (up to degree two) dictionary method, and our method across different noise levels, $\sigma_x = \sigma_y = \sigma = 0.02, 0.03, 0.04,$ and $0.05$.

Our method outperforms other methods for all noise levels. The relative test errors of the group Lasso neural network and sparse Legendre polynomial-based dictionary method are similar. The standard neural network without regularization is the worse method, which emphasizes the importance of regularization in high dimensional problems. Moreover, the sensitivities of neural network and group Lasso neural network are always 1 while their specificities are always 0. That is, these models always include all variables in their model which aligns with the finding in [15, 16]. On the other hand, the specificities of our model and the sparse Legendre polynomial-based dictionary method are similar (see Fig. 5). However, the sensitivity of our model is always 1 while the sensitivity of the sparse Legendre polynomial-based dictionary method decreases from 1 to 0.8 as the noise levels increase. It means our model always captures all active variables and when the noise levels are high, the sparse Legendre polynomial-based dictionary method often misses about one out of four active variables.
4.4 Non-polynomial functions

Next, we examine the effectiveness of our algorithm to approximate non-polynomial functions. Specifically, we consider the following settings.

**Setting 1:** The underlying function is

\[ f_1(x) = (x_{19}^{4/3} - x_{16}^{4/3})x_{17}^{4/3} - x_{18}^{4/3} + 8. \]

In this setting, the inputs are noiseless \((\sigma_x = 0)\) while the outputs has Gaussian noise with standard deviation \(\sigma_y = 0.02\).

**Setting 2:** The underlying function is

\[ f_2(x) = (e^{x_{19}/50} - e^{x_{16}/50})e^{x_{17}/50} - e^{x_{18}/50} + 8. \]

In this case, the outputs are noiseless \((\sigma_y = 0)\) while the inputs has Gaussian noise with standard deviation \(\sigma_x = 0.02\).

**Setting 3:** The underlying function is

\[ f_3(x) = (e^{x_{19}/10} - x_{16}^{2/3})x_{17} - x_{18}^{4/5} + 8. \]

Here, both inputs and outputs are corrupted by noises \((\sigma_x = \sigma_y = 0.02)\).

In all settings, \(x \in \mathbb{R}^{40}\) is generated from the Lorenz-96 system as mentioned in Sect. 4.1. The learned curves are presented in Fig. 6 and the average relative rest errors over 100 simulated data for each setting are shown in Table 2. In all settings, our method outperforms the sparse Legendre polynomial-based (up to degree two)
Table 2  Relative test errors of the sparse Legendre polynomial-based dictionary method and of our method in approximating functions in Settings 1, 2, and 3

| Setting | Our method | Sparse Legendre polynomial-based dictionary method |
|---------|------------|----------------------------------------------------|
| 1       | 0.141      | 0.844                                              |
| 2       | 0.0009     | 0.007                                              |
| 3       | 0.146      | 0.622                                              |

Fig. 7  Learned curve (in red) of the sparse Legendre polynomial-based (up to degree two) dictionary method (left) and of our method (right) versus ground-truth curve (in blue) in approximating $f(x) = g(Ax)$, where $g(z) = (z_4 - z_1)z_2 - z_3 + 8$, $z = (z_1, z_2, z_3, z_4)$, and $A \in \mathbb{R}^{4 \times 40}$

dictionary method. We note that relative errors of the second setting are smaller than those of other cases because the value of the underlying function is larger. Additionally, the sensitivities and specificities of the sparse Legendre polynomial-based dictionary method are always 1 and 0, respectively, indicating that this method cannot capture the active variables. On the other hand, the sensitivities of our method are 1, 0.915, 1 for three settings and the specificities are 0.889, 0.9997, 0.927, respectively.

4.5 The case $f(x) = g(Ax)$

In this experiment, we examine the underlying function $f(x) = g(Ax)$ depends only on a few linear combinations of variables. Here, the matrix $A \in \mathbb{R}^{4 \times 50}$ is a Gaussian random matrix and $g : \mathbb{R}^4 \rightarrow \mathbb{R}$ is given as follows:

$g(z) = (z_4 - z_1)z_2 - z_3 + 8$,  $z \in \mathbb{R}^4$.

The inputs $x \in \mathbb{R}^{40}$ are generated from the Lorenz-96 system with no noise and the outputs are

$y = f(x) + \varepsilon_y M_y = g(Ax) + \varepsilon_y M_y$,  $\varepsilon_y \sim \mathcal{N}(0, 0.02^2)$.

For this experiment, the training process is the same as previous sections. The learned curves from our method and the sparse Legendre polynomial-based (up to degree two) dictionary method are visualized in Fig. 7. Clearly, our proposed method outperforms the sparse Legendre polynomial-based dictionary method in approximating the underlying function.
5 Conclusion

We proposed a new neural network framework for high-dimensional function approximation with time-dependent data where the underlying function is nonlinear and depends only on few active variables. By utilizing an adaptive group Lasso penalty to the weights of the first hidden layer of the neural network, our model can identify the active variables with high accuracy. We also proposed an adaptation of this framework to the scenario where the underlying function depends only on few linear combinations of variables. In this case, we apply the group Lasso penalty to the weights of the second (instead of the first) hidden layer of the neural network. Through various experiments, we show that our method outperforms recent popular function approximation methods including the sparse Legendre polynomial-based dictionary method, the standard neural network, and the group Lasso neural network. In addition, we proved that the proposed optimization procedure is guaranteed to achieve loss decay.

In [16], the authors proved that Adaptive group Lasso can select active variables of analytic neural networks with high probability when data are iid. A future research direction is to provide similar result for high-dimensional time-dependent data. Another direction is to utilize adaptive group Lasso to automatically resize the neural network structures so that the learned function has the sparsest architecture.

Acknowledgements LSTH was supported by startup funds from Dalhousie University, the Canada Research Chairs program, the NSERC Discovery Grant, and the NSERC Discovery Launch Supplement. GT was supported by the NSERC Discovery Grant and the NSERC Discovery Launch Supplement. We thank Hans De Sterck for his helpful feedback which led to significant improvements.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

A Proof of Theorem 1

Let $p^{k+1} = \frac{\theta^{(k+1)} - \theta^{(k+1)}}{\tau_k} \in \partial J(\theta^{(k+1)})$. Then, by the definition of proximal operator,

$$ s^k = -\frac{\theta^{(k+1)} - \theta^{(k)}}{\tau_k} = -p^{k+1} - \frac{\theta^{(k+1)} - \theta^{(k)}}{\tau_k}. \quad (13) $$

Using similar argument as in Theorem 3.2 [6] and in Lemma 1 [49], we have:

$$ \mathcal{L}(\theta^{(k)}) \leq \langle \nabla \mathcal{L}(\theta^{(k)}), \theta^{(k+1)} - \theta^{(k)} \rangle + \frac{C}{2} \|\theta^{(k+1)} - \theta^{(k)}\|^2 $$

$$ = \langle s^k, \theta^{(k+1)} - \theta^{(k)} \rangle + \langle \nabla \mathcal{L}(\theta^{(k)}) - s^k, \theta^{(k+1)} - \theta^{(k)} \rangle + \frac{C}{2} \|\theta^{(k+1)} - \theta^{(k)}\|^2 $$

Birkhäuser
\begin{align*}
&= (p^{k+1}, \theta^{(k)} - \theta^{(k+1)}) + \langle \nabla \mathcal{L}(\theta^{(k)}) - g^k, \theta^{(k+1)} - \theta^{(k)} \rangle + \left( \frac{C}{2} - \frac{1}{\tau_k} \right) \| \theta^{(k+1)} - \theta^{(k)} \|^2 \\
&= -D_j^{p^{k+1}}(\theta^{(k)}, \theta^{(k+1)}) + J(\theta^{(k)}) - J(\theta^{(k+1)}) + \langle \nabla \mathcal{L}(\theta^{(k)}) - g^k, \theta^{(k+1)} - \theta^{(k)} \rangle \\
&\quad + \left( \frac{C}{2} - \frac{1}{\tau_k} \right) \| \theta^{(k+1)} - \theta^{(k)} \|^2,
\end{align*}

where the first inequality is due to the Lipschitz smoothness of the function \( \mathcal{L} \) (with constant \( C \)), the second equality comes from the substitution of \( g^k \) (Eq. (13)), and the last equality is obtained from the Bregman distance formula [5]:

\[
D_j^{p^{k+1}}(\theta^{(k)}, \theta^{(k+1)}) = J(\theta^{(k)}) - J(\theta^{(k+1)}) - \langle p^{k+1}, \theta^{(k)} - \theta^{(k+1)} \rangle,
\]

for \( p^{k+1} \in \partial J(\theta^{(k+1)}) \).

Using Cauchy–Schwarz inequality and Young inequality, we have

\[
\mathcal{F}(\theta^{(k+1)}) + D_j^{p^{k+1}}(\theta^{(k)}, \theta^{(k+1)}) + \left( \frac{1}{\tau_k} - \frac{C}{2} \right) \| \theta^{(k+1)} - \theta^{(k)} \|^2 \\
\leq \mathcal{F}(\theta^{(k)}) + \langle \nabla \mathcal{L}(\theta^{(k)}) - g^k, \theta^{(k+1)} - \theta^{(k)} \rangle \\
\leq \mathcal{F}(\theta^{(k)}) + \nu \frac{1}{2} \| \nabla \mathcal{L}(\theta^{(k)}) - g^k \|^2 + \frac{1}{2\nu} \| \theta^{(k+1)} - \theta^{(k)} \|^2,
\]

for any \( \nu > 0 \). Since \( J(\cdot) \) is convex and proper, the Bregman distance \( D_j^{p^{k+1}}(\theta^{(k)}, \theta^{(k+1)}) \) is always nonnegative. Therefore,

\[
\mathcal{F}(\theta^{(k+1)}) + \left( \frac{1}{\tau_k} - \frac{C}{2} - \frac{1}{2\nu} \right) \| \theta^{(k+1)} - \theta^{(k)} \|^2 \\
\leq \mathcal{F}(\theta^{(k)}) + \nu \| \nabla \mathcal{L}(\theta^{(k)}) - g^k \|^2.
\]

Choosing \( \nu = \frac{1}{C} \) yields

\[
\mathcal{F}(\theta^{(k+1)}) + \left( \frac{1}{\tau_k} - C \right) \| \theta^{(k+1)} - \theta^{(k)} \|^2 \\
\leq \mathcal{F}(\theta^{(k)}) + \frac{1}{2C} \| g^k - \nabla \mathcal{L}(\theta^{(k)}) \|^2. \tag{14}
\]

Taking the expectation with respect to the mini-batches on both sides of Eq. (14), we obtain Eq. (6).

This proves the expectation of the function \( \mathcal{F}(\theta) = \mathcal{L}(\theta) + J(\theta) \) decreases with each iteration, up to the variance of our gradient approximation \( \nabla \mathcal{L}(\theta; \omega) \).
B Proof of Theorem 2

Proof From Eq. (13), we have

\[ -\frac{1}{\tau_k} (\theta^{(k+1)} - \theta^{(k)}) - g^k + \nabla \mathcal{L}(\theta^{(k+1)}) = p^{k+1} + \nabla \mathcal{L}(\theta^{(k+1)}) \in \partial \mathcal{F}(\theta^{(k+1)}). \]

Therefore,

\[
\text{dist}(0, \partial \mathcal{F}(\theta^{(k+1)}))^2 = \inf_{v \in \partial \mathcal{F}(\theta^{(k+1)})} \|0 - v\|^2_2 \\
\leq \left\| \frac{1}{\tau_k} (\theta^{(k+1)} - \theta^{(k)}) + g^k - \nabla \mathcal{L}(\theta^{(k+1)}) \right\|^2_2 \\
= \left\| g^k - \nabla \mathcal{L}(\theta^{(k)}) + \nabla \mathcal{L}(\theta^{(k)}) - \nabla \mathcal{L}(\theta^{(k+1)}) + \frac{1}{\tau_k} (\theta^{(k+1)} - \theta^{(k)}) \right\|^2_2 \\
\leq 3 \left\| g^k - \nabla \mathcal{L}(\theta^{(k)}) \right\|^2_2 + 3 \left\| \nabla \mathcal{L}(\theta^{(k)}) - \nabla \mathcal{L}(\theta^{(k+1)}) \right\|^2_2 + 3 \left\| \frac{1}{\tau_k} (\theta^{(k+1)} - \theta^{(k)}) \right\|^2_2 \\
\leq 3 \left\| g^k - \nabla \mathcal{L}(\theta^{(k)}) \right\|^2_2 + 3C^2 \left\| \theta^{(k)} - \theta^{(k+1)} \right\|^2_2 + 3\tau_k^{-2} \left\| \theta^{(k+1)} - \theta^{(k)} \right\|^2_2 \\
\leq 3 \left\| g^k - \nabla \mathcal{L}(\theta^{(k)}) \right\|^2_2 + 3 \left( C^2 + (\tau_k)^{-2} \right) \left\| \theta^{(k+1)} - \theta^{(k)} \right\|^2_2 \\
\leq 3 \frac{C^2 + (\tau_k)^{-2}}{(\tau_k)^{-1} - C} \left( \frac{1}{2C} \left\| g^k - \nabla \mathcal{L}(\theta^{(k)}) \right\|^2_2 + \mathcal{F}(\theta^{(k)}) - \mathcal{F}(\theta^{(k+1)}) \right),
\]

where the last inequality is obtained from Eq. (14).

Let \( B = \max_{t \in [\gamma_1, \gamma_2]} \frac{t^{-2} + C^2}{t^{-1} - C} = \max \left\{ \gamma_1^{-2} + C^2, \gamma_2^{-2} + C^2 \right\} < \infty. \) Then

\[
\text{dist}(0, \partial \mathcal{F}(\theta^{(k+1)}))^2 \leq 3 \left\| g^k - \nabla \mathcal{L}(\theta^{(k)}) \right\|^2_2 + 3B \left( \frac{1}{2C} \left\| g^k - \nabla \mathcal{L}(\theta^{(k)}) \right\|^2_2 + \mathcal{F}(\theta^{(k)}) - \mathcal{F}(\theta^{(k+1)}) \right) \\
\leq \left( 3 + \frac{3B}{2C} \right) \left\| g^k - \nabla \mathcal{L}(\theta^{(k)}) \right\|^2_2 + 3B \left( \mathcal{F}(\theta^{(k)}) - \mathcal{F}(\theta^{(k+1)}) \right). 
\]

Taking expectation with respect to the mini-batches, we have

\[
\mathbb{E}[\text{dist}(0, \partial \mathcal{F}(\theta^{(k+1)}))^2] \leq \left( 3 + \frac{3B}{2C} \right) \mu + 3B \mathbb{E}[\mathcal{F}(\theta^{(k)}) - \mathcal{F}(\theta^{(k+1)})].
\]

Therefore,

\[
\frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}[\text{dist}(0, \partial F(\theta^{(k+1)}))^2] \leq \left( 3 + \frac{3B}{2C} \right) \mu + \frac{3B}{K} \mathbb{E}[\mathcal{F}(\theta^{0}) - \mathcal{F}(\theta^{(k)})]. \quad \Box
\]
C Proof of Theorem 3

First, we introduce some notations.

Definition 2 Let \( f : V \rightarrow \mathbb{R}^{n \times p} \) be a bounded matrix-valued function between vector spaces \((V, \|\cdot\|_V)\) and \((\mathbb{R}^{n \times p}, \|\cdot\|_F)\). Define

\[
M_f = \sup_{v \in V} \|f(v)\|_F, \tag{15}
\]

where \(\|A\|_F\) is the Frobenius norm of a matrix \(A\).

Definition 3 Let \( f : V \rightarrow \mathbb{R}^{n \times p}, g : V \rightarrow \mathbb{R}^{p \times q}\) be matrix-valued functions defined on a normed vector space \((V, \|\cdot\|_V)\). Define the product \(f \star g : V \rightarrow \mathbb{R}^{n \times q}\) by

\[
(f \star g)(v) = f(v)g(v), \quad \forall v \in V, \tag{16}
\]

where \(f(v)g(v)\) is the matrix product of the matrices \(f(v)\) and \(g(v)\).

We can easily compute a Lipschitz constant of the matrix-valued function \(f \star g\).

Lemma 1 Let \( f : V \rightarrow \mathbb{R}^{n \times p}, g : V \rightarrow \mathbb{R}^{p \times q}\) be bounded and \(L_f, L_g\)-Lipschitz functions, respectively. Then \(f \star g\) is \((L_f M_g + L_g M_f)\)-Lipschitz.

Corollary 1 Let \( f_i : V \rightarrow \mathbb{R}^{m_i \times m_i+1}\) be bounded and \(L_{f_i}\)-Lipschitz. Then \(F = f_1 \star \ldots \star f_N\) is Lipschitz with constant \(L_F = \sum_{i=1}^{N} \left( L_{f_i} \prod_{j \neq i} M_{f_j} \right) \).

Proof This follows from repeatedly using Lemma 1 on the functions \(f_i\) and \((f_{i+1} \star \ldots \star f_N)\), and noting \(M_{f \star g} \leq M_f M_g\) for submultiplicative norms. \(\square\)

We now detail our notation convention for derivatives involving vectors and matrices.

Definition 4 Let \( y : \mathbb{R}^d \rightarrow \mathbb{R}^n\) be a vector-valued function with a vector input. The derivative with respect to its input is the \(n \times d\) Jacobian matrix

\[
\frac{\partial y}{\partial x} = \begin{bmatrix}
\frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_d} \\
\vdots & \ddots & \vdots \\
\frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_d}
\end{bmatrix},
\]

where \(y = [y_1, \ldots, y_n]^T\) and \(x = [x_1, \ldots, x_d]^T\).

Note that, if \(n = 1\), the derivative \(\frac{\partial y}{\partial x}\) is the transpose of the gradient

\[
\frac{\partial y}{\partial x} = \begin{bmatrix}
\frac{\partial y}{\partial x_1} \\
\vdots \\
\frac{\partial y}{\partial x_d}
\end{bmatrix} = (\nabla y)^T (x).
\]

On the other hand, if \(y = Ax\), we have \(\frac{\partial y}{\partial x} = A\).
Definition 5 Let $y : \mathbb{R}^{n \times p} \rightarrow \mathbb{R}$ be a scalar-valued function with a matrix input $X$. The derivative with respect to its $n \times p$ matrix input is the $p \times n$ matrix

$$\frac{\partial y}{\partial X} = \begin{bmatrix} \frac{\partial y}{\partial x_{11}} & \cdots & \frac{\partial y}{\partial x_{1p}} \\ \vdots & \ddots & \vdots \\ \frac{\partial y}{\partial x_{n1}} & \cdots & \frac{\partial y}{\partial x_{np}} \end{bmatrix}. $$

It is sometimes convenient to rewrite the affine function $T(x; W, b) := Wx + b$, where $x \in \mathbb{R}^d$, $b \in \mathbb{R}^h$, $W \in \mathbb{R}^{h \times d}$, in terms of a single matrix $\tilde{W} \in \mathbb{R}^{h \times (d+1)}$ as follows:

$$T(x; W, b) = \tilde{W} \tilde{x} = \begin{bmatrix} W \\ b \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}. $$

Using the chain rule, the partial derivatives of the one-hidden layer neural network function with respect to its weights and biases are given below.

**Proposition 1** Under the assumptions in Theorem 3, the derivative of

$$\ell(x, y; \theta) = \frac{1}{2} \| F_{\theta}(x) - y \|^2_2$$

with respect to its inner and outer layer parameters is given by

$$\frac{\partial \ell(\theta)}{\partial \tilde{W}_1} = \tilde{x}r^\top W_2 \text{diag}(\sigma'(z)), $$

$$\frac{\partial \ell(\theta)}{\partial \tilde{W}_2} = \tilde{s}r^\top,$$

where $r = F_{\theta}(x) - y$, $z = W_1x + b_1$, $s = \sigma(z)$, $\tilde{x}$ to denote the vector $x$ with an appended 1, and $\tilde{s}$ to denote the vector $s$ with an appended 1.

We are now ready to prove Theorem 3.

**Proof of Theorem 3** For a fixed data point $(x, y)$, define the matrix-valued functions on $\theta = [\tilde{W}_1, \tilde{W}_2]$:

1. $G_1 : \theta \mapsto [\sigma(\tilde{W}_1 \tilde{x})^\top, 1] \in \mathbb{R}^{1 \times (h+1)}$
2. $G_2 : \theta \mapsto \tilde{W}_2^\top \in \mathbb{R}^{(h+1) \times n}$
3. $G_3 : \theta \mapsto \tilde{W}_2 \in \mathbb{R}^{n \times h}$
4. $G_4 : \theta \mapsto \text{diag}(\sigma'(\tilde{W}_1 \tilde{x})) \in \mathbb{R}^{h \times h}$

Then

$$f_1 := \frac{\partial \ell(\theta)}{\partial \tilde{W}_1} = \tilde{x}*(G_1*G_2 - y^\top)*G_3*G_4(\theta) \quad \text{and}$$
\[
f_2 := \frac{\partial \ell(\theta)}{\partial \hat{W}_2} = G_1^\top (G_1 \ast G_2 - y^\top)(\theta).
\]

We compute Lipschitz constants and upper bounds of \(G_i\) (see Definition 5) for \(i = 1, 2, 3, 4\):

\[
M_{G_1} \leq \sqrt{h+1}, \quad M_{G_2} \leq C_1 \sqrt{n(h+1)}, \quad M_{G_3} \leq C_1 \sqrt{n(h+1)}, \quad M_{G_4} \leq \sqrt{h+1},
\]

\[
L_{G_1} = C_\sigma \|\hat{x}\|_2, \quad L_{G_2} = L_{G_3} = 1, \quad L_{G_4} = C_\sigma \|\hat{x}\|_2.
\]

Using Corollary 1, we obtain:

\[
L_{f_1} = 2C_1 \sqrt{n}(h+1)^{3/2} \left(1 + C_\sigma C_1 \sqrt{n}\|\hat{x}\|_2\right) \|\hat{x}\|_2
\]

\[
L_{f_2} = (h+1) \left(2C_\sigma C_1 \sqrt{n}\|\hat{x}\|_2 + 1\right) \leq 2(h+1) \left(1 + C_\sigma C_1 \sqrt{n}\|\hat{x}\|_2\right).
\]

Choose \(L_{f_2} = 2(h+1) \left(C_\sigma C_1 \sqrt{n}\|\hat{x}\|_2 + 1\right).\) Therefore, a Lipschitz constant for \(\nabla_\theta \ell\) is

\[
L_{\nabla_\theta \ell} = \sqrt{(L_{f_1})^2 + (L_{f_2})^2} = 2(h+1) \left(1 + C_\sigma C_1 \sqrt{n}\|\hat{x}\|_2\right) \sqrt{C_1^2 n(h+1)\|\hat{x}\|_2^2 + 1}
\]

\[
= O\left(n C_\sigma C_1^2 h^{3/2}\|\hat{x}\|_2^2\right),
\]

which completes the proof. \(\square\)

References

1. Adcock, B., Brugiapaglia, S., Webster, C.G.: Compressed sensing approaches for polynomial approximation of high-dimensional functions. In: Compressed Sensing and its Applications, pp. 93–124. Springer, Cham (2017)
2. Barron, A.R.: Neural net approximation. In: Proc. 7th Yale Workshop on Adaptive and Learning Systems, vol. 1, pp. 69–72 (1992)
3. Bellot, A., Branson, K., van der Schaar, M.: Neural graphical modelling in continuous-time: consistency guarantees and algorithms. In: International Conference on Learning Representations (2021)
4. Brunton, S.L., Proctor, J.L., Kutz, J.N.: Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proc. Natl. Acad. Sci. 113(15), 3932–3937 (2016)
5. Bregman, L.M.: The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming. USSR Comput. Math. Math. Phys. 7(3), 200–217 (1967)
6. Bungert, L., Roith, T., Tenbrinck, D., Burger, M.: A Bregman learning framework for sparse neural networks. arXiv preprint arXiv:2105.04319, 2021
7. Constantine, P.G., Diaz, P.: Global sensitivity metrics from active subspaces. Reliab. Eng. Syst. Safety 162, 1–13 (2017)
8. Chkifa, A., Dexter, N., Tran, H., Webster, C.: Polynomial approximation via compressed sensing of high-dimensional functions on lower sets. Math. Comput. 87(311), 1415–1450 (2018)
9. Constantine, P.G., Dow, E., Wang, Q.: Active subspace methods in theory and practice: applications to kriging surfaces. SIAM J. Sci. Comput. 36(4), A1500–A1524 (2014)
10. Viet Cuong, N., Tung Ho, L.S., Dinh, V.: Generalization and robustness of batched weighted average algorithm with V-geometrically ergodic Markov data. In: International Conference on Algorithmic Learning Theory, pp. 264–278. Springer (2013)
11. Constantine, P.G., Kent, C., Bui-Thanh, T.: Accelerating Markov chain Monte Carlo with active subspaces. SIAM J. Sci. Comput. 38(5), A2779–A2805 (2016)
12. Cohen, N., Sharir, O., Shashua, A.: On the expressive power of deep learning: a tensor analysis. In: Conference on Learning Theory, pp. 698–728. PMLR (2016)
13. Chen, Z., Kailiang, W., Xiu, D.: Methods to recover unknown processes in partial differential equations using data. J. Sci. Comput. 85(2), 1–23 (2020)
14. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control, Signals Syst. 2(4), 303–314 (1989)
15. Dinh, V., Tung Ho, L.S.: Consistent feature selection for neural networks via adaptive group lasso. arXiv preprint arXiv:2006.00334 (2020)
16. Dinh, V.C., Tung Ho, L.S.: Consistent feature selection for analytic deep neural networks. Adv. Neural Inf. Process. Syst. 33, 2420–2431 (2020)
17. Dinh, V., Tung Ho, L.S., Viet Cuong, N., Nguyen, D., Nguyen, B.T.: Learning from non-iid data: fast rates for the one-vs-all multiclass plug-in classifiers. In: International Conference on Theory and Applications of Models of Computation, pp. 375–387. Springer (2015)
18. Eldan, R., Shamir, O.: The power of depth for feedforward neural networks. In: Conference on Learning Theory, pp. 907–940. PMLR (2016)
19. Ho, L.S.T., Dinh, V.: Searching for minimal optimal neural networks. Stat. Probab. Lett. 183, 109353 (2022)
20. Hsieh, C.-J., Olsen, P.: Nuclear norm minimization via active subspace selection. In: International Conference on Machine Learning, pp. 575–583. PMLR (2014)
21. Hang, H., Steinwart, I.: Fast learning from α-mixing observations. J. Multivar. Anal. 127, 184–199 (2014)
22. Ho, L.S.T., Schaeffer, H., Tran, G., Ward, R.: Recovery guarantees for polynomial coefficients from weakly dependent data with outliers. J. Approx. Theory 259, 105472 (2020)
23. Kingma, D.P., Adam, J.B.: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
24. Kumpati, S.N., Kannan, P., et al.: Identification and control of dynamical systems using neural networks. IEEE Trans. Neural Netw. 1(1), 4–27 (1990)
25. Kang, S.H., Liao, W., Liu, Y.: Identifying differential equations with numerical time evolution. J. Sci. Comput. 87(1), 1–27 (2021)
26. Ljung, L.: System identification. Springer (1998)
27. Lorenz, E.N.: Predictability: a problem partly solved. In: Proc. Seminar on Predictability, vol. 1 (1996)
28. Li, S., Yang, Y.: Hierarchical deep learning for data-driven identification of reduced-order models of nonlinear dynamical systems. Nonlinear Dyn. 105(4), 3409–3422 (2021)
29. Fei, L., Zhong, M., Tang, S., Maggioni, M.: Nonparametric inference of interaction laws in systems of agents from trajectory data. Proc. Natl. Acad. Sci. 116(29), 14424–14433 (2019)
30. Murray, K., Chiang, D.: Auto-sizing neural networks: with applications to n-gram language models. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 908–916 (2015)
31. Murray, K., Kinnison, J., Nguyen, T.Q., Scheirer, W., Chiang, D.: Auto-sizing the transformer network: improving speed, efficiency, and performance for low-resource machine translation. In: Proceedings of the Third Workshop on Neural Generation and Translation (2019)
32. Parikh, N., Boyd, S.: Proximal algorithms. Found. Trends Optim. 1(3), 127–239 (2014)
33. Petzold, L.: Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. SIAM J. Sci. Stat. Comput. 4(1), 136–148 (1983)
34. Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., Liao, Q.: Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. Int. J. Autom. Comput. 14(5), 503–519 (2017)
35. Quarranta, G., Lacarbonara, W., Masri, S.F.: A review on computational intelligence for identification of nonlinear dynamical systems. Nonlinear Dyn. 99(2), 1709–1761 (2020)
36. Qin, T., Kailiang, W., Xiu, D.: Data driven governing equations approximation using deep neural networks. J. Comput. Phys. 395, 620–635 (2019)
37. Raissi, M., Karniadakis, G.E.: Hidden physics models machine learning of nonlinear partial differential equations. J. Comput. Phys. 357, 125–141 (2018)
38. Rauhut, H., Ward, R.: Sparse Legendre expansions via ℓ1-minimization. J. Approx. Theory 164(5), 517–533 (2012)
40. Scardapane, S., Comminiello, D., Hussain, A., Uncini, A.: Group sparse regularization for deep neural networks. Neurocomputing 241, 81–89 (2017)
41. Steinwart, I., Hush, D., Scovel, C.: Learning from dependent observations. J. Multivar. Anal. 100(1), 175–194 (2009)
42. Schaeffer, H., Tran, G., Ward, R.: Extracting sparse high-dimensional dynamics from limited data. SIAM J. Appl. Math. 78(6), 3279–3295 (2018)
43. Sun, Y., Zhang, L., Schaeffer, H.: Neupde: neural network based ordinary and partial differential equations for modeling time-dependent data. In: Mathematical and Scientific Machine Learning, pp. 352–372. PMLR (2020)
44. Telgarsky, M.: Benefits of depth in neural networks. In: Conference on Learning Theory, pp. 1517–1539. PMLR (2016)
45. Tran, G., Ward, R.: Exact recovery of chaotic systems from highly corrupted data. Multiscale Model. Simul. 15(3), 1108–1129 (2017)
46. Wit, E., van den Heuvel, E., Romeijn, J.-W.: All models are wrong: an introduction to model uncertainty. Stat. Neerlandica 66(3), 217–236 (2012)
47. Wang, R., Kalnay, E., Balachandran, B.: Neural machine-based forecasting of chaotic dynamics. Nonlinear Dyn. 98(4), 2903–2917 (2019)
48. Wong, K.C., Li, Z., Tewari, A.: Lasso guarantees for $\beta$-mixing heavy-tailed time series. Ann. Stat. 48(2), 1124–1142 (2020)
49. Yun, J., Lozano, A.C., Yang, E.: A general family of stochastic proximal gradient methods for deep learning. arXiv preprint arXiv:2007.07484 (2020)
50. Zhang, H., Wang, J., Sun, Z., Zurada, J.M., Pal, N.R.: Feature selection for neural networks using group lasso regularization. IEEE Trans. Knowl. Data Eng. 32(4), 659–673 (2019)