國立陽明交通大學
統計學研究所
碩士論文

Department of Statistics
National Yang Ming Chiao Tung University
Master Thesis

結合捲積層及自編碼器實現空間稀疏主成分分析

Spatial Sparse Principal Component Analysis Using Autoencoders with Convolutions

研究生 : 林宣廷 (Shiuan-Ting Lin)

指導教授 : 黃信誠 (Hsin-Cheng Huang)

洪慧念 (Hui-Nien Hung)

中華民國 一一三年七月
July 2024

結合捲積層及自編碼器實現空間稀疏主成分分析

Spatial Sparse Principal Component Analysis Using Autoencoders with Convolutions

研究生：林宣廷
指導教授：黃信誠
洪慧念

Student: Shiuan-Ting Lin

Advisor: Dr. Hsin-Cheng Huang
Dr. Hui-Nien Hung

國立陽明交通大學
統計學研究所
碩士論文

A Thesis
Submitted to Department of Statistics
College of Science
National Yang Ming Chiao Tung University
in partial Fulfilment of the Requirements
for the Degree of
Master
in
Statistics

July 2024

Taiwan, Republic of China

中華民國 一一三年七月

Acknowledgement

首先，我必須衷心感謝黃信誠教授對我的協助與指導。教授的鼓勵與指教我銘記在心，對於論文內容的深入討論與無私的幫助，我感激不盡，我有時雖然在會議時不上心，或者有時提出許多天馬行空的想法，但教授仍耐心的指引我正確的方向，使得我能夠順利的完成我的論文，若無教授細心及耐心的協助，我將無法將論文順利的完成。我也在與教授交流的過程中學習到非常多，將來我也會更努力增進對於細節的追求！

同時，我也非常感謝我的共同指導教授洪慧念教授，教授的建議與支持幫助我順利克服了研究路上的許多難關，對此我深感感激，在會議時總是爲我指出了關鍵的問題癥結點，使得我能夠順利地克服重重難關。

我也要特別感謝我的家人，在我求學的道路上總是無條件地支持我，鼓勵我追尋心中的理想，並尊重我的每一個決定，在碩士班這段時間，我雖不常回家，但每次與父母家人的對話都讓我獲益良多，並讓我感受到滿滿的關心與照顧。

此外，我無法不提及陪伴我十年的女朋友艷娟，她在我迷茫和困難時刻總是給予我最大的支持和激勵，讓我有勇氣面對一切挑戰。感謝她的陪伴，使我的學習旅程充滿了希望和力量，我們雖經歷了起起伏伏，但未來將一起發光發熱！

而我也想要感謝在我學習路上與我一同切磋學習的好朋友們：阿文、小玉、斗哥、小胖、Boris、Yoyo、冠憲、奕勳、長駿、汽油、豪豪，你們的陪伴讓我的碩士生活充滿了色彩，期許我們未來都能夠活出精采的人生！

而我也要感謝所有在這段學術旅程中幫助過我的所有人，是你們充實了我的碩士生活以及研究之路，祝福各位未來前程似錦，事事順心。

結合捲積層及自編碼器實現空間稀疏主成分分析

學生：林宣廷

指導教授：黃信誠 博士
洪慧念 博士

國立陽明交通大學 統計學研究所

摘　要

本研究提出了一種結合捲積神經網路 (Convolutional neural network) 和自編碼器 (Autoencoder) 的架構，以實現保有空間結構的稀疏主成分分析。我們利用自編碼器找出最能解釋空間變異性的主成分，同時透過捲積神經網路捕捉區域性的空間特徵 (Spatial pattern)。此外，我們在損失函數中加入稀疏懲罰項以促進主成分的解釋性，透過模擬實驗，我們呈現了所提方法比傳統主成分分析和稀疏主成分分析等常見方法的優越之處。

關鍵詞: 典型相關分析、類神經網路、非線性主成分分析、正則化

Spatial Sparse Principal Component Analysis Using Autoencoders with Convolutions

Student: Shiuan-Ting Lin

Advisor: Dr. Hsin-Cheng Huang
Dr. Hui-Nien Hung

Department of Statistics
National Yang Ming Chiao Tung University

Abstract

This study proposes an innovative architecture that combines Convolutional Neural Networks (CNN) and Autoencoders to achieve sparse principal component analysis while preserving spatial structures. We utilize the autoencoder to discover the principal components that best explain spatial variability while simultaneously capturing local spatial patterns through CNN. Additionally, we incorporate a sparse penalty term in the loss function to promote the interpretability of the principal components. Through simulation experiments, we demonstrate that the proposed approach offer significant advantage over traditional methods, including conventional principal component analysis and sparse principal component analysis.

Keyword: Canonical correlation analysis, Neural network, Nonlinear principal component analysis, Regularization

Contents

List of Figures

Chapter 1

Introduction

Principal Component Analysis (PCA) (Jolliffe [2005]) is a widely employed method for dimensionality reduction, finding applications in diverse fields such as atmospheric science (Wilks [2011]) and genetics (Agrawal et al. [2020]). Its primary purpose is to reveal underlying patterns in the data. In the context of PCA, these patterns correspond to loading vectors, where each loading vector represents the contribution of the original variables to a principal component.

To enhance the interpretability of principal components, incorporating an ℓ_1 penalty on the loading vectors in the loss function provides a solution. This ℓ_1 penalty encourages sparse patterns (Tibshirani [1996]), where most elements of the loading vectors become zero. This approach, known as Sparse Principal Component Analysis (SPCA) (Zou et al. [2006]), enhances interpretability through sparsity of the loading vectors. However, PCA and SPCA do not adequately address problems involving patterns with spatial structures.

In machine learning, the fusion of PCA with neural networks has led to the development of advanced dimensionality reduction techniques. One such approach involves the use of autoencoders (Zhai et al. [2018]), a type of neural network designed to reduce dimensionality while preserving essential data features. Autoencoders operate by encoding the input data into a compressed representation and then decoding it back to the original format, ensuring minimal information loss. This process is conceptually similar to PCA but offer greater flexibility and capacity to model complex, non-linear relationships.

Moreover, Convolutional Neural Networks (CNNs) (Gu et al. [2018]) are specifically designed to handle data with spatial structures, such as images. CNNs are capable of capturing local spatial patterns through their convolutional layers, making them particularly effective in preserving spatial dependencies in the data. By intergrating CNNs with autoencoders, we can achieve a model that not only reduces dimensionality but also retains spatial information.

By intergrating the ideas of autoencoders with the spatial awareness of CNNs, we develop a powerful method for spatial PCA. Our model, named Sparse AutoEncoder with Convolutions (SAE-C), adds an ℓ_1 penalty to the loss function. This addition encourages sparsity in the learned patterns, similar to the concept of SPCA. So, the SAE-C model not only preserves spatial structures using the CNN architecture but also enhances interpretability by inducing sparsity in the learned patterns.

Chapter 2

Principal Component Analysis

2.1 Principal Component Analysis

Principal Component Analysis (PCA) (Jolliffe [2005]) is a widely employed method for reducing the dimensionality of data while retaining as much variability as possible. The goal of PCA is to identify the principal components, which are linear combinations of the original variables that capture the maximum variance in the data.

Consider an input matrix, $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)' \in \mathbb{R}^{n \times p}$, where n is the number of observations, and p is the number of variables. The sample covariance matrix $\boldsymbol{S}_X$ for the data matrix $\boldsymbol{X}$ is given by:

$$\boldsymbol{S}_X = \frac{1}{n-1}(\boldsymbol{X} - \mathbf{1}_n \bar{\boldsymbol{x}}')'(\boldsymbol{X} - \mathbf{1}_n \bar{\boldsymbol{x}}'), \tag{2.1}$$

where

$$\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i \tag{2.2}$$

is the sample mean of the data and $\mathbf{1}_n$ is a vector of ones with dimension n. PCA involves performing an eigen-decomposition of the sample covariance matrix:

$$\boldsymbol{S}_X = \hat{\boldsymbol{V}} \hat{\boldsymbol{\Lambda}}_X \hat{\boldsymbol{V}}' = (\hat{\boldsymbol{v}}_1, \ldots, \hat{\boldsymbol{v}}_p)\mathrm{diag}(\hat{\lambda}_1, \ldots, \hat{\lambda}_p)(\hat{\boldsymbol{v}}_1, \ldots, \hat{\boldsymbol{v}}_p)',$$

where $\hat{\lambda}_1 \geq \hat{\lambda}_2 \geq \cdots \geq \hat{\lambda}_p$ are the eigenvalues, and $\hat{\boldsymbol{v}}_i \in \mathbb{R}^p$ for $i = 1, \ldots, p$ are the corresponding eigenvectors. The matrix of the first K eigenvectors,

$$\hat{\boldsymbol{V}}_K = (\hat{\boldsymbol{v}}_1, \ldots, \hat{\boldsymbol{v}}_K) \in \mathbb{R}^{p \times K},$$

solves the minimization problem:

$$\hat{\boldsymbol{V}}_K = \underset{\boldsymbol{V}_K}{\text{argmin}} \|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{V}_K\boldsymbol{V}_K'\|_F, \text{ subject to } \boldsymbol{V}_K'\boldsymbol{V}_K = \boldsymbol{I}_K, \tag{2.3}$$

where $\boldsymbol{V}_K \in \mathbb{R}^{p \times K}$, $\| \cdot \|_F$ denotes the Frobenius norm, and $\boldsymbol{I}_K \in \mathbb{R}^{K \times K}$ is the identity matrix.

In PCA, the eigenvectors of the sample covariance matrix are referred to as loading vectors, representing the directions of maximum variance in the data. The principal component scores are the projections of the data onto these loading vectors. The sample covariance matrix of the principal component scores is diagonal, indicating that the principal components are uncorrelated. This diagonal structure, along with the orthonormality of the loading vectors, ensures that each principal component captures a unique aspect of the data's variability.

2.2 Sparse Principal Component Analysis

While PCA offers several desirable properties, including the orthogonality of loading vectors and uncorrelated principal component scores, interpreting these scores can be challenging especially in high-dimensional data contexts. In such context, the loading vectors often involve many variables, making it difficult to discerm which variables are most influential. This challenge is further compounded when the data exhibit spatial structures, as traditional PCA does not account for spatial dependencies and patterns. To address this issues and enhance interpretability, an ℓ_1 penalty is introduced on the loading vector estimation in the loss function to induce sparsity, promoting zero entries in the loading vectors.

Zou et al. [2006] proposed a modification to the PCA loss function to encourage sparsity in the loading vectors. The loading matrix for SPCA, $\hat{\boldsymbol{V}}_K^{(\text{s})} = (\hat{\boldsymbol{v}}_1^{(\text{s})}, \ldots, \hat{\boldsymbol{v}}_K^{(\text{s})}) \in \mathbb{R}^{p \times K}$, where $\hat{\boldsymbol{v}}_k^{(\text{s})} \in \mathbb{R}^p$ for $k = 1, \ldots, K$, consists of the estimated sparse loading vectors. The corresponding coefficient matrix $\hat{\boldsymbol{A}} \in \mathbb{R}^{p \times K}$ is obtained by solving the following optimization problem:

$$(\hat{\boldsymbol{V}}_K^{(\mathrm{s})}, \hat{\boldsymbol{A}}) = \underset{(\boldsymbol{V}_K, \boldsymbol{A})}{\operatorname{argmin}} \left(\|\boldsymbol{X} - \boldsymbol{X}\boldsymbol{V}_K\boldsymbol{A}'\|_F + \tau_0 \sum_{k=1}^{K} \|\boldsymbol{v}_k\|_2^2 + \tau_1 \sum_{k=1}^{K} \|\boldsymbol{v}_k\|_1 \right),$$
$$\text{subject to } \boldsymbol{A}'\boldsymbol{A} = \boldsymbol{I}_K, \qquad (2.4)$$

where $\boldsymbol{V}_K = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_K) \in \mathbb{R}^{p \times K}$, $\boldsymbol{A} \in \mathbb{R}^{p \times K}$, $\| \cdot \|_2$ is the ℓ_2 norm, $\| \cdot \|_1$ is the ℓ_1 norm, τ_0 is the ℓ_2 penalty parameter required to obtain exact PCA when the sparsity constraint vanishes ($\tau_1 = 0$), shown in Zou et al. [2006], and τ_1 is the parameter that control the degree of sparsity.

Due to the introduction of the ℓ_1 penalty term, the resulting $\hat{\boldsymbol{v}}_k^{(\mathrm{s})}$'s, which are the estimated loading vectors, become sparse, thereby enhancing interpretability.

2.3 Principal Component Analysis using Autoencoders

2.3.1 Autoencoders

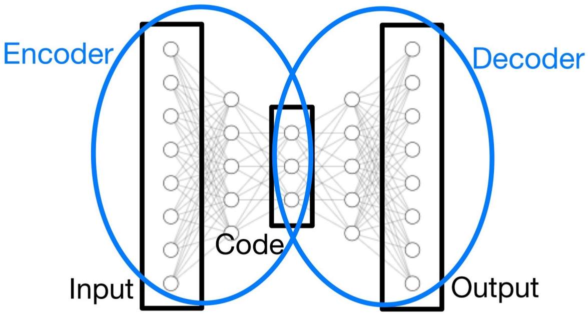


Figure 2.1: Illustration of an autoencoder.

Autoencoders are a type of artificial neural network (ANN) optimized for dimensionality reduction. They consist of two primary components: the encoder and the decoder. The encoder compresses the input data into a lower-dimensional latent space (codes), while the decoder attempts to reconstruct the input data from these codes. This process is illustrated in Figure 2.1.

In an L-layer ANN, each layer is defined by a function $f_l : \mathbb{R}^p \to \mathbb{R}^{p'}$. The function at the

l-th layer can be expressed as:

$$f_l(\boldsymbol{x}_i|\boldsymbol{\theta}) = \alpha_l(\boldsymbol{W}_l\boldsymbol{x}_i + \boldsymbol{b}_l);\ l = 1, \ldots, L, \tag{2.5}$$

where $\boldsymbol{\theta} = \{\boldsymbol{W}_l, \boldsymbol{b}_l\}$, $\boldsymbol{W}_l \in \mathbb{R}^{p' \times p}$ is the weight matrix, and $\boldsymbol{b}_l \in \mathbb{R}^{p'}$ is the bias vector. The function α_l is an activation function, such as Rectified Linear Unit (ReLU), Leaky ReLU, or Identity. The ReLU activation function, ReLU : $\mathbb{R} \to \mathbb{R}$, is defined as:

$$\text{ReLU}(x) = \text{max}(0, x), \tag{2.6}$$

the Leaky ReLU activation function, LeakyReLU : $\mathbb{R} \to \mathbb{R}$, is defined as:

$$\text{LeakyReLU}(x) = \begin{cases} x; & \text{if } x > 0, \\ ax; & \text{if } x \leq 0, \end{cases}$$

where a is a small constant (e.g., 0.01), and the Identity function, Identity : $\mathbb{R} \to \mathbb{R}$, is defined as:

$$\text{Identity}(x) = x, \tag{2.7}$$

and is used when preserving linearity is necessary. These activation functions are applied to each coordinate. For instance, $\alpha_l(\boldsymbol{x}_i) = (\text{ReLU}(x_{i1}), \ldots, \text{ReLU}(x_{ip}))'$, where $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})'$.

The essential components of an autoencoder are outlined as follows:

- **Encoder:** It's an L-layer ANN represented by $f_{\text{E}} = f_L \circ f_{L-1} \circ \cdots \circ f_1 : \mathbb{R}^p \to \mathbb{R}^d$ parametrized by $\boldsymbol{\theta}_{\text{E}} = \{\boldsymbol{W}_1, ..., \boldsymbol{W}_L, \boldsymbol{b}_1, ..., \boldsymbol{b}_L\}$ including the weight matrices and bias vectors for the encoder. $f_{\text{E}}(\boldsymbol{x}|\boldsymbol{\theta}_{\text{E}}) = \boldsymbol{z}$ maps an input $\boldsymbol{x} \in \mathbb{R}^p$ to a code $\boldsymbol{z} \in \mathbb{R}^d$ ($d < p$).

- **Decoder:** Functioning inversely to the encoder, this is an L'-layer ANN represented by $f_{\text{D}} = f_{L+L'} \circ f_{L+L'-1} \circ \cdots \circ f_{L+1} : \mathbb{R}^d \to \mathbb{R}^p$ parametrized by $\boldsymbol{\theta}_{\text{D}} = \{\boldsymbol{W}_{L+1}, ..., \boldsymbol{W}_{L+L'}, \boldsymbol{b}_{L+1}, ..., \boldsymbol{b}_{L+L'}\}$ including the weight matrices and bias vectors for the decoder. $f_{\text{D}}(\boldsymbol{z}|\boldsymbol{\theta}_{\text{D}}) = \hat{\boldsymbol{x}}$

maps the code back to the reconstructed input space.

- **Autoencoder:** The combination of the encoder and decoder, represented by $f_{\mathrm{AE}} = f_{\mathrm{D}} \circ f_{\mathrm{E}} : \mathbb{R}^p \to \mathbb{R}^p$, aims to minimize the loss between the outputs of f_{AE} and the original data. The estimated parameters $\hat{\boldsymbol{\theta}}_{\mathrm{E}}$ and $\hat{\boldsymbol{\theta}}_{\mathrm{E}}$ satisfy the following optimization problem:

$$(\hat{\boldsymbol{\theta}}_{\mathrm{E}}, \hat{\boldsymbol{\theta}}_{\mathrm{D}}) = \underset{(\boldsymbol{\theta}_{\mathrm{E}}, \boldsymbol{\theta}_{\mathrm{D}})}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - f_{\mathrm{AE}}(\boldsymbol{x}_i | \boldsymbol{\theta}_{\mathrm{E}}, \boldsymbol{\theta}_{\mathrm{D}})\|_2^2, \tag{2.8}$$

  where $\boldsymbol{\theta}_{\mathrm{E}}$ and $\boldsymbol{\theta}_{\mathrm{D}}$ represent the weight matrices and bias vectors for the encoder and the decoder, respectively. This mean squared error function (2.8), known as the reconstruction loss (Michelucci [2022]), is minimized using gradient descent techniques (Chong and Żak [2013]), such as stochastic gradient descent (SGD), RMSProp, and adaptive momentum (Adam).

2.4 Connection between Autoencoders and PCA

The code, $f_{\mathrm{E}}(\boldsymbol{x}|\hat{\boldsymbol{\theta}}_{\mathrm{E}}) = \boldsymbol{z} \in \mathbb{R}^d$, generated by the encoder is a lower-dimensional version of the input data $\boldsymbol{x} \in \mathbb{R}^p$. This compression is analogous to the dimensionality reduction achieved through PCA, where the data is projected onto a set of orthogonal directions that capture the most variance.

The connection between autoencoders and PCA becomes evident when analyzing their objectives and transformations under simplifying assumptions that the encoder and decoder are only single-layer ANNs with the identity activation function with zero bias vectors. This autoencoder, composed of fully connected layers, is represented as:

$$f_{\mathrm{AE}}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathrm{E}}, \boldsymbol{\theta}_{\mathrm{D}}) = \boldsymbol{W}_2(\boldsymbol{\theta}_{\mathrm{D}}) \boldsymbol{W}_1(\boldsymbol{\theta}_{\mathrm{E}})' \boldsymbol{x},$$

where $\boldsymbol{W}_1(\boldsymbol{\theta}_{\mathrm{E}}), \boldsymbol{W}_2(\boldsymbol{\theta}_{\mathrm{D}}) \in \mathbb{R}^{p \times K}$, $\boldsymbol{\theta}_{\mathrm{E}} = \{\boldsymbol{W}_1\}$ and $\boldsymbol{\theta}_{\mathrm{D}} = \{\boldsymbol{W}_2\}$. The estimated matrices $\hat{\boldsymbol{W}}_1$ and $\hat{\boldsymbol{W}}_2$ solve the optimization problem (2.8).

The simplified autoencoder task connects to PCA by the uniqueness of the projection matrix that project the inputs onto the column space of $\hat{\boldsymbol{V}}_K$. The projection matrix for these first K principal component loadings can be written as $\hat{\boldsymbol{V}}_K\hat{\boldsymbol{V}}_K' \in \mathbb{R}^{p\times p}$. Therefore, the reconstruction loss for the PCA reconstruction, $\boldsymbol{X}\hat{\boldsymbol{V}}_K\hat{\boldsymbol{V}}_K'$, and the autoencoder reconstruction, $\boldsymbol{X}\hat{\boldsymbol{W}}_1\hat{\boldsymbol{W}}_2'$, can be connected as follows:

$$\|\boldsymbol{X} - \boldsymbol{X}\hat{\boldsymbol{V}}_K\hat{\boldsymbol{V}}_K'\|_F = \|\boldsymbol{X} - \boldsymbol{X}\hat{\boldsymbol{W}}_1\hat{\boldsymbol{W}}_2'\|_F,$$

where $\hat{\boldsymbol{V}}_K\hat{\boldsymbol{V}}_K'$ is a rank-K $p \times p$ matrix projecting the inputs onto a K dimensional subspace. Thus, we conclude that $\hat{\boldsymbol{V}}_K\hat{\boldsymbol{V}}_K' = \hat{\boldsymbol{W}}_1\hat{\boldsymbol{W}}_2'$ but $\hat{\boldsymbol{V}}_K$ is not necessarily equal to $\hat{\boldsymbol{W}}_1$.

This equivalence illustrates how autoencoders, under specific conditions, perform a task similar to PCA by projecting the data into a lower-dimensional space that captures the most significant variance, thereby bridging the methodologies of autoencoders and PCA.

Chapter 3

Spatial PCA Methods

In this chapter, we incorporate convolutional layers into the structure of autoencoders. This addition enables the codes to retain spatial information. Additionally, we introduce an ℓ_1 penalty in the loss function to induce sparsity, thereby enhancing the interpretability of the loading vectors.

3.1 Sparse PCA using Autoencoders

Inspired by the loss function used in SPCA (2.4), we propose an autoencoder architecture to induce sparsity in the encoder weights. This model features a one-layer encoder and a one-layer decoder, both with zero bias vectors and employing identity activation functions, that means the trainable parameters of encoder includes only one weight matrix, $\boldsymbol{\theta}_{\text{E}} = \{\boldsymbol{W}_1\}$, so does decoder, $\boldsymbol{\theta}_{\text{D}} = \{\boldsymbol{W}_2\}$. This implies that the activation functions for both the encoder and the decoder are all identity transformations (2.7) which maintain linearity throughout the model. The encoder $f_{\text{E}} : \mathbb{R}^p \to \mathbb{R}^K$ parametrized by $\boldsymbol{\theta}_{\text{E}}$ and decoder $f_{\text{D}} : \mathbb{R}^K \to \mathbb{R}^p$ parametrized by $\boldsymbol{\theta}_{\text{D}}$ are defined as follows:

$$\begin{aligned} f_{\text{E}}(\boldsymbol{x}|\boldsymbol{\theta}_{\text{E}}) =& \boldsymbol{W}_1(\boldsymbol{\theta}_{\text{E}})'\boldsymbol{x} = \boldsymbol{z}, \\ f_{\text{D}}(\boldsymbol{z}|\boldsymbol{\theta}_{\text{D}}) =& \boldsymbol{W}_2(\boldsymbol{\theta}_{\text{D}})\boldsymbol{z} = \hat{\boldsymbol{x}}. \end{aligned}$$

To promote weight sparsity in the autoencoder's encoder, we incorporate an ℓ_1 penalty term into the loss function. Additionally, we impose constraints on the decoder weights. The optimization problem, aimed to be minimized with respect to the input data points $\boldsymbol{x}_i \in \mathbb{R}^p, i =$

$1, ..., n$, is formulated as follows:

$$\begin{aligned}(\hat{\boldsymbol{V}}_K^{(\mathrm{SE})}, \hat{\boldsymbol{W}}_2) &= \underset{\boldsymbol{\theta}_{\mathrm{E}},\boldsymbol{\theta}_{\mathrm{D}}}{\operatorname{argmin}} \left(\sum_{i=1}^{n} \|\boldsymbol{x}_i - f_{\mathrm{D}} \circ f_{\mathrm{E}}(\boldsymbol{x}_i|\boldsymbol{\theta}_{\mathrm{E}}, \boldsymbol{\theta}_{\mathrm{D}})\|_2^2 + \tau_2 \sum_{k=1}^{K} \|\boldsymbol{w}_{1,k}(\boldsymbol{\theta}_{\mathrm{E}})\|_1 \right) \\ &= \underset{(\boldsymbol{W}_1(\boldsymbol{\theta}_{\mathrm{E}}),\boldsymbol{W}_2(\boldsymbol{\theta}_{\mathrm{D}}))}{\operatorname{argmin}} \left(\sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{W}_2(\boldsymbol{\theta}_{\mathrm{D}})\boldsymbol{W}_1(\boldsymbol{\theta}_{\mathrm{E}})'\boldsymbol{x}_i\|_2^2 + \tau_2 \sum_{k=1}^{K} \|\boldsymbol{w}_{1,k}(\boldsymbol{\theta}_{\mathrm{E}})\|_1 \right), \\ &\quad \text{subject to } \|\boldsymbol{w}_{2,k}(\boldsymbol{\theta}_{\mathrm{D}})\|_2 = 1;\ k = 1, \ldots, K, \end{aligned} \tag{3.1}$$

where $\boldsymbol{W}_1(\boldsymbol{\theta}_{\mathrm{E}}) = (\boldsymbol{w}_{1,1}(\boldsymbol{\theta}_{\mathrm{E}}), \ldots, \boldsymbol{w}_{1,K}(\boldsymbol{\theta}_{\mathrm{E}})) \in \mathbb{R}^{p \times K}$ and $\boldsymbol{W}_2(\boldsymbol{\theta}_{\mathrm{D}}) = (\boldsymbol{w}_{2,1}(\boldsymbol{\theta}_{\mathrm{D}}), \ldots, \boldsymbol{w}_{2,K}(\boldsymbol{\theta}_{\mathrm{D}}))$ $\in \mathbb{R}^{p \times K}$. The sparsity parameter τ_2 controls the degree of sparsity, and $\hat{\boldsymbol{V}}_K^{(\mathrm{SE})}(\tau_2)$ is the estimated sparse encoder representing the sparse loading matrix for the Sparse AutoEncoder (SAE) method with sparsity parameter τ_2.

The selection of the sparsity parameter τ_2 in (3.1) is crucial and can be determined using the validation data $\boldsymbol{X}_{\text{valid}}$ to minimize the reconstruction loss (validation MSE), as described by:

$$(\hat{\tau}_2, \hat{K}_2) = \underset{(\tau_2,K)\in \boldsymbol{S}_2}{\operatorname{argmin}} \|\boldsymbol{X}_{\text{valid}} - \boldsymbol{X}_{\text{valid}} \hat{\boldsymbol{V}}_K^{(\mathrm{SE})}(\tau_2)\hat{\boldsymbol{W}}_2'\|_F, \tag{3.2}$$

where $\boldsymbol{X}_{\text{valid}} \in \mathbb{R}^{n' \times p}$ represents the validation data with n' data, $\hat{\boldsymbol{V}}_K^{(\mathrm{SE})}(\tau_2)$ and $\hat{\boldsymbol{W}}_2$ are the estimated encoder and decoder matrices with sparsity parameter τ_2 from (3.1), and the candidate set S_2 is a predefined set that all the combinations of τ_2 and K that we wish to evaluate to find the optimal values that minimize the reconstruction loss. When selecting the optimal sparsity parameter τ_2 and the number of principal component K, the estimated sparse encoder representing the loading matrix for the SAE method is:

$$\hat{\boldsymbol{V}}_{\hat{K}_2}^{(\mathrm{SE})}(\hat{\tau}_2).$$

Training is performed using gradient descent as mentioned in Chapter 2.3.1, where the choice of initial values significantly affects the convergence rate. Starting with the smallest τ_2 from the candidate set S_2, the model is trained progressively with increasing τ_2, using weights from the previous training phases as initialization. Specifically, for $\tau_2 = 0$, $\boldsymbol{W}_1(\boldsymbol{\theta}_{\mathrm{E}})$ and $\boldsymbol{W}_2(\boldsymbol{\theta}_{\mathrm{D}})$ are initialized with the PCA loading matrix $\hat{\boldsymbol{V}}_K$ to ensure that the solution matches the PCA result when $\tau_2 = 0$. For subsequent training phases, the initial values for $\boldsymbol{W}_1(\boldsymbol{\theta}_{\mathrm{E}})$ and $\boldsymbol{W}_2(\boldsymbol{\theta}_{\mathrm{D}})$ are set to $\hat{\boldsymbol{V}}_K^{(\mathrm{SE})}$ and $\hat{\boldsymbol{W}}_2$ obtained from the previous τ_2. This progressive training approach not only speeds up convergence but also improves the model's performance by leveraging previously gained knowledge.

3.2 Introduction to Convolutional Layers in CNNs

Convolution is a fundamental operation in signal processing and is crucial for convolutional neural networks (CNNs) (Li et al. [2021]), especially when handling spatial data like images or spatially correlated signals. Convolution layers in CNNs utilize this operation to extract higher-level features by applying filters or kernels that capture specific spatial hierarchies. These filters move across the input data, creating feature maps that highlight various aspects of the data, such as edges, textures, and more complex patterns.

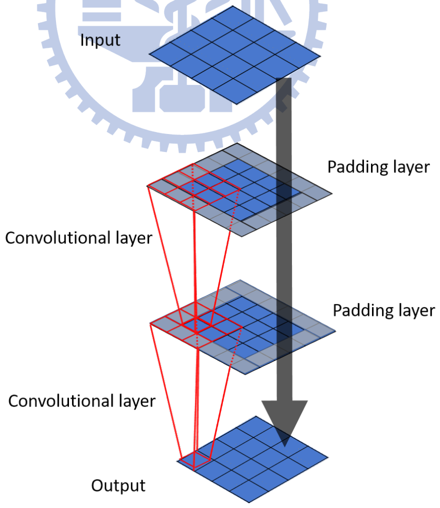


Figure 3.1: Illustration of layer functions including zero padding layers and convolutional layers. Grey indicates zero padding, and red indicates convolutional layers.

To ensure that the convolution operation does not result in out-of-bound indices, zero padding (Yamashita et al. [2018]) is applied to the input matrix $\boldsymbol{Z} \in \mathbb{R}^{p_1 \times p_2}$ before performing the convolutional operator with filter size $d_1 \times d_2$. This process extends the dimensions of the input matrix by adding zeros around the border, resulting in a padded input matrix $\tilde{\boldsymbol{Z}} \in \mathbb{R}^{(p_1+d_1-1)\times(p_2+d_2-1)}$.

Formally, if $\boldsymbol{Z} \in \mathbb{R}^{p_1 \times p_2}$ is a $p_1 \times p_2$ matrix, the padded input matrix,

$$\tilde{\boldsymbol{Z}} = \text{pad}(\boldsymbol{Z}) \in \mathbb{R}^{(p_1+d_1-1)\times(p_2+d_2-1)}$$

is defined as follows:

$$\tilde{z}_{i,j} = \begin{cases} z_{i-\lceil d_1/2\rceil+1, j-\lceil d_2/2\rceil+1}; & \text{if } \lceil d_1/2\rceil \leq i \leq p_1 - \lfloor d_1/2\rfloor \text{ and } \lceil d_2/2\rceil \leq j \leq p_2 - \lfloor d_2/2\rfloor, \\ 0; & \text{otherwise,} \end{cases}$$

where $z_{i,j}$ is the (i,j)-th element of $\boldsymbol{Z}$, $\tilde{z}_{i,j}$ is the (i,j)-th element of $\tilde{\boldsymbol{Z}}$, $\lfloor\cdot\rfloor$ denotes the floor division, and $\lceil\cdot\rceil$ denotes the ceiling division.

For a convolutional layer in a CNN, the convolutional operator, denoted by $*$, is applied to extract features from input data. As illustrated in Figure 3.1, the convolutional operator with a convolutional filter $\boldsymbol{C} \in \mathbb{R}^{d_1 \times d_2}$ is defined as follows:

$$(\boldsymbol{C} * \tilde{\boldsymbol{Z}})_{i,j} = \sum_{k=1}^{d_1} \sum_{l=1}^{d_2} c_{k,l} \tilde{z}_{i+k-1, j+l-1}; \; i = 1, \ldots, p_1, j = 1, \ldots, p_2,$$

where $c_{k,l}$ is the (k,l)-th element of $\boldsymbol{C}$, $\tilde{z}_{i,j}$ is the padded input matrix $\tilde{\boldsymbol{Z}}$ with the (i,j)-th element $\tilde{z}_{i,j}$, and $(\boldsymbol{C} * \tilde{\boldsymbol{Z}})_{i,j}$ is the (i,j)-th element of $\boldsymbol{C} * \tilde{\boldsymbol{Z}} \in \mathbb{R}^{p_1 \times p_2}$. This operator effectively slides the filter over the padded input matrix to compute the convolution and produce the output feature map.

The convolutional layer with zero padding is used in the proposed model. This layer is represented by the function $g_{\boldsymbol{C}} : \mathbb{R}^{p_1 \times p_2} \to \mathbb{R}^{p_1 \times p_2}$, defined as:

$$g_C(\boldsymbol{Z}) = \alpha(\boldsymbol{C} * \text{pad}(\boldsymbol{Z}) + \boldsymbol{B}), \tag{3.3}$$

where α is the activation function applied to each component in the matrix, $\boldsymbol{B} \in \mathbb{R}^{p_1 \times p_2}$ is the bias matrix, and $\text{pad}(\cdot)$ is the zero padding function.

3.3 Spatial Sparse PCA using Autoencoders

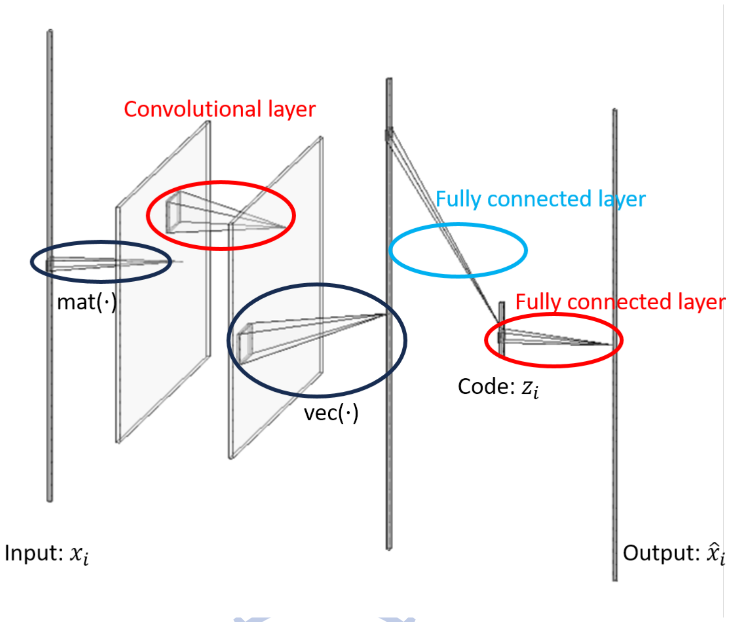


Figure 3.2: Model architecture of the proposed 2D autoencoder with convolutions for input vector $\boldsymbol{x}_i$ and output vector $\hat{\boldsymbol{x}}_i$. First, reshape the vector into a matrix using mat(·), then pass it through convolutional layers. Next, reshape the output back to a vector using vec(·) and pass it through a fully connected layer to reduce dimensionality. Finally, pass it through another fully connected layer to restore the original dimension. Red indicates ℓ_2 constraint on convolutional and fully connected layers; blue indicates ℓ_1 penalty on the fully connected layer.

The proposed spatial sparse autoencoder architecture that utilizes convolutional layers, referred to as Sparse AutoEncoders with Convolutions (SAE-C), illustrated in Figure 3.2. This architecture allows for the preservation of spatial patterns, which is inherent to the design of convolutional layers. A key innovation in our approach is the introduction of an ℓ_1 penalty to the encoder's fully connected layer to induce sparsity, along with ℓ_2 constraints on the convolutional filters of the encoder and the fully connected layer of the decoder, adapting the principles of SPCA to a spatial context.

If the inputs are vectors, a reshape layer is considered before passing to convolutional layers with zero padding. We assume the dimensionality p of the given inputs $\boldsymbol{x}_i$ to be a multiplication of two integers q and r, i.e., $p = qr$. The function mat(·) reshapes the given input $\boldsymbol{x}_i$ into a matrix $\boldsymbol{X}_i \in \mathbb{R}^{q \times r}$ as follows:

$$\mathrm{mat}(\boldsymbol{x}_i) = \boldsymbol{X}_i, \text{ where } X_{i,j,k} = x_{i,(k-1)q+j};\ j = 1, \ldots, q, k = 1, \ldots, r. \tag{3.4}$$

Here, $X_{i,j,k}$ is the (j, k)-th element of $\boldsymbol{X}_i$ and $x_{i,l}$ is the l-th element of $\boldsymbol{x}_i$. Conversely, we can flatten it back into $\boldsymbol{x}_i$. This transformation can be expressed as:

$$\mathrm{vec}(\boldsymbol{X}_i) = \boldsymbol{x}_i, \text{ where } x_{i,(k-1)q+j} = X_{i,j,k};\ j = 1, \ldots, q, k = 1, \ldots, r. \tag{3.5}$$

vec$(\cdot)$ is the function that flattens the $q \times r$ matrix $\boldsymbol{X}_i$ into a vector of length qr.

The proposed SAE-C consists of an encoder with M convolutional layers with zero bias matrices (3.3) followed with a fully connected layer (2.5) reducing the dimensionality to K and a decoder with one fully connected layer, both employing identity activation functions (2.7). This choice of activation functions ensures that the model maintains linearity, crucial for interpreting the PCA-like transformation effectively. The encoder $f_{\mathrm{E}} : \mathbb{R}^p \to \mathbb{R}^K$ parametrized by $\boldsymbol{\theta}_{\mathrm{E}} = \{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_M, \boldsymbol{W}_{M+1}\}$ and decoder $f_{\mathrm{D}} : \mathbb{R}^K \to \mathbb{R}^p$ parametrized by $\boldsymbol{\theta}_{\mathrm{D}} = \{\boldsymbol{W}_{M+2}\}$ are defined as follows:

$$\begin{aligned} f_{\mathrm{E}}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathrm{E}}) &= \boldsymbol{W}_{M+1}(\boldsymbol{\theta}_{\mathrm{E}})'\mathrm{vec}(g_{\boldsymbol{C}_M} \circ \cdots \circ g_{\boldsymbol{C}_2} \circ g_{\boldsymbol{C}_1}(\mathrm{mat}(\boldsymbol{x})|\boldsymbol{\theta}_{\mathrm{E}})) = \boldsymbol{z}, \\ f_{\mathrm{D}}(\boldsymbol{z}|\boldsymbol{\theta}_{\mathrm{D}}) &= \boldsymbol{W}_{M+2}(\boldsymbol{\theta}_{\mathrm{D}})\boldsymbol{z} = \hat{\boldsymbol{x}}, \end{aligned} \tag{3.6}$$

where $g_{\boldsymbol{C}_m}$'s are convolutional layers with zero padding (3.3) with filters $\boldsymbol{C}_m \in \mathbb{R}^{d_1 \times d_2}, m = 1, ..., M$, and weight matrices $\boldsymbol{W}_{M+1}(\boldsymbol{\theta}_{\mathrm{E}}), \boldsymbol{W}_{M+2}(\boldsymbol{\theta}_{\mathrm{D}}) \in \mathbb{R}^{p \times K}$.

Given data points $\boldsymbol{x}_i \in \mathbb{R}^p, i = 1, ..., n$, the optimization problem is defined as follows:

$$\begin{aligned}
(\hat{\boldsymbol{V}}_K^{(\text{CE})}, \hat{\boldsymbol{W}}_{M+2}) &= \underset{\boldsymbol{\theta}_\text{E}, \boldsymbol{\theta}_\text{D}}{\text{argmin}} \left(\sum_{i=1}^{n} \|\boldsymbol{x}_i - f_\text{D} \circ f_\text{E}(\boldsymbol{x}_i | \boldsymbol{\theta}_\text{E}, \boldsymbol{\theta}_\text{D})\|_2^2 + \tau_3 \sum_{k=1}^{K} \|\boldsymbol{w}_{M+1,k}(\boldsymbol{\theta}_\text{E})\|_1 \right) \\
&= \underset{(\boldsymbol{V}_K(\boldsymbol{\theta}_\text{E}), \boldsymbol{W}_{M+2}(\boldsymbol{\theta}_\text{D}))}{\text{argmin}} \left(\sum_{i=1}^{n} \|\boldsymbol{x}_i - \boldsymbol{W}_{M+2}(\boldsymbol{\theta}_\text{D}) \boldsymbol{V}_K(\boldsymbol{\theta}_\text{E})' \boldsymbol{x}_i\|_2^2 \right. \\
&\qquad \left. + \tau_3 \sum_{k=1}^{K} \|\boldsymbol{w}_{M+1,k}(\boldsymbol{\theta}_\text{E})\|_1 \right), \\
&\qquad \text{subject to } \|\boldsymbol{w}_{M+2,k}(\boldsymbol{\theta}_\text{D})\|_2^2 = \|\boldsymbol{C}_m\|_F = 1;\ k = 1, \ldots, K, m = 1, ..., M
\end{aligned} \tag{3.7}$$

where $\boldsymbol{W}_{M+1}(\boldsymbol{\theta}_\text{E}) = (\boldsymbol{w}_{M+1,1}(\boldsymbol{\theta}_\text{E}), \ldots, \boldsymbol{w}_{M+1,K}(\boldsymbol{\theta}_\text{E}))$ and $\boldsymbol{W}_{M+2}(\boldsymbol{\theta}_\text{D}) = (\boldsymbol{w}_{M+2,1}(\boldsymbol{\theta}_\text{D}), \ldots, \boldsymbol{w}_{M+2,K}(\boldsymbol{\theta}_\text{D})) \in \mathbb{R}^{p \times K}$, and τ_3 is the sparse parameter controlling the sparsity. $\boldsymbol{V}_K(\boldsymbol{\theta}_\text{E}) \in \mathbb{R}^{p \times K}$ is the loading matrix which is the composition of the convolutional layers and the fully connected layer.

To determine the tuning parameters of the proposed model structure where $\hat{\boldsymbol{V}}_K^{(\text{CE})}(M, \boldsymbol{d}, \tau_3)$ and $\hat{\boldsymbol{W}}_{M+2}$ are the estimated encoder and decoder matrices given M convolutional layers, filter size, $\boldsymbol{d} = (d_1, d_2)$, sparse parameter, τ_3, and number of estimated principal components, K, the tuning parameters are selected from the candidate set, S_3, using the validation data $\boldsymbol{X}_\text{valid} \in \mathbb{R}^{n' \times qr}$ where n' is the number of validation data. The approach is to minimize the reconstruction loss on the validation data (validation MSE), given by:

$$(\hat{M}, \hat{\boldsymbol{d}}, \hat{\tau}_3, \hat{K}_3) = \underset{(M, \boldsymbol{d}, \tau_3, K) \in S_3}{\text{argmin}} \|\boldsymbol{X}_\text{valid} - \boldsymbol{X}_\text{valid} \hat{\boldsymbol{V}}_K^{(\text{CE})}(M, \boldsymbol{d}, \tau_3) \hat{\boldsymbol{W}}_{M+2}'\|_F, \tag{3.8}$$

where the candidate set S_3 which is a predefined set that all the combinations of M, $\boldsymbol{d}$, τ_3 and K that we wish to evaluate to find the optimal values that minimize the validation MSE. After the training process and the selection of tuning parameters, the estimated loading matrix for the proposed SAE-C method is denoted as:

$$\hat{\boldsymbol{V}}_{\hat{K}_3}^{(\text{CE})}(\hat{M}, \hat{\boldsymbol{d}}, \hat{\tau}_3).$$

In our approach, the training process of (3.7) begins with the minimum values of τ_3, M, d_1 and d_2. We initialize the $\boldsymbol{W}_{M+1}$ and $\boldsymbol{W}_{M+2}$ matrices of the proposed SAE-C model using the PC loading matrix obtained from $\hat{\boldsymbol{V}}_K$. The $\boldsymbol{C}_m$ matrices are initialized as follows: If d_1 and d_2 are even:

$$c_{m,i,j} = \begin{cases} 1; & \text{if } i = d_1/2, d_1/2+1, j = d_2/2, d_2/2+1, \\ 0; & \text{if } i \neq d_1/2, d_1/2+1, j \neq d_2/2, d_2/2+1, \end{cases} \tag{3.9}$$

if d_1 and d_2 are odd:

$$c_{m,i,j} = \begin{cases} 1; & \text{if } i = \lfloor d_1/2 \rfloor + 1, j = \lfloor d_2/2 \rfloor + 1, \\ 0; & \text{if } i \neq \lfloor d_1/2 \rfloor + 1, j \neq \lfloor d_2/2 \rfloor + 1, \end{cases} \tag{3.10}$$

If d_1 and d_2 are even and odd, respectively:

$$c_{m,i,j} = \begin{cases} 1; & \text{if } i = d_1/2, d_1/2+1, j = \lfloor d_2/2 \rfloor + 1, \\ 0; & \text{if } i \neq d_1/2, d_1/2-1, j \neq \lfloor d_2/2 \rfloor + 1, \end{cases} \tag{3.11}$$

if d_1 and d_2 are odd and even, respectively:

$$c_{m,i,j} = \begin{cases} 1; & \text{if } i = \lfloor d_1/2 \rfloor + 1, j = d_2/2, d_2/2+1, \\ 0; & \text{if } i \neq \lfloor d_1/2 \rfloor + 1, j \neq d_2/2, d_2/2+1, \end{cases} \tag{3.12}$$

where $c_{m,i,j}$ is the (i, j)-th element of the convolution filter $\boldsymbol{C}_m$. The model then progressively trains with increasing τ_3, keeping M and $\boldsymbol{d}$ fixed. The $\hat{\boldsymbol{W}}_{M+1}$, $\hat{\boldsymbol{W}}_{M+2}$, and $\boldsymbol{C}_m$ matrices from the previous τ_3 step are used as initialization. Next, with τ_3 and $\boldsymbol{d}$ fixed, the training process continues by increasing M. We use the $\hat{\boldsymbol{W}}_{M+1}$, $\hat{\boldsymbol{W}}_{M+2}$, and $\boldsymbol{C}_j$'s matrices from the previous τ_3 and M as initialization, and the new convolution filters are initialized using (3.9) or (3.10). Finally, with τ_3 and M fixed, we use the $\hat{\boldsymbol{W}}_{M+1}$ and $\hat{\boldsymbol{W}}_{M+2}$ matrices from the previous τ_3 and M as initialization, and the convolution filters are initialized using (3.9), (3.10), (3.11) or (3.12).

In the decoder, convolutional layers are not considered because the decoder's primary purpose is to reconstruct the data. Unlike the encoder, which utilizes regularization and convolutional layers to filter out irrelevant information and capture spatial patterns, the decoder focuses on accurately reconstructing the original data.

The interplay of the model's key parameters: τ_3, $\boldsymbol{d}$, and M, significantly influences the characteristics of the estimated loading vectors. As the sparsity parameter τ_3 increases, it promotes greater sparsity in the estimated loading vectors, effectively zeroing out more elements and focusing on the most influential features. This increased sparsity can lead to more interpretable results, highlighting the most critical spatial patterns in the data. Conversely, the size of the convolutional filter, $\boldsymbol{d}$, affects the scale of spatial information captured. Smaller filter sizes, d_1 and d_2, allow the model to capture more localized spatial relationships, resulting in more detailed and fine-grained estimations of the loading vectors. This can be particularly useful when dealing with data that exhibits intricate local patterns. On the other hand, the depth of the encoder, represented by M, impacts the smoothness of the estimated loading vectors. A deeper encoder, with a larger M, considers a broader spatial context, potentially leading to smoother estimations. This is because each additional layer allows the model to integrate information from a wider area, potentially capturing more global patterns and trends in the data.

3.4 Nonlinear Spatial PCA Using Autoencoders

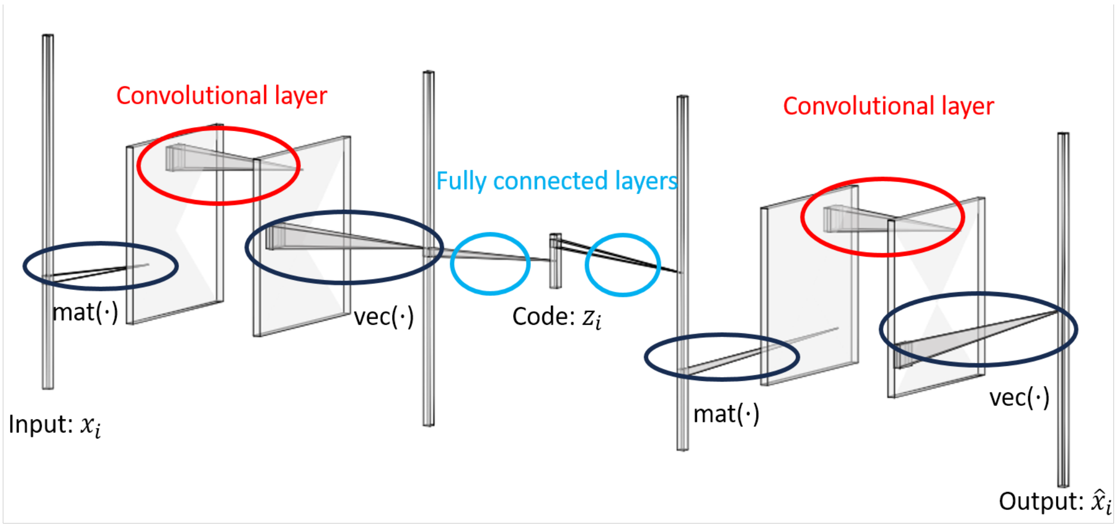


Figure 3.3: Model architecture of the proposed nonlinear autoencoder with convolutions for input vector $\boldsymbol{x}_i$ and output vector $\hat{\boldsymbol{x}}_i$. First, reshape the vector into a matrix using mat(·), then pass it through convolutional layers. Next, reshape the output back to a vector using vec(·) and pass it through a fully connected layer to reduce dimensionality. After that, pass it through another fully connected layer to restore the original dimension. Finally, pass it through convolutional layers to complete the reconstruction.

In this section, we extend the framework of Nonlinear Autoencoders with Convolutions (NAE-C) to a nonlinear context by modifying the model to include an equal number of convolutional layers in both the encoder and the decoder, as illusatrated in Figure 3.3. Unlike the linear variant that employed identity activation functions, the nonlinear approach utilizes non-linear activation functions. These modifications are designed to preserve spatial structures in the data while capturing more complex patterns.

The loss function of the autoencoder with convolutions, denoted similar to (2.8), involves trainable parameters $\boldsymbol{\theta}_{\mathrm{E}} = \{\boldsymbol{C}_1, ..., \boldsymbol{C}_M, \boldsymbol{B}_1, ..., \boldsymbol{B}_M, \boldsymbol{W}_{M+1}, \boldsymbol{b}_{M+1}\}$ and $\boldsymbol{\theta}_{\mathrm{D}} = \{\boldsymbol{W}_{M+2}, \boldsymbol{b}_{M+2}, \boldsymbol{C}_{M+3}, ..., \boldsymbol{C}_{2M+2}, \boldsymbol{B}_{M+3}, ..., \boldsymbol{B}_{2M+2}\}$. Here, the weight matrices in (2.8) are replaced by convolutional filters, and convolutional operations are applied. The encoder $f_{\mathrm{E}} : \mathbb{R}^p \rightarrow \mathbb{R}^K$

parametrized by $\boldsymbol{\theta}_{\mathrm{E}}$, and decoder $f_{\mathrm{D}} : \mathbb{R}^K \rightarrow \mathbb{R}^p$ parametrized by $\boldsymbol{\theta}_{\mathrm{D}}$ are defined as follows:

$$\begin{aligned} f_{\mathrm{E}}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathrm{E}}) =& f_{M+1}(\mathrm{vec}(g_{\boldsymbol{C}_M} \circ \cdots \circ g_{\boldsymbol{C}_2} \circ g_{\boldsymbol{C}_1}(\mathrm{mat}(\boldsymbol{x})|\boldsymbol{\theta}_{\mathrm{E}}))|\boldsymbol{\theta}_{\mathrm{E}}) = \boldsymbol{z}, \\ f_{\mathrm{D}}(\boldsymbol{z}|\boldsymbol{\theta}_{\mathrm{D}}) =& \mathrm{vec}(g_{\boldsymbol{C}_{2M+2}} \circ \cdots \circ g_{\boldsymbol{C}_{M+3}}(\mathrm{mat}(f_{M+2}(\boldsymbol{z}|\boldsymbol{\theta}_{\mathrm{D}}))|\boldsymbol{\theta}_{\mathrm{D}}) = \hat{\boldsymbol{x}}. \end{aligned}$$

Additionally, the activation functions are ReLU for every layer except the last layer, where the activation function is Identity. Each encoder and decoder includes one fully connected layer right after and before convolutional layers, respectively, which linearly reduces and increases the dimension. NAE-C solves the optimization problem:

$$(\hat{\boldsymbol{\theta}}_{\mathrm{E}}, \hat{\boldsymbol{\theta}}_{\mathrm{D}}) = \underset{(\boldsymbol{\theta}_{\mathrm{E}}, \boldsymbol{\theta}_{\mathrm{D}})}{\mathrm{argmin}} \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - f_{\mathrm{D}} \circ f_{\mathrm{E}}(\boldsymbol{x}_i|\boldsymbol{\theta}_{\mathrm{E}}, \boldsymbol{\theta}_{\mathrm{D}})\|_2^2. \tag{3.13}$$

Unlike the linear case, the objective function for the NAE-C does not include the ℓ_1 sparsity penalty. Instead, it focuses on achieving high-fidelity reconstruction and extracting meaningful spatial codes. Employing multiple filters and a symmetric encoder-decoder structure is anticipated to enable the NAE-C to effectively capture and reconstruct complex patterns in high-dimensional data while maintaining spatial and structural integrity.

The tuning parameters in this case are the dimension of codes K, the size of the convolutional filters d, and the number of convolutional layers M from the candidate set $S_{(\mathrm{n})}$. Following the similar manner of hyperparameter tuning as (3.8), we use validation data for parameter tuning:

$$(\hat{M}_{(\mathrm{n})}, \hat{d}_{(\mathrm{n})}, \hat{K}_{(\mathrm{n})}) = \underset{(M,d,K) \in S_{(\mathrm{n})}}{\mathrm{argmin}} \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i^{(\mathrm{v})} - f_{\mathrm{D}} \circ f_{\mathrm{E}}(\boldsymbol{x}_i^{(\mathrm{v})}|\hat{\boldsymbol{\theta}}_{\mathrm{E}}(M,d,K), \hat{\boldsymbol{\theta}}_{\mathrm{D}}(M,d,K)\|_2^2. \tag{3.14}$$

Thus, the estimated encoder and decoder parameters for the NAE-C method are:

$$\hat{\boldsymbol{\theta}}_{\mathrm{E}}(\hat{M}_{(\mathrm{n})}, \hat{d}_{(\mathrm{n})}, \hat{K}_{(\mathrm{n})}) \text{ and } \hat{\boldsymbol{\theta}}_{\mathrm{D}}(\hat{M}_{(\mathrm{n})}, \hat{d}_{(\mathrm{n})}, \hat{K}_{(\mathrm{n})}). \tag{3.15}$$

To estimate the loading vectors for NAE-C method, we employ the first-order Taylor expansion. The Taylor expansion allows us to approximate the nonlinear transformation of the autoencoder with a linear one around a given point, thereby facilitating the estimation of the loading vectors.

Given the encoder $f_{\mathrm{E}}(\cdot|\boldsymbol{\theta}_{\mathrm{E}})$, the first-order Taylor expansion of the encoder output $\boldsymbol{z} \in \mathbb{R}^K$ around a data point $\boldsymbol{x}_0 \in \mathbb{R}^p$ is expressed as:

$$f_{\mathrm{E}}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathrm{E}}) \approx f_{\mathrm{E}}(\boldsymbol{x}_0|\boldsymbol{\theta}_{\mathrm{E}}) + \nabla_{\boldsymbol{x}} f_{\mathrm{E}}(\boldsymbol{x}_0|\boldsymbol{\theta}_{\mathrm{E}})(\boldsymbol{x} - \boldsymbol{x}_0),$$

where $\nabla_{\boldsymbol{x}} f_{\mathrm{E}}(\boldsymbol{x}_0|\boldsymbol{\theta}_{\mathrm{E}})$ denotes the Jacobian matrix (Waldron et al. [1985]) of the encoder evaluated at $\boldsymbol{x}_0$. The Jacobian matrix of a vector-valued function $\boldsymbol{f} : \mathbb{R}^p \to \mathbb{R}^K$ with respect to $\boldsymbol{x} = (x_1, x_2, \ldots, x_p)'$ is defined as:

$$\nabla_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_p} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_p} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_K}{\partial x_1} & \frac{\partial f_K}{\partial x_2} & \cdots & \frac{\partial f_K}{\partial x_p} \end{bmatrix}.$$

This linear approximation enables us to interpret the autoencoder's behavior in terms of loading vectors, which are the rows of the Jacobian matrix. Formally, the loading vectors $\hat{\boldsymbol{v}}_i^{(\mathrm{NCE})}$ for the i-th data point $\boldsymbol{x}_i$ are the rows of the Jacobian matrix:

$$\hat{\boldsymbol{V}}^{(\mathrm{NCE})} = \nabla_{\boldsymbol{x}} f_{\mathrm{E}}(\boldsymbol{x}_i|\hat{\boldsymbol{\theta}}_{\mathrm{E}}), \tag{3.16}$$

where $\hat{\boldsymbol{V}}^{(\mathrm{NCE})} = (\hat{\boldsymbol{v}}_1^{(\mathrm{NCE})}, ..., \hat{\boldsymbol{v}}_K^{(\mathrm{NCE})})'$ This approach ensures that the loading vectors are adapted to the local structure of the data, capturing the intrinsic nonlinear relationships while preserving spatial information. By leveraging the first-order Taylor expansion, we achieve a robust and interpretable estimation of the loading vectors for nonlinear spatial PCA using autoencoders.

Chapter 4

Simulations

In this section, we compare the performance of different methods: PCA, SPCA, SAE, and SAE-C with varying filter sizes and numbers of convolutional layers for one-dimensional (1D) case and two-dimensional (2D) case.

4.1 PCA: One-Dimensional Data Simulation

In the 1D data simulation, the simulated data $\boldsymbol{x}_i \in \mathbb{R}^{100}$ for $i = 1, \dots, 120$ are generated by the following equation:

$$\boldsymbol{x}_i = z_{i,1}\boldsymbol{v}_1 + z_{i,2}\boldsymbol{v}_2 + \boldsymbol{\epsilon}_i, \tag{4.1}$$

where $\boldsymbol{z}_i = (z_{i,1}, z_{i,2})' \sim N(\mathbf{0}, \text{diag}(9, 4))$, $\boldsymbol{\epsilon}_i \sim N(\mathbf{0}, \boldsymbol{I}) \in \mathbb{R}^{100}$, and $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^{100}$. The two loading vectors, $\boldsymbol{v}_1 = (v_1(s_1), \dots, v_1(s_{100}))'$ and $\boldsymbol{v}_2 = (v_2(s_1), \dots, v_2(s_{100}))'$, are defined at 100 1D spatial locations $s_j = -5 + j/10,\ j = 1, \dots, 100$, as follows:

$$v_1(s_j) = \begin{cases} c_1 \exp\left(-(s_j - 2)^2/0.5\right); & j = 54, \dots, 87, \\ 0; & j = 1, \dots, 53, 88, \dots, 100, \end{cases}$$

$$v_2(s_j) = \begin{cases} c_2 \exp\left(-(s_j + 2)^2/0.5\right); & j = 14, \dots, 47, \\ 0; & j = 1, \dots, 13, 48, \dots, 100. \end{cases}$$

Here, c_1 and c_2 are scaling constants such that $\|\boldsymbol{v}_1\|_2 = \|\boldsymbol{v}_2\|_2 = 1$, where $\|\cdot\|_2$ denotes the ℓ_2 norm. The loading vectors are shown in Figure 4.1. Given these simulated data, we can estimate the loading matrix using SAE-C (3.7) to obtain $\hat{\boldsymbol{V}}_K^{(\text{CE})}(M, \boldsymbol{d}, \tau_3)$ with 100×1 as the reshaped

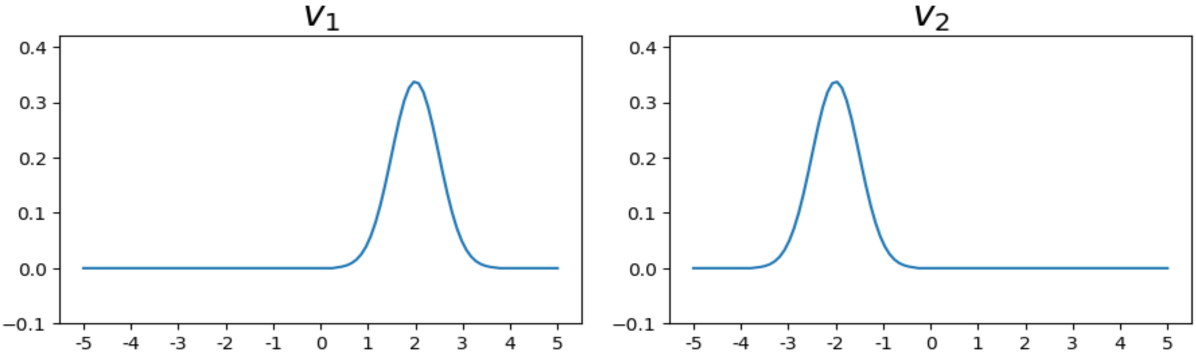


Figure 4.1: Two loading vectors, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, for 1D data simulation.

dimension in the mat$(\cdot)$ (3.4) and vec$(\cdot)$ (3.5).

In this experiment, we aimed to investigate how varying key parameters—the number of convolutional layers (M), different filter sizes ($\boldsymbol{d}$), and sparse parameters (τ_3)—affect the estimation of loading vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ in a 1D data simulation context. The candidate set for the tuning parameters (3.8) for $\hat{\boldsymbol{V}}_K^{(\mathrm{CE})}(M, \boldsymbol{d}, \tau_3)$, includes:

$$\begin{aligned} S_3 = &\{(M, \boldsymbol{d}, \tau_3, K) | M = 1, 2, 4, 7, \boldsymbol{d} = (7,1), (11,1), (15,1), (19,1), \\ &\tau_3 = 0, 0.005, 0.01, \ldots, 0.1, K = 2, 3, 4\}. \end{aligned}$$

We select the optimal tuning parameters based on the validation MSE using 80 validation data $\boldsymbol{X}_{\text{valid}} \in \mathbb{R}^{80\times 100} = (\boldsymbol{x}_1^{(\mathrm{v})}, ..., \boldsymbol{x}_{100}^{(\mathrm{v})})'$ where $\boldsymbol{x}_i^{(\mathrm{v})}$'s are generated in the same manner as (4.1). Given the need to explore all combinations of tuning parameters, we can further study the effects of changes in these key parameters.

First, as shown in Figure 4.2, we investigated the effect of the number of convolutional layers M. We kept the filter size $\boldsymbol{d} = (7, 1)$, the sparse parameter $\tau_3 = 0.05$, and the number of principal components $K = 2$ constant while varying the number of convolutional layers M (1, 2, 4, 7). The results showed that increasing the number of convolutional layers led to smoother estimated loading vectors. For smaller M values, the estimated loading vectors were rough and primarily captured the main peaks of the loading vectors. As M increased, the loading vectors became smoother but also introduced slight noise or additional oscillations outside the main peak regions, indicating a trade-off between sparsity and detail: higher M values enhance the model's capacity to capture detailed spatial patterns at the cost of potential overfitting.

Second, as illustrated in Figure 4.3, we examined the effect of the filter size $\boldsymbol{d}$. We held the number of convolutional layers $M = 4$, the sparse parameter $\tau_3 = 0.05$, and the number of principal components $K = 2$ constant, varying the filter size $\boldsymbol{d}$ $((7,1),(11,1),(15,1),(19,1))$. The observations revealed that smaller filter sizes resulted in more sparse loading estimates. Smaller filter sizes (e.g., $\boldsymbol{d} = (7,1)$) produced sparser estimations, while larger filter sizes (e.g., $\boldsymbol{d} = (19,1)$) provided smoother estimations of the loading vectors, capturing the primary peaks but potentially causing oversmoothing. This experiment highlighted that adjusting the filter size can control the sparseness of the estimated loading vectors, which is crucial for accurately capturing underlying spatial patterns.

Third, as depicted in Figure 4.4, we investigated the effect of the sparsity parameter τ_3. We kept the number of convolutional layers $M = 4$, the filter size $\boldsymbol{d} = (7,1)$, and the number of principal components $K = 2$ constant while varying the sparsity parameter τ_3 (0, 0.025, 0.05, 0.075). The findings indicated that higher values of τ_3 led to sparser loading vector estimates. However, the degree of sparsity was also affected by the filter size $\boldsymbol{d}$. With $\tau_3 = 0$, the estimates were very noisy and exhibited numerous fluctuations. As τ_3 increased, the loading vectors became sparser, with more clearly defined peaks and reduced noise. Higher sparsity parameters effectively captured the core features of the data while minimizing unnecessary fluctuations, but the filter size $\boldsymbol{d}$ played a crucial role in determining the overall sparsity.

These demonstrations showed how key parameters influence the estimation of loading vectors in a 1D data simulation, reflecting similar trends analyzed in Chapter 3.3. Increasing the number of convolutional layers enhances the model's ability to capture detailed patterns but carries the risk of overfitting. Larger filter sizes result in smoother loading vector estimations, crucial for accurately capturing spatial patterns. Higher sparsity parameters promote sparsity, reducing noise and fluctuations, and helping to capture the main features of the data.

In the next experiment, we aim to compare our proposed SAE-C method with PCA, SPCA, and SAE. For selecting the number of principal components K for the loading matrix of PCA $\hat{\boldsymbol{V}}_K$ (2.3), we use the validation MSE with the validation data $\boldsymbol{X}_{\text{valid}}$ to select K from the set

$S_0 = \{K|K = 2, 3, 4\}$,

$$\hat{K}_0 = \underset{K\in S_0}{\text{argmin}}\, \|\boldsymbol{X}_{\text{valid}} - \boldsymbol{X}_{\text{valid}}\hat{\boldsymbol{V}}_K\hat{\boldsymbol{V}}_K'\|_F. \tag{4.2}$$

For selecting the number of principal components K and the sparse parameter τ_1 for the loading matrix of SPCA $\hat{\boldsymbol{V}}_K^{(\text{s})}$ and the corresponding matrix $\hat{\boldsymbol{A}}$ (2.4), we also use the validation MSE with the validation data $\boldsymbol{X}_{\text{valid}}$ to select both K and τ_1 from the set $S_1 = \{(\tau_1, K)|\tau_1 = 0, 0.01, \ldots, 1, K = 2, 3, 4\}$. Here, τ_0 set to be constant 0.01,

$$(\hat{\tau}_1, \hat{K}_1) = \underset{(K,\tau_1)\in S_1}{\text{argmin}}\, \|\boldsymbol{X}_{\text{valid}} - \boldsymbol{X}_{\text{valid}}\hat{\boldsymbol{V}}_K^{(\text{s})}(\tau_1)\hat{\boldsymbol{A}}'\|_F. \tag{4.3}$$

For selecting sparse parameter τ_2 and the number of principal components K for the loading matrix of SAE $\hat{\boldsymbol{V}}_K^{(\text{SE})}(\tau_2)$ (3.1), the candidate set for the tuning parameters, denoted as S_2 in (3.2), is $\{(\tau_2, K)|\tau_2 = 0, 0.05, 0.1, \ldots, 1, K = 2, 3, 4\}$.

To evaluate the performance of the models, we used testing data $\boldsymbol{X}_{\text{test}} = (\boldsymbol{x}_1^{(t)}, \cdots, \boldsymbol{x}_{100}^{(t)})' \in \mathbb{R}^{100\times 100}$, where $\boldsymbol{x}_i^{(t)}$ are generated in the same manner as (4.1). We evaluated model performance using the testing reconstruction loss (Testing MSE):

$$\text{Testing MSE} = \|\boldsymbol{X}_{\text{test}} - \boldsymbol{X}_{\text{test}}\tilde{\boldsymbol{V}}_{\hat{K}}\tilde{\boldsymbol{D}}'\|_F, \tag{4.4}$$

where $\boldsymbol{X}_{\text{test}}$ is the testing data, $\tilde{\boldsymbol{V}}_{\hat{K}} = (\tilde{\boldsymbol{v}}_1, \ldots, \tilde{\boldsymbol{v}}_{\hat{K}})$ is the estimated loading matrix obtained from all the PCA methods, and $\tilde{\boldsymbol{D}}$ is the corresponding matrix, which is $\hat{\boldsymbol{V}}_{\hat{K}_0}$ for PCA (4.2), $\hat{\boldsymbol{A}}$ for SPCA (4.3), $\hat{\boldsymbol{W}}_2$ for SAE (3.2), and $\hat{\boldsymbol{W}}_{\hat{M}+2}$ for SAE-C (3.8).

To further evaluate the estimations, we considered the Frobenius norm of the difference between the true projection matrix and the estimated projection matrix, denoted as:

$$\|\boldsymbol{V}_K\boldsymbol{V}_K' - \tilde{\boldsymbol{V}}_{\hat{K}}(\tilde{\boldsymbol{V}}_{\hat{K}}'\tilde{\boldsymbol{V}}_{\hat{K}})^{-1}\tilde{\boldsymbol{V}}_{\hat{K}}'\|_F, \tag{4.5}$$

where $\boldsymbol{V}_K\boldsymbol{V}_K'$ represents the true projection matrix, $\boldsymbol{V}_{\hat{K}} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{\hat{K}})$, and $\tilde{\boldsymbol{V}}_{\hat{K}}(\tilde{\boldsymbol{V}}_{\hat{K}}'\tilde{\boldsymbol{V}}_{\hat{K}})^{-1}\tilde{\boldsymbol{V}}_{\hat{K}}'$

is the estimated projection matrix where $\tilde{\boldsymbol{V}}_{\hat{K}}$ is the estimate by all PCA method.

We replicated the simulation 100 times to ensure the robustness of our results, with each replication taking approximately 30 minutes on a Google Colab Tesla T4 GPU. Training was performed with early stopping if the loss did not change by at least 0.01 within 20 epochs, with a maximum of 100 epochs allowed. This extensive simulation provides a comprehensive assessment of the model's performance. Based on the results presented in Figure 4.5 (a), our method SAE-C outperforms traditional techniques such as PCA, SPCA, and SAE in terms of testing MSE. Additionally, the estimated projection matrix demonstrates a higher degree of similarity to the true projection matrix, as depicted in Figure 4.5 (b). Furthermore, when examining the estimations of the loading vectors in Figure 4.6, our method exhibits the ability to capture smooth loading vectors, indicating effectiveness in extracting spatial information from the data, and identify the order of $\hat{\boldsymbol{v}}_k^{(\mathrm{EC})}$ by calculating the sample variance of the estimated codes, similarly to PCA. The corresponding loading vector with a larger sample variance has a higher order.

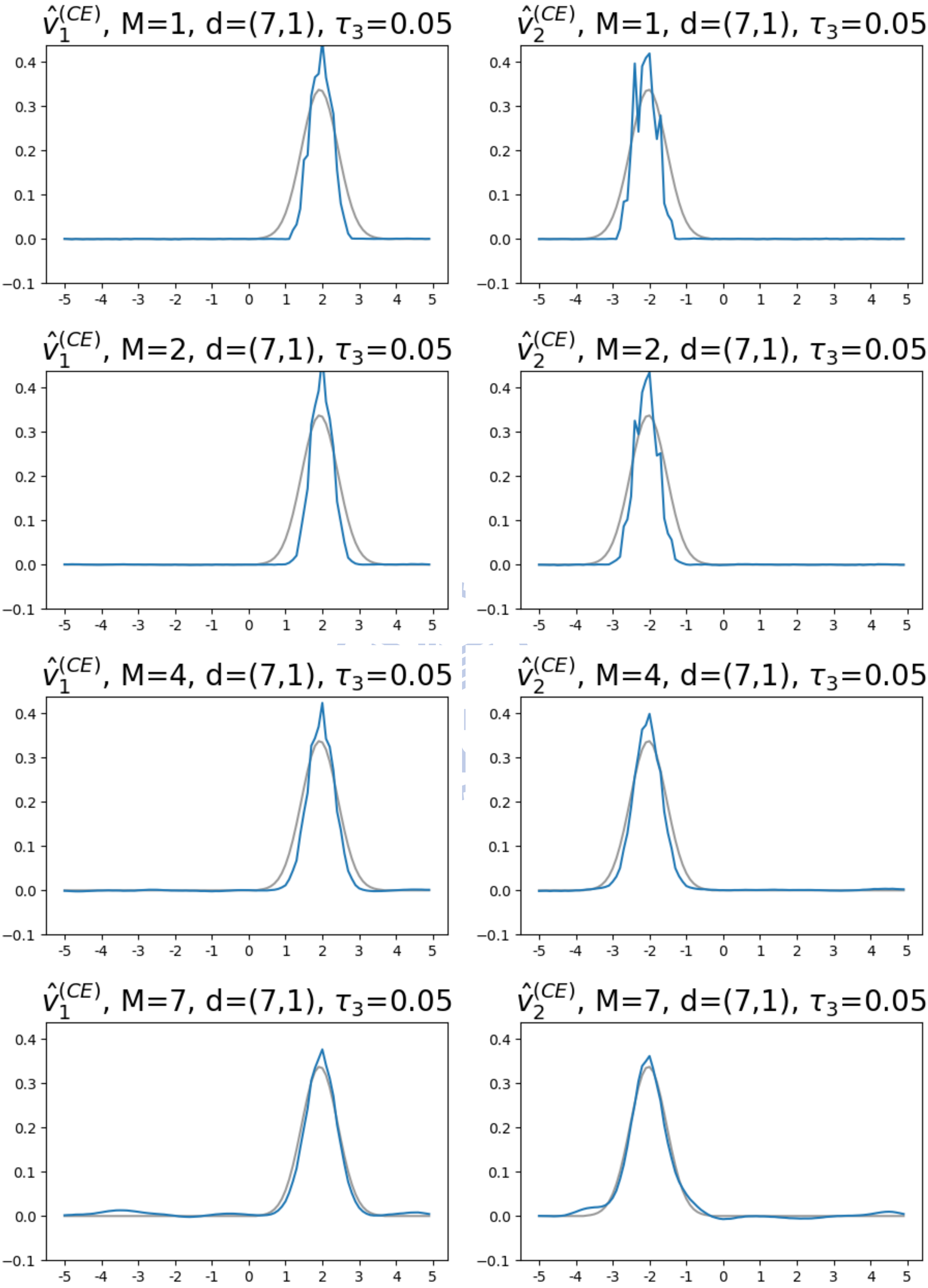


Figure 4.2: Effect of the number of convolutional layers (M): Estimated loading vectors for different numbers of convolutional layers (M) with fixed filter size $\boldsymbol{d} = (7, 1)$, sparse parameter $\tau_3 = 0.05$, and the number of principal components $K = 2$.

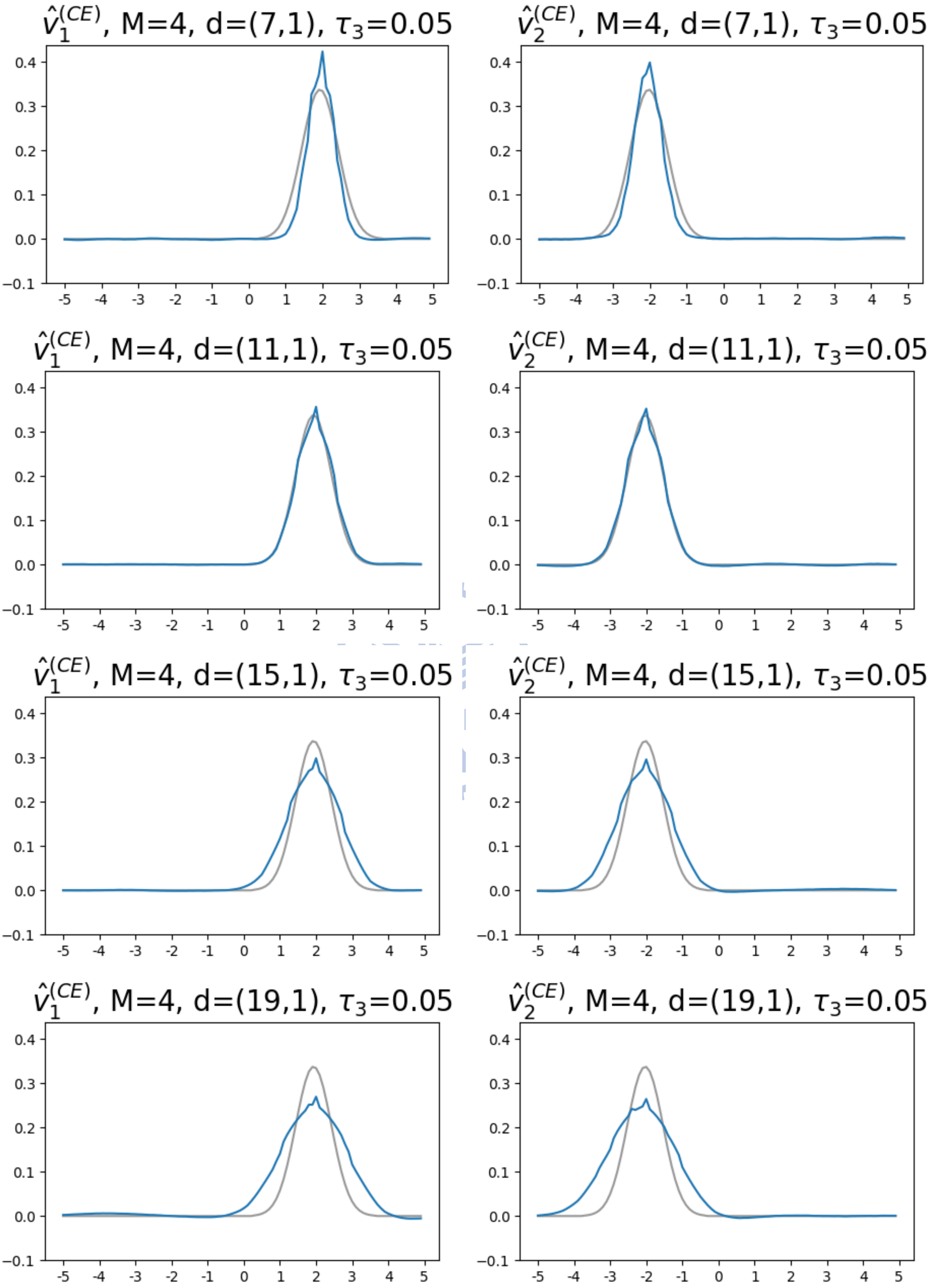


Figure 4.3: Effect of filter size ($\boldsymbol{d}$): Estimated loading vectors for different size of convolutional layers ($\boldsymbol{d}$) with fixed number of filter $M = 4$, sparse parameter $\tau_3 = 0.05$, and the number of principal components $K = 2$.

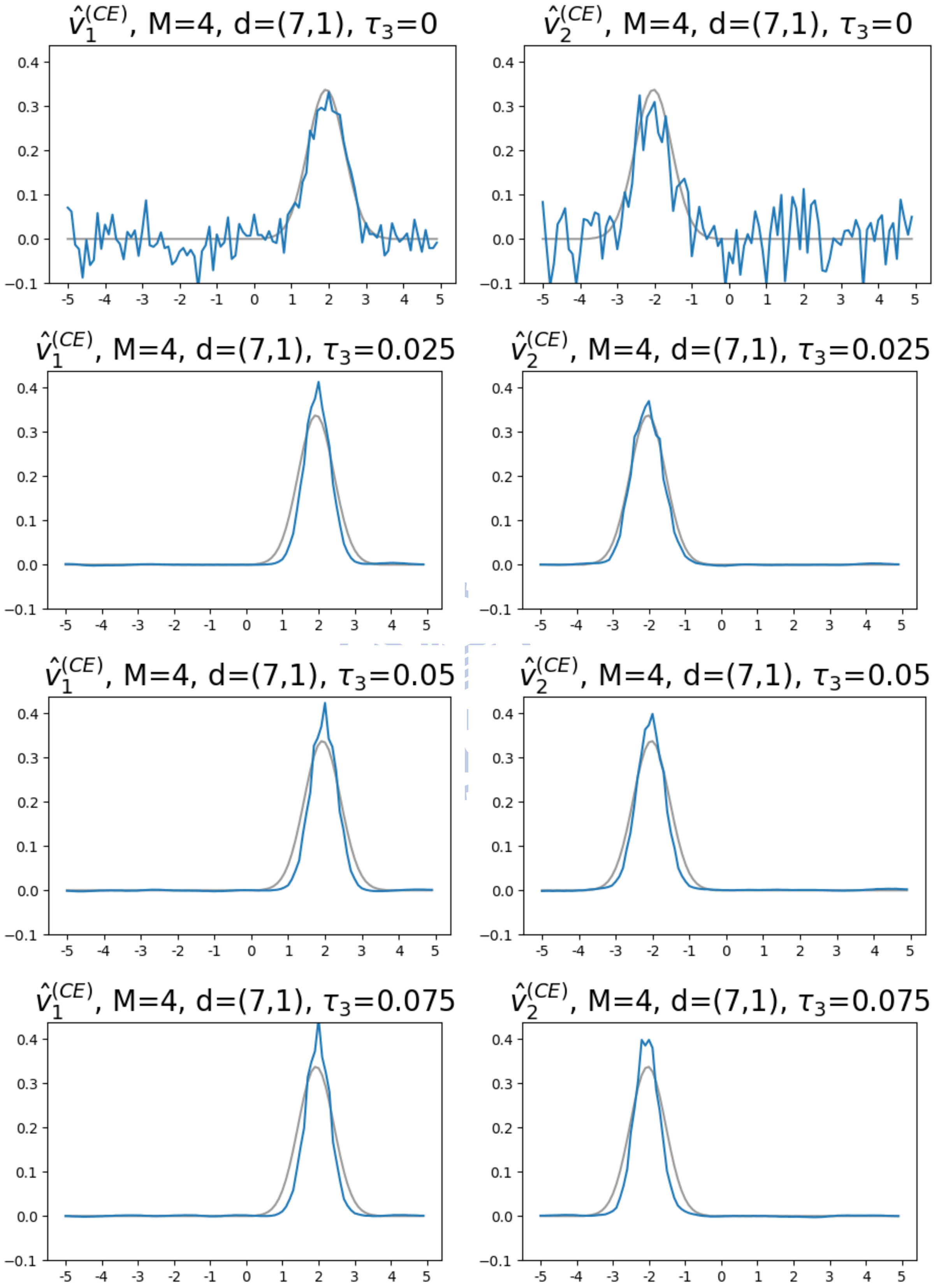
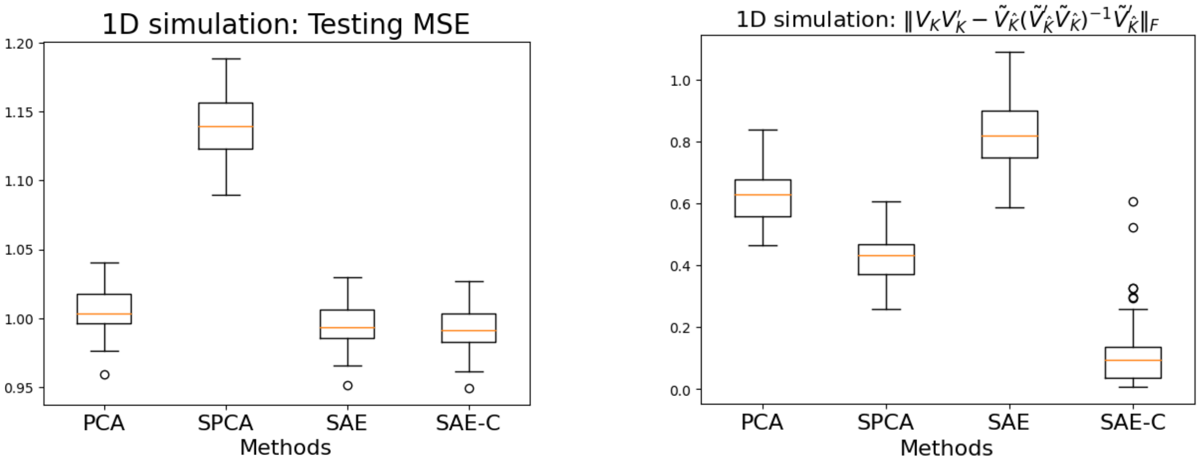


Figure 4.4: Effect of sparse parameter (τ_3): Estimated loading vectors for different sparse parameter (τ_3) with fixed number of convolutional layers $M = 4$, filter size $\boldsymbol{d} = (7, 1)$, and the number of principal components $K = 2$.



(a) (b)

Figure 4.5: Model performance for 1D data simulation: Testing MSE (4.4) and Frobenius norm of the difference between the true projection matrix and the estimated projection matrix (4.5) with 100 repeated simulations.

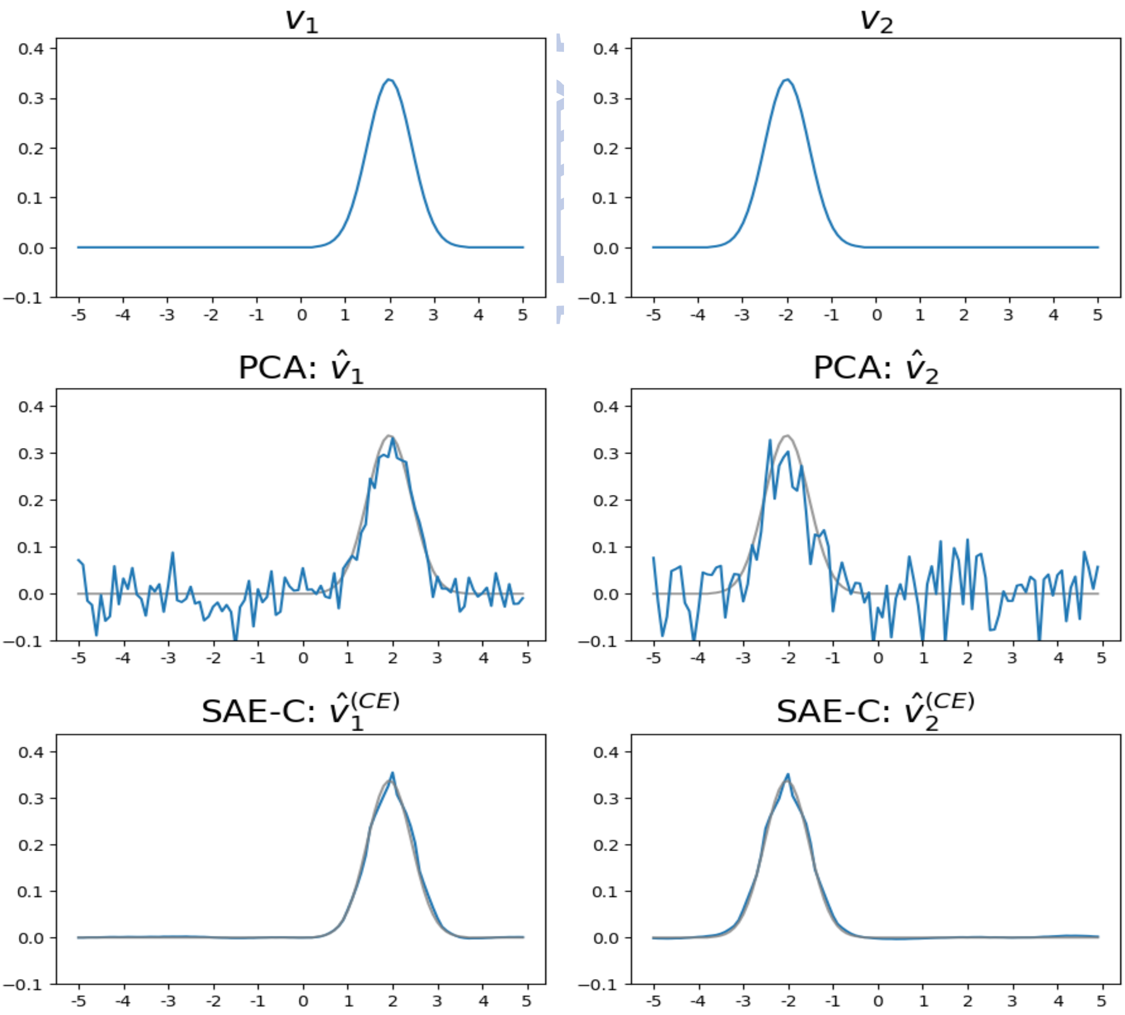


Figure 4.6: Estimated loading vectors for 1D data simulation using PCA and SAE-C.

4.2 PCA: Two-Dimensional Data Simulation

In the 2D data simulation, the simulated data $\boldsymbol{x}_i$ for $i = 1, \ldots, 500$, using the same generating formula (4.1) with $\boldsymbol{z}_i = (z_{i,1}, z_{i,2})' \sim N(\mathbf{0}, \text{diag}(9, 4))$, $\boldsymbol{\epsilon}_i \sim N(\mathbf{0}, \boldsymbol{I}) \in \mathbb{R}^{400}$, and $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^{400}$. The two loading vectors, $\boldsymbol{v}_1 = (v_1(s_1), \ldots, v_1(s_{400}))'$ and $\boldsymbol{v}_2 = (v_2(s_1), \ldots, v_2(s_{400}))'$, are defined at 400 2D spatial locations, denoted as $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{400}$, where each $\boldsymbol{s}_j = (s_{xj}, s_{yj})$ is equally spaced in $[-5, 5]^2$, as follows:

$$v_1(x, y) = \begin{cases} c_3 \exp\left(-0.2((x-3)^2 + (y-3)^2)\right); & (x, y) \in \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{400}\}, \\ 0; & \text{otherwise}, \end{cases} \tag{4.6}$$

$$v_2(x, y) = \begin{cases} c_4 \exp\left(-0.2((x+3)^2 + (y+3)^2)\right); & (x, y) \in \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{400}\}, \\ 0; & \text{otherwise}, \end{cases} \tag{4.7}$$

Here, c_3 and c_4 are scaling constants such that $\|\boldsymbol{v}_1\|_2 = \|\boldsymbol{v}_2\|_2 = 1$. The loading vectors which are sparse, mutually orthogonal are shown in Figure 4.7. Given these simulated data, we can estimate the loading matrix using SAE-C (3.7) to obtain $\hat{\boldsymbol{V}}_K^{(\text{CE})}(M, \boldsymbol{d}, \tau_3)$ with 20×20 as the reshaped dimension in mat$(\cdot)$ (3.4) and vec$(\cdot)$ (3.5).

Similar to Chapter 4.2, in the following experiment, we focused on understanding how vary-

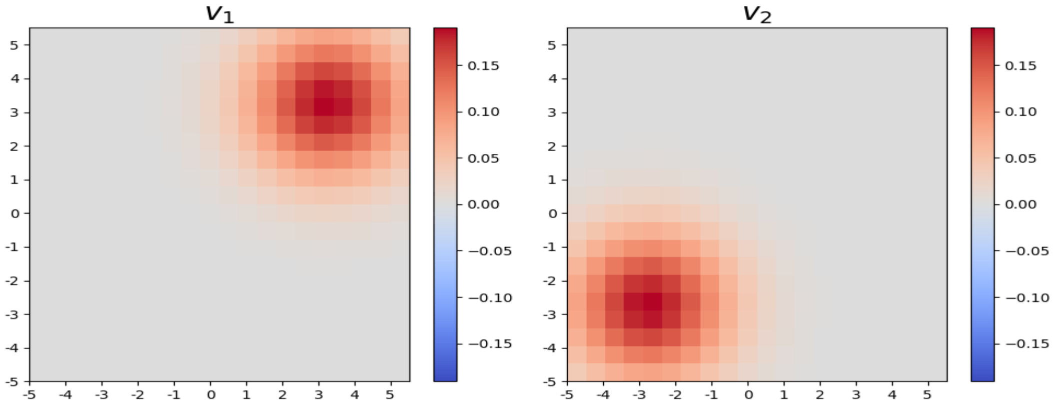


Figure 4.7: Two loading vectors, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, for 2D data simulation.

ing key parameters—the number of convolutional layers (M), different filter sizes ($\boldsymbol{d}$), and sparse parameters (τ_3)—affects the estimation of loading vectors, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, in a 2D data simulation scenario. The candidate set in (3.8) for $\hat{\boldsymbol{V}}_K^{(\mathrm{CE})}(M, d, \tau_3)$ is

$$S_3 = \{(M, \boldsymbol{d}, \tau_3, K) \mid M = 1, 2, 4,\ \boldsymbol{d} = (3,3), (7,7), (11,11),\ \tau_3 = 0, 0.0005, 0.001, \ldots, 0.01,\\ K = 2, 3, 4\}.$$

The optimal parameters are selected using validation MSE (3.8) for $\hat{\boldsymbol{V}}_K^{(\mathrm{CE})}(M, \boldsymbol{d}, \tau_3)$ with 200 validation data $\boldsymbol{X}_{\text{valid}} = (\boldsymbol{x}_1^{(\mathrm{v})}, \ldots, \boldsymbol{x}_{200}^{(\mathrm{v})})' \in \mathbb{R}^{200\times 400}$ where $\boldsymbol{x}_i^{(\mathrm{v})}$'s are generated in the same manner as (4.1). Given the need to explore all combinations of tuning parameters, we can further study the effects of changes in these key parameters.

First, as shown in Figure 4.8, we investigated the effect of the number of convolutional layers M. We kept the convolution filter size $\boldsymbol{d} = (7,7)$, the sparsity parameter $\tau_3 = 0.005$, and the number of principal components $K = 2$ constant, varying the number of convolutional layers M $(1, 2, 4)$. The outcomes demonstrated that increasing M resulted in smoother loading vector estimates. With fewer layers, the estimates were more irregular and noisy, capturing the principal peaks with less clarity. As M increased, the smoothness of the loading vectors improved, indicating that more layers help in refining the spatial patterns captured by the model.

Second, we studied the effect of filter size $\boldsymbol{d}$. We fixed the number of convolutional layers at $M = 2$, the sparsity parameter $\tau_3 = 0.005$, and the number of principal components $K = 2$, while varying the convolution filter size $\boldsymbol{d}$ $((3,3), (7,7), (11,11))$. Results (Figure 4.9) showed that larger filter sizes produced smoother loading vector estimates. Smaller filter sizes (e.g., $\boldsymbol{d} = (3,3)$) led to rougher estimates with more noise, whereas larger filter sizes (e.g., $\boldsymbol{d} = (11,11)$) yielded smoother, more precise loading vectors that better captured the main peaks with minimal noise. This suggests that larger filters are more effective in generating smooth loading vectors, which are essential for accurately representing spatial patterns.

Third, we investigated the effect of the sparsity parameter τ_3. We kept the number of convolutional layers $M = 2$, the filter size $\boldsymbol{d} = (7,7)$, and the number of principal components $K = 2$

constant while adjusting the sparsity parameter τ_3 (0, 0.0025, 0.005, 0.0075). The findings (Figure 4.10) indicated that higher values of τ_3 led to sparser loading vector estimates. However, the degree of sparsity was also affected by the filter size $\boldsymbol{d}$. As τ_3 increased, the loading vectors became sparser, with more clearly defined peaks and reduced noise. Higher sparsity parameters effectively captured the core features of the data while minimizing unnecessary fluctuations, but the filter size $\boldsymbol{d}$ played a crucial role in determining the overall sparsity.

These demonstrations showed how key parameters influence the estimation of loading vectors in a 2D data simulation, reflecting similar trends to those analyzed in Chapter 3.3. Increasing the number of convolutional layers enhances the smoothness of the loading vectors, larger filter sizes lead to smoother and more accurate estimations, and higher sparsity parameters promote sparsity in the loading vectors, effectively reducing noise and highlighting key data features.

In the next experiment, similar to the previous chapter, we aim to compare our proposed SAE-C method with PCA, SPCA, and SAE. To estimate the loading matrix for PCA $\hat{\boldsymbol{V}}_K$, we determine the number of principal components K by minimizing the validation MSE using the validation dataset $\boldsymbol{X}_{\text{valid}}$ (4.2). The values of K considered are in the set $S_0 = \{K|K = 2, 3, 4\}$. For SPCA, the estimation of the loading matrix $\hat{\boldsymbol{V}}_K^{(\mathrm{s})}$ also relies on the validation MSE with $\boldsymbol{Y}_{\text{valid}}$ (4.3). We select both the number of principal components K and the sparse parameter τ_1 from the set $S_1 = \{(\tau_1, K)|\tau_1 = 0, 0.01, \ldots, 1, K = 2, 3, 4\}$. For SAE, the loading matrix $\hat{\boldsymbol{V}}_K^{(\mathrm{SE})}(\tau_2)$ is estimated by selecting the tuning parameters from the candidate set S_2 as defined in (3.2). The set S_3 is given by $\{(\tau_2, K) \mid \tau_2 = 0, 0.05, 0.1, \ldots, 1, K = 2, 3, 4\}$.

To evaluate the performance of the models, we also utilize the testing MSE (4.4) with 300 testing data points $\boldsymbol{X}_{\text{test}} = (\boldsymbol{x}_1^{(\mathrm{t})}, ..., \boldsymbol{x}_{300}^{(\mathrm{t})})' \in \mathbb{R}^{300\times 400}$, where $\boldsymbol{x}_i^{(\mathrm{t})}$'s are generated following the same process as described in (4.1). We replicated the simulation 100 times to ensure the robustness of our results, with each replication taking approximately 12 minutes on a Google Colab Tesla T4 GPU. Training was performed with early stopping if the loss did not change by at least 0.01 within 20 epochs, with a maximum of 100 epochs allowed. This extensive simulation provides a comprehensive assessment of the model's performance. We also evaluate the performance by considering the Frobenius norm of the difference between the true projection matrix and the estimated projection matrix (4.5). We observe favorable performance in terms of the testing MSE (Figure 4.11 (a)). Moreover, the estimated projection matrix (Figure 4.11 (b)) aligns well with the true projection matrix, indicating the model's capability in accurately estimating the underlying projection structure. The 2D simulation results demonstrate the effectiveness of our method (Figure 4.12). Regarding the estimation of loadings, our model not only demonstrates superior performance but also exhibits smoother loadings, which closely resemble the true loadings.

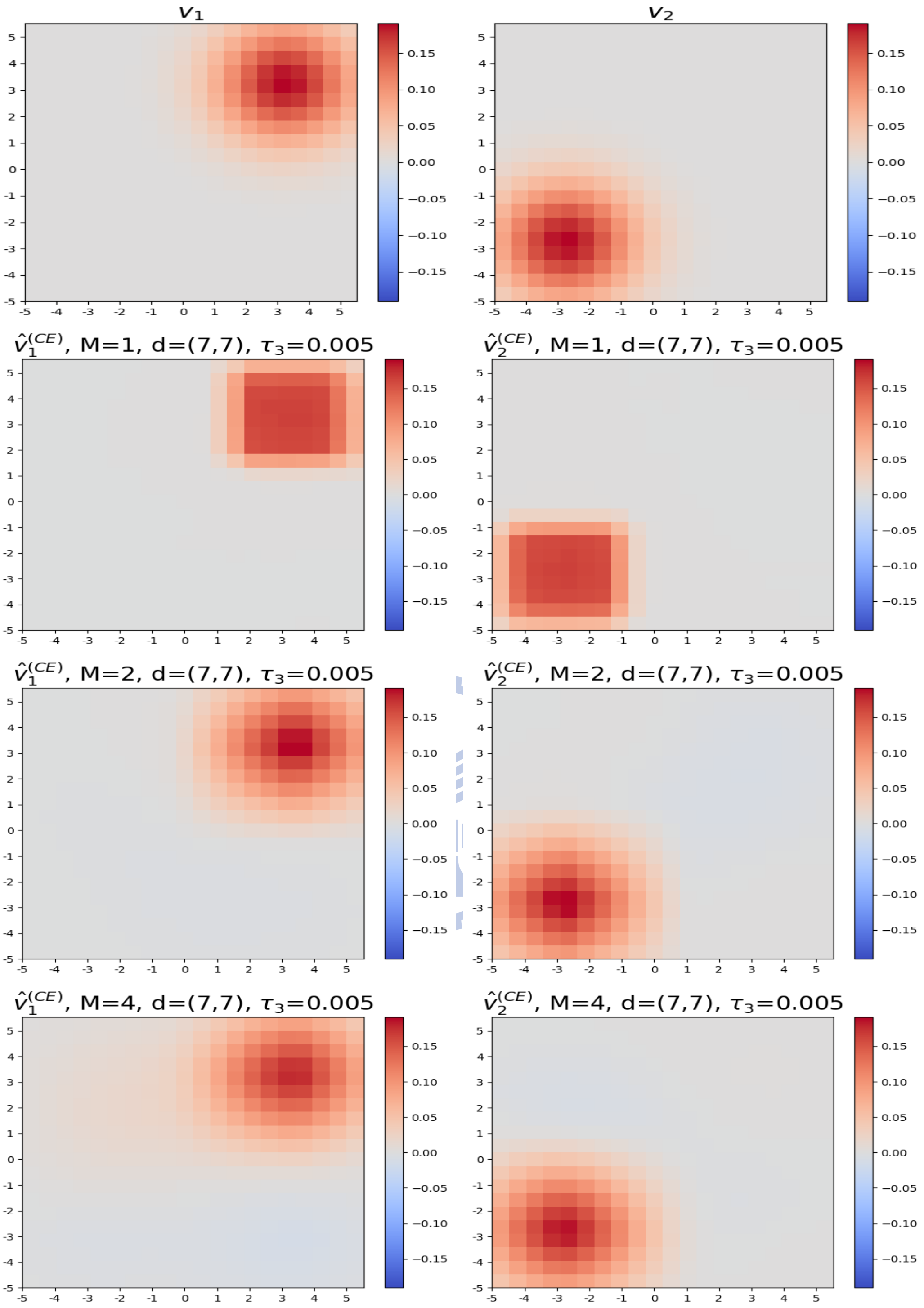


Figure 4.8: Effect of the number of convolutional layers (M): Estimated loading vectors for different numbers of convolutional layers (M) with fixed filter size $\boldsymbol{d} = (7, 7)$, sparse parameter $\tau_3 = 0.005$, and the number of principal components $K = 2$.

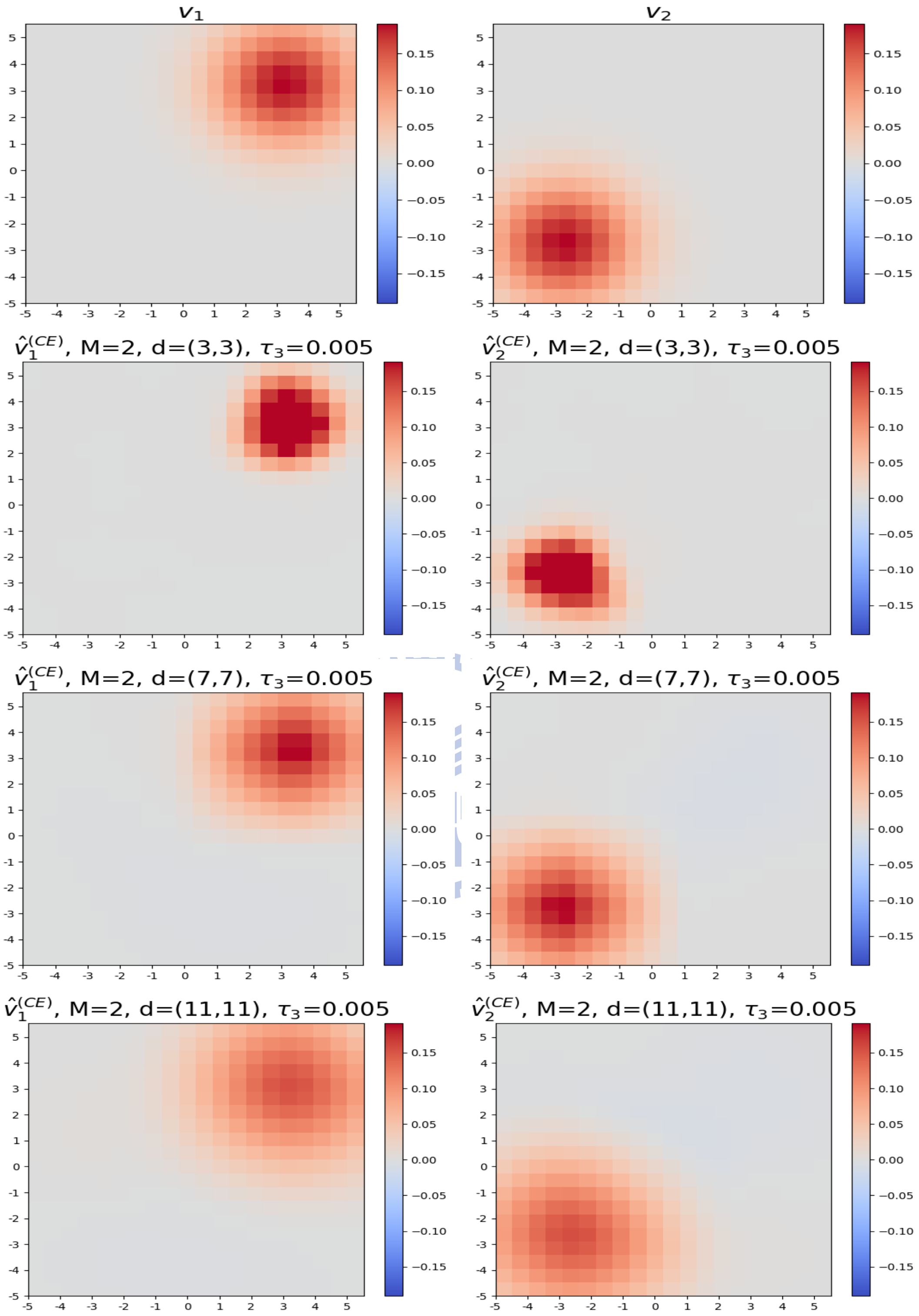


Figure 4.9: Effect of filter size ($\boldsymbol{d}$): Estimated loading vectors for different filter size ($\boldsymbol{d}$) with fixed numbers of convolutional layers $M = 2$, sparse parameter $\tau_3 = 0.005$, and the number of principal components $K = 2$.

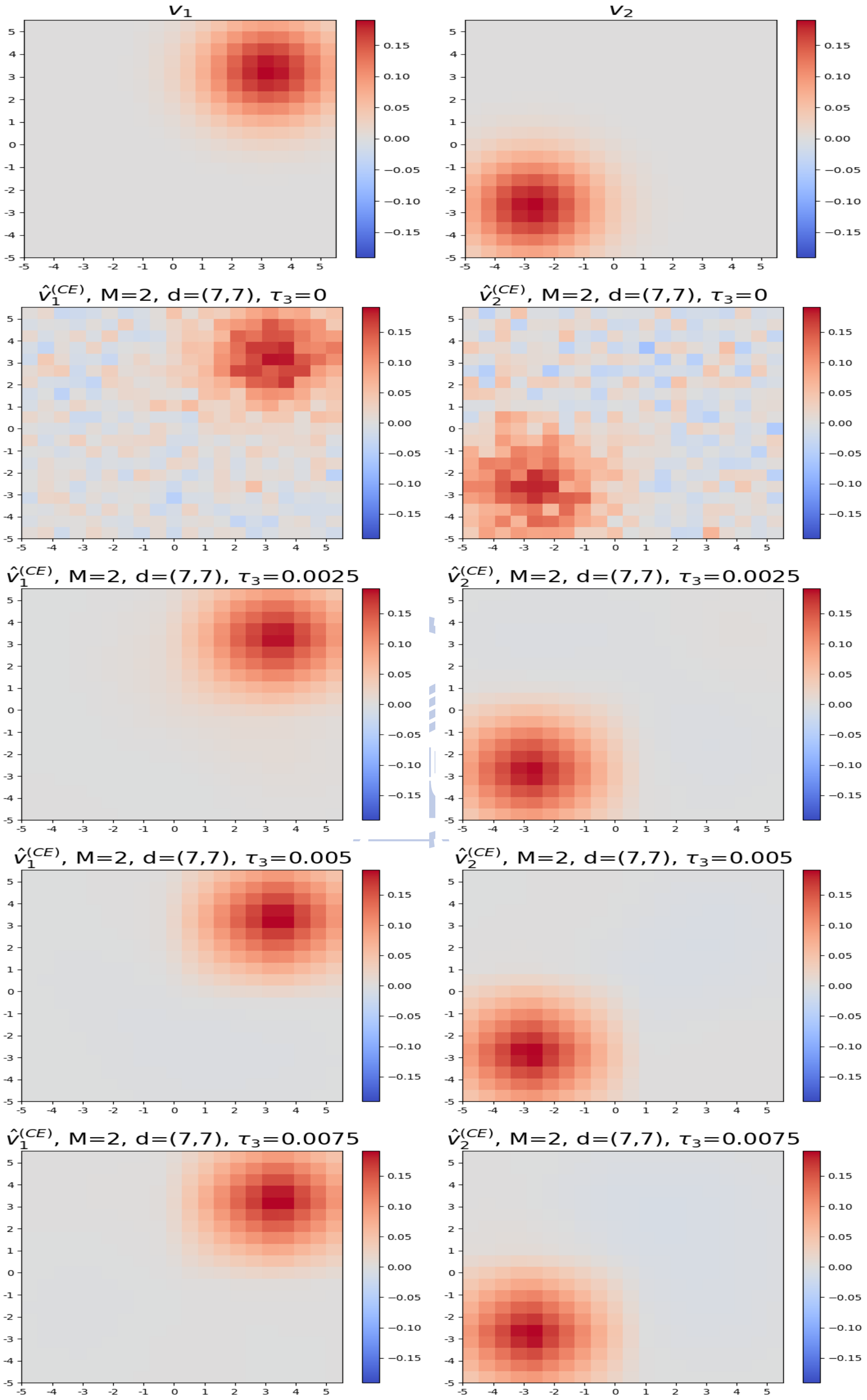
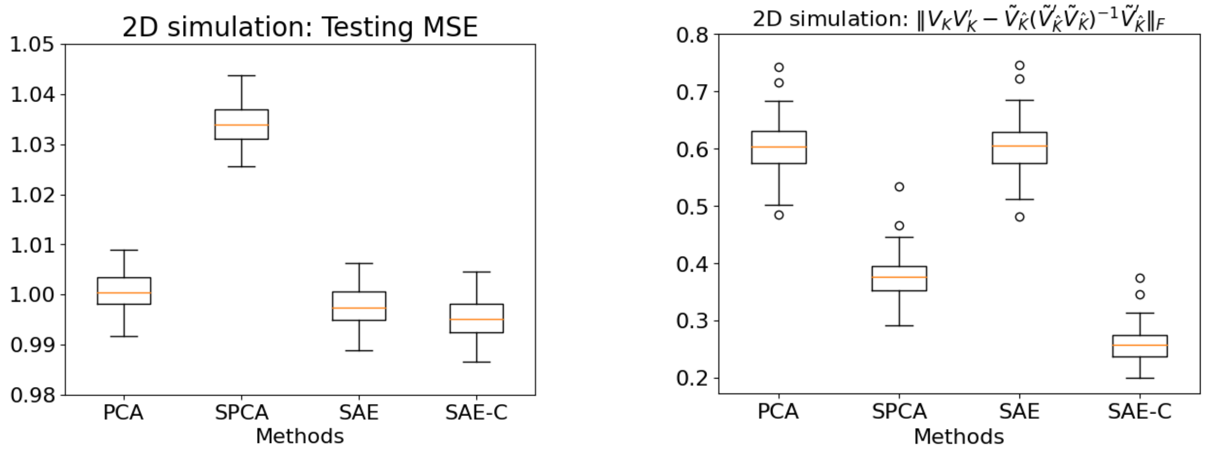


Figure 4.10: Effect of sparsity parameter (τ_3): Estimated loading vectors for different sparse parameter (τ_3) with fixed numbers of convolutional layers $M = 2$, filter size $\boldsymbol{d} = (7, 7)$, and the number of principal components $K = 2$.



(a)

(b)

Figure 4.11: Model performance for 2D data simulation: Testing MSE (4.4) and Frobenius norm of the difference between the true projection matrix and the estimated projection matrix (4.5) with 100 repeated simulations.

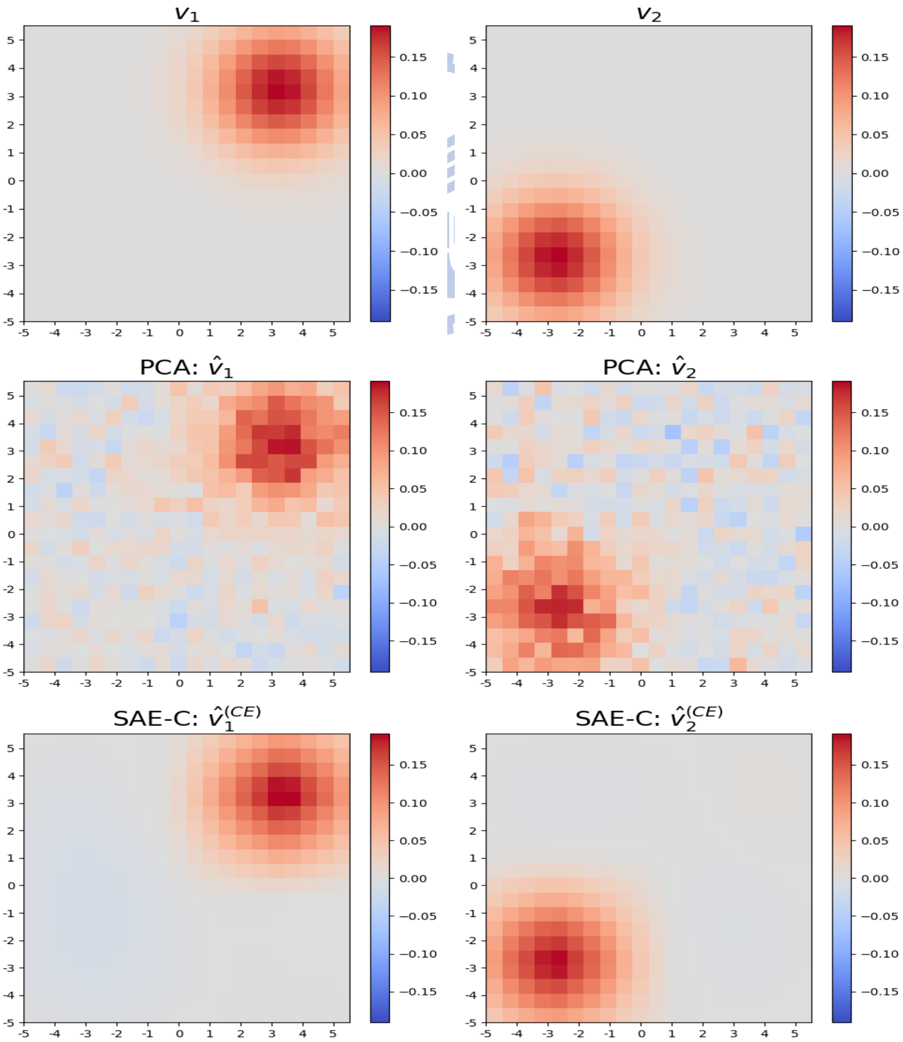


Figure 4.12: Estimated loading vectors for 2D data simulation using PCA and SAE-C.

4.3 Nonlinear PCA: Performance with and without Convolutional Layers

Using the simulation data $\boldsymbol{x}_i$'s from Chapter 4.2 with two linear spatial patterns (4.6) and (4.7), we conducted a case study of the NAE-C models with and without considering convolutional layers. The latter NAE-C model only considers fully connected layers f_l's (2.5) with ReLU activation functions (2.6). Specifically, the encoder $f_{\mathrm{E}} : \mathbb{R}^p \to \mathbb{R}^K$ without considering convolutional layers is parametrized by $\boldsymbol{\theta}_{\mathrm{E}} = \{\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{M+1}, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{M+1}\}$ and the decoder $f_{\mathrm{D}} : \mathbb{R}^K \to \mathbb{R}^p$ is parametrized by $\boldsymbol{\theta}_{\mathrm{D}} = \{\boldsymbol{W}_{M+2}, \ldots, \boldsymbol{W}_{2M+2}, \boldsymbol{b}_{M+2}, \ldots, \boldsymbol{b}_{2M+2}\}$ without considering convolutional layers. The activation functions are ReLU for every layer except the last layer, where the activation function is Identity. The encoder and decoder consist of weight matrices $\boldsymbol{W}_m \in \mathbb{R}^{p_m \times p_{m-1}}$ for $m = 1, \ldots, 2M + 2$, and bias vectors $\boldsymbol{b}_m \in \mathbb{R}^{p_m}$ for $m = 1, \ldots, 2M + 2$. The dimensions p_j are defined as $p_j = p_{2M+2-j} = p - j\lfloor p/(M + 1)\rfloor$ for $j = 1, \ldots, M$, with $p_{M+1} = K$. The encoder, decoder are defined as follows:

$$\begin{aligned} f_{\mathrm{E}}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathrm{E}}) =& f_{M+1} \circ \cdots \circ f_2 \circ f_1(\boldsymbol{x}|\boldsymbol{\theta}_{\mathrm{E}}) = \boldsymbol{z}, \\ f_{\mathrm{D}}(\boldsymbol{z}|\boldsymbol{\theta}_{\mathrm{D}}) =& f_{2M+2} \circ \cdots \circ f_{M+3} \circ f_{M+2}(\boldsymbol{z}|\boldsymbol{\theta}_{\mathrm{D}}) = \hat{\boldsymbol{x}}, \end{aligned}$$

and $\hat{\boldsymbol{\theta}}_{\mathrm{E}}$ and $\hat{\boldsymbol{\theta}}_{\mathrm{D}}$ solve the optimization problem (3.13). Using the loading vector estimator mentioned in the previous section (3.16) and parameter tuning method:

$$(\hat{M}_{(\mathrm{n2})}, \hat{K}_{(\mathrm{n2})}) = \underset{(M,K)\in S_{(\mathrm{n2})}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i^{(\mathrm{v})} - f_{\mathrm{D}} \circ f_{\mathrm{E}}(\boldsymbol{x}_i^{(\mathrm{v})}|\hat{\boldsymbol{\theta}}_{\mathrm{E}}(M, K), \hat{\boldsymbol{\theta}}_{\mathrm{D}}(M, K)\|_2^2. \tag{4.8}$$

Thus, the estimated encoder and decoder parameters for the NAE-C method without convolutional layers are:

$$\hat{\boldsymbol{\theta}}_{\mathrm{E}}(\hat{M}_{(\mathrm{n2})}, \hat{K}_{(\mathrm{n2})}) \text{ and } \hat{\boldsymbol{\theta}}_{\mathrm{D}}(\hat{M}_{(\mathrm{n2})}, \hat{K}_{(\mathrm{n2})}). \tag{4.9}$$

The estimated loading vectors $\hat{\boldsymbol{v}}_i^{(\mathrm{nCE})}, i = 1, 2$ around the sample mean of the data (2.2), without convolutional layers (4.9) and with convolutional layers (3.15) are shown in Figure 4.13. The estimates show that the estimated loading vectors considering convolutional layers capture the spatial patterns more effectively. Testing MSE for NAE-C is defined as follows:

$$\frac{1}{n}\sum_{i=1}^{n} \|\boldsymbol{x}_i^{(\mathrm{t})} - f_{\mathrm{D}} \circ f_{\mathrm{E}}(\boldsymbol{x}_i^{(\mathrm{t})}|\hat{\boldsymbol{\theta}}_{\mathrm{E}}, \hat{\boldsymbol{\theta}}_{\mathrm{D}})\|_2^2, \tag{4.10}$$

where $\hat{\boldsymbol{\theta}}_{\mathrm{E}}$ and $\hat{\boldsymbol{\theta}}_{\mathrm{D}}$ are the estimated parameters for encoders and decoders, respectively. The candidate set for NAE-C method with considering convolutional layers (3.14) is

$$S_{(\mathrm{n})} = \{(M, \boldsymbol{d}, K) | M = 1, 2, 4, 7, \boldsymbol{d} = (3, 1), (7, 1), (11, 1), K = 2, 3, 4\},$$

and the candidate set for NAE-C method without considering convolutional layers (4.8) is

$$S_{(\mathrm{n2})} = \{(M, K) | M = 0, 1, 2, 3, K = 2, 3, 4\},$$

We replicated the simulation 50 times to ensure the robustness of our results, with each replication taking approximately 5 minutes for NAE-C with considering convolutional layers and 1 minute for NAE-C without considering convolutional layers on a Google Colab Tesla A100 GPU. Training was performed with early stopping if the loss did not change by at least 0.01 within 100 epochs, with a maximum of 500 epochs allowed. Testing MSE for PCA, NAE-C with considering convolutional layers and testing MSE for NAE-C without considering convolutional layers are shown in Figure 4.14. The nonlinear methods could be easily expected to perform worse than PCA in terms of testing MSE due to the inherent linearity of the simulation data. Interestingly, our experiments also revealed that the NAE-C method without considering convolutional layers performs even worse than NAE-C with convolutional layers. This demonstrates the effectiveness of convolutional layers in capturing spatial information, highlighting their importance in enhancing model performance. Future work could focus on simulating nonlinear data and comparing our model with PCA or other linear methods to further validate the advantages of incorporating convolutional layers in capturing complex spatial patterns.

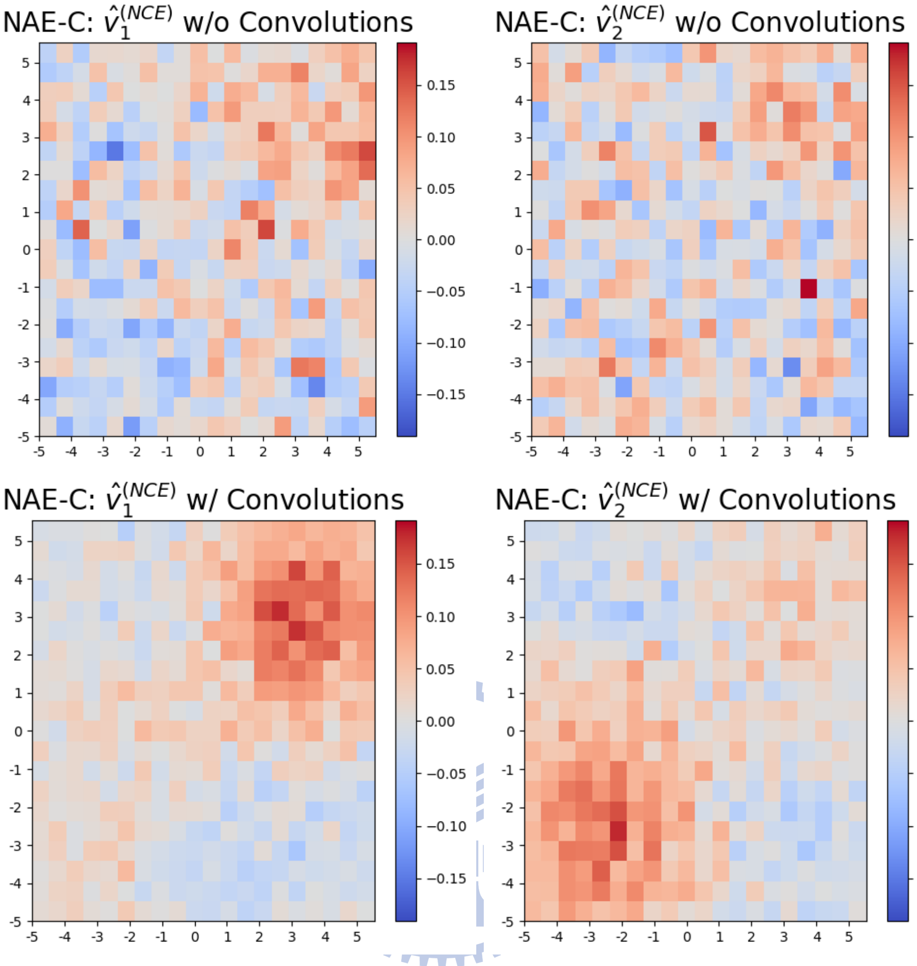


Figure 4.13: Nonlinear PCA: Loading vector estimates $\hat{\boldsymbol{v}}_i^{(\mathrm{nCE})}, i = 1, 2$ with and without convolutional layers.

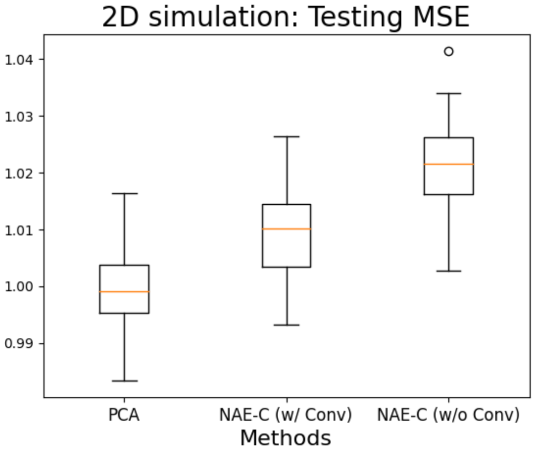


Figure 4.14: Model performance for 2D data simulation with nonlinear methods: Testing MSE (4.10) with 100 repeated simulations.

Chapter 5

Real Data Analysis

Sea Surface Temperature Data Analysis

Analyzing the latent patterns of sea surface temperature (SST) is crucial for understanding the underlying atmospheric mechanisms. The data used in this analysis, consisting of 216 monthly SST images, can be found at NEO: NASA Earth Observation. Specific patterns, such as those over the equatorial Pacific Ocean, are closely related to phenomena like El Niño. In this section, we employ the proposed spatial PCA method, SAE-C, to extract patterns from monthly SST data over the equatorial Pacific Ocean spanning from January 2003 to December 2020.

These 216 monthly SST data over the equatorial Pacific Ocean spans from 20°S to 20°N and 90°W to 170°W at one-degree intervals, theoretically resulting in $40 \times 80 = 3200$ grid locations. However, after excluding land areas, we are left with 3154 locations. To implement the convolution on the data, we use the K nearest neighbors method to reconstruct the missing values for the land areas, using 3 neighbors. Consequently, the dimensionality for the equatorial Pacific Ocean data is 3200. Sample mean and sample variance are shown in Figure 5.1 and monthly SST images over the equatorial Pacific Ocean from January 2003 to June 2007 with interval 6 months can be observed in Figure 5.2.

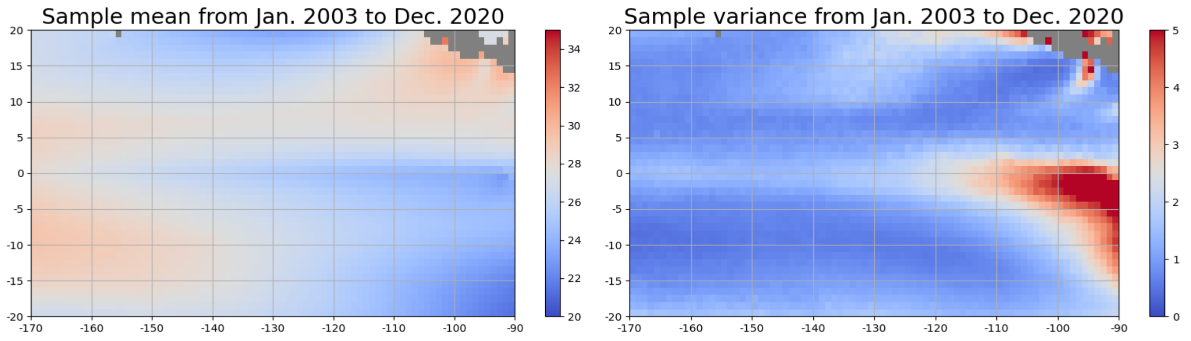


Figure 5.1: Sample mean and sample variance of monthly Sea Surface Temperature (SST) images over the equatorial Pacific Ocean from Jan. 2003 to Dec. 2020.

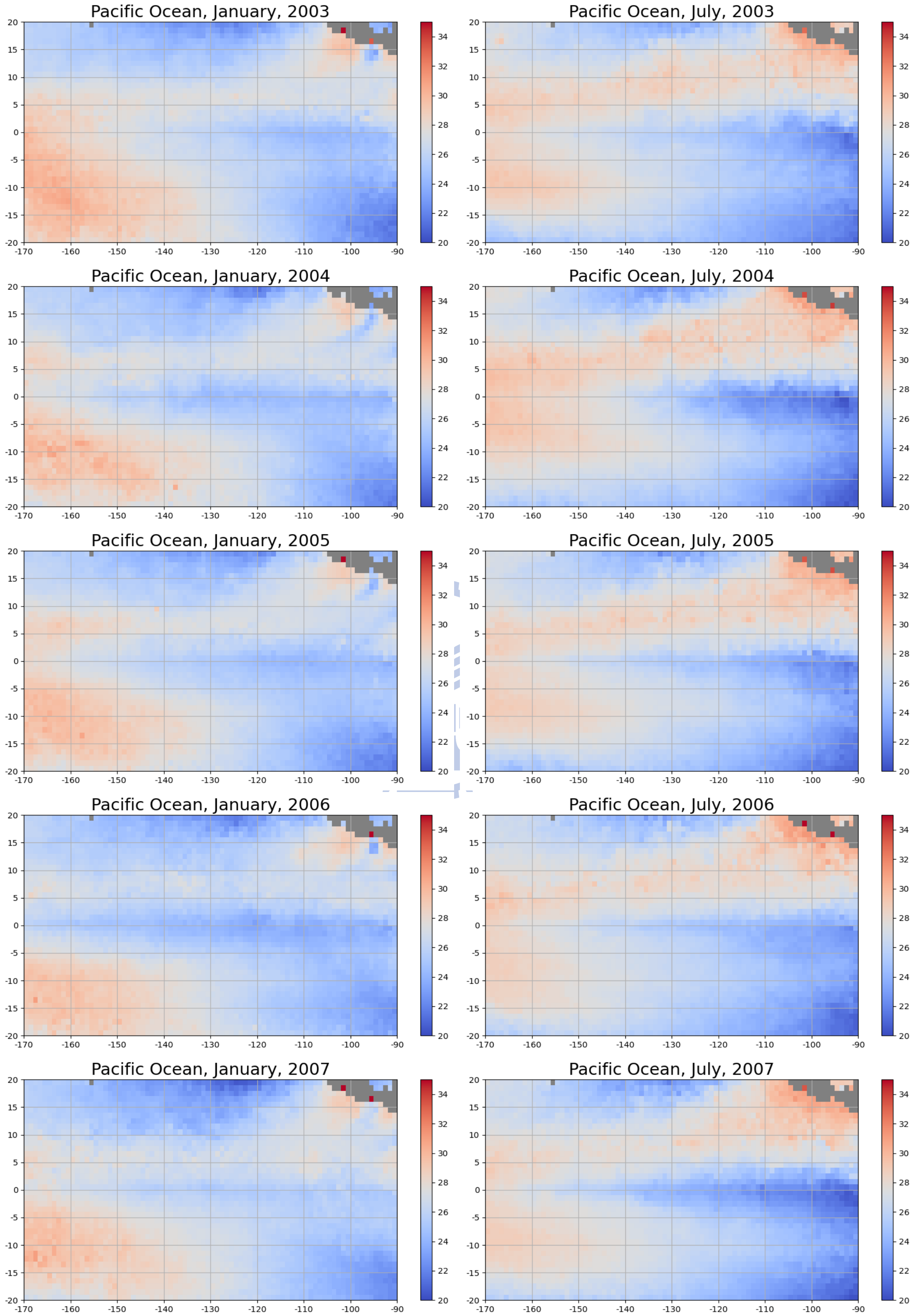

Pacific Ocean, January, 2003
Pacific Ocean, July, 2003
Pacific Ocean, January, 2004
Pacific Ocean, July, 2004
Pacific Ocean, January, 2005
Pacific Ocean, July, 2005
Pacific Ocean, January, 2006
Pacific Ocean, July, 2006
Pacific Ocean, January, 2007
Pacific Ocean, July, 2007
20
15
10
5
0
-5
-10
-15
-20
-170
-160
-150
-140
-130
-120
-110
-100
-90
34
32
30
28
26
24
22
20

Figure 5.2: Monthly SST images over the equatorial Pacific Ocean from January 2003 to June 2007 with interval 6 months.

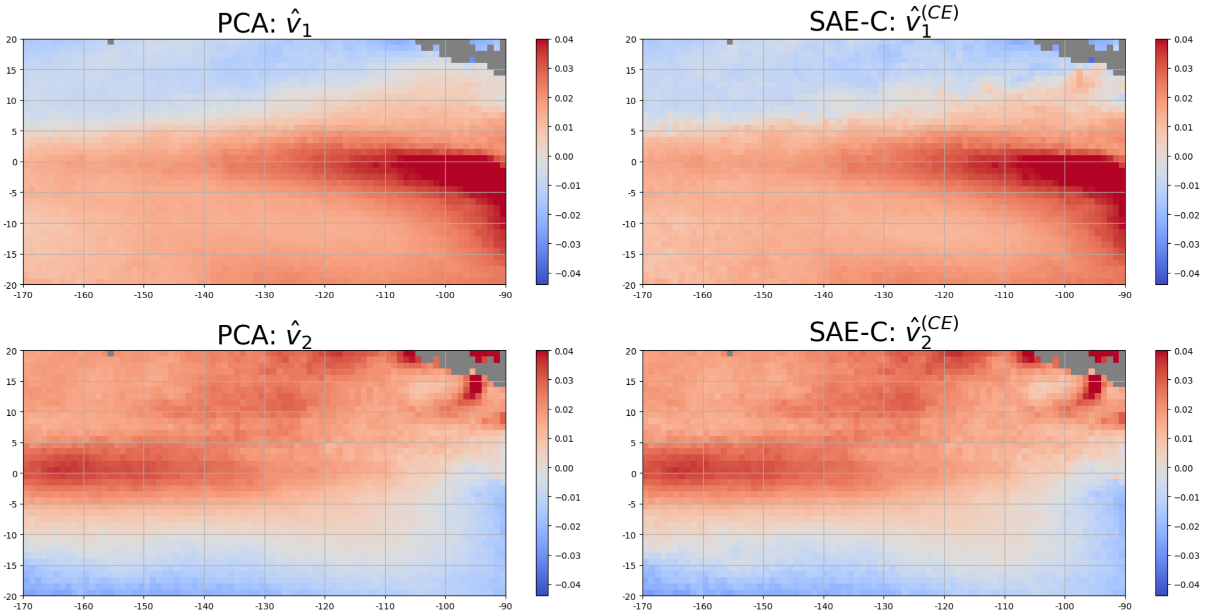


Figure 5.3: First two loading vectors for SST data over the equatorial Pacific Ocean using PCA and SAE-C.

We randomly partitioned the images into 50 training images $\boldsymbol{X} \in \mathbb{R}^{50\times 3200}$, 66 validation images $\boldsymbol{X}_{\text{valid}} \in \mathbb{R}^{66\times 3200}$, and 100 testing images $\boldsymbol{X}_{\text{test}} \in \mathbb{R}^{100\times 3200}$. Given these data, we implemented PCA (2.3), SAE-C (3.7), and NAE-C (3.13) to obtain $\hat{\boldsymbol{V}}_K$, $\hat{\boldsymbol{V}}_K^{(\text{CE})}(M,d,\tau_3)$, and $\hat{\boldsymbol{V}}_K^{(\text{NCE})}(M,d)$, respectively. We used (4.2), (3.8), and (3.14) to select optimal tuning parameters. The candidate sets for tuning parameters for PCA, SAE-C, and NAE-C were:

$$\begin{aligned} S_0 =& \{K | K = 2,3,\ldots,18\},\\ S_3 =& \{(M,d,\tau_3,K) | M = 1,2,4, d = 11,15,19, \tau_3 = 0, 0.001, 0.002, \ldots, 0.01,\\ &K = 2,\ldots,18\},\\ S_{(\text{n})} =& \{(M,d,K) | M = 1,2,4,7, d = 11,15,19, K = 2,\ldots,18\}, \end{aligned}$$

respectively. The first two estimated patterns for the dataset are illustrated in Figure 5.3.

For the equatorial Pacific Ocean SST data, the testing MSE (4.4) was 0.1452 for PCA, 0.1413 for SAE-C, and 0.2825 for NAE-C (4.10), indicating that SAE-C provided a slight improvement over PCA. For the equatorial Pacific Ocean SST data, our chosen parameters were $(\hat{M},\hat{d},\hat{\tau}_3,\hat{K}_3) = (1,11,0,18)$ for SAE-C, and $(\hat{M}_{(\text{n})},\hat{d}_{(\text{n})},\hat{K}_{(\text{n})}) = (1,19,14)$ for NAE-C. The

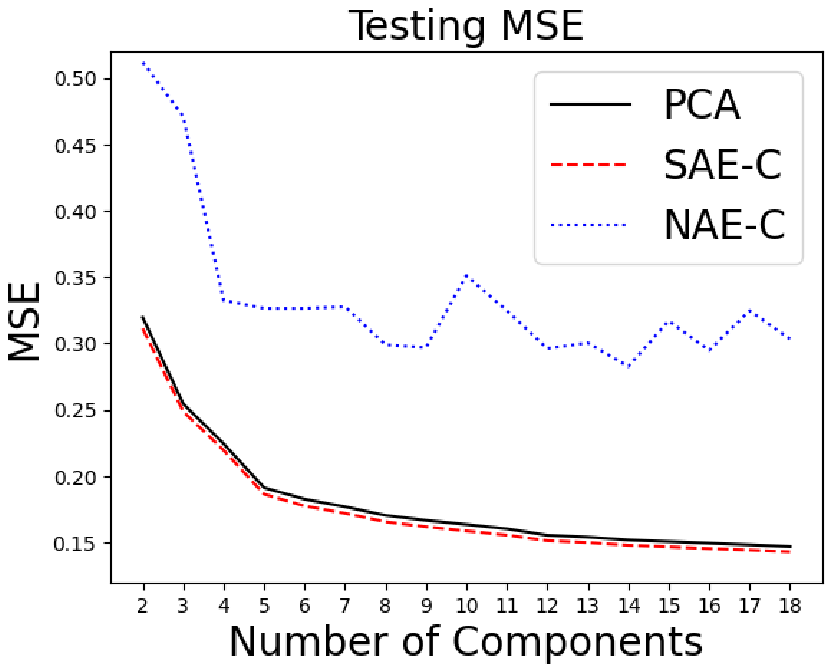


Figure 5.4: Testing MSE vs. number of principal components for SST data over the equatorial Pacific Ocean using PCA, SAE-C, and NAE-C.

comparisons of testing MSE between PCA, SAE-C, and NAE-C, varying with the number of principal components, for the equatorial Pacific Ocean SST data are shown in Figure 5.4, demonstrating that the proposed SAE-C method outperforms the traditional PCA method.

Chapter 6

Conclusion

This study proposes an innovative approach combining CNNs and autoencoders to achieve spatial SPCA. Our proposed SAE-C model, demonstrates superior performance in capturing essential features from spatial data, as evidenced by the results of our simulations and experiments.

The proposed SAE-C model represents a significant advancement in SPCA, effectively integrating convolutional layers to capture spatial patterns in high-dimensional data. Our approach outperforms traditional methods, offering improved accuracy, smoother estimations, and enhanced interpretability. The findings affirm the robustness and efficacy of incorporating convolutional layers in autoencoders for spatial data analysis, paving the way for future research and applications in various fields requiring detailed and accurate dimensionality reduction techniques.

6.1 Challenges and Future Work

While our SAE-C model performed well in simulations, certain challenges remain. The inherent lack of sparse patterns in some datasets, such as the SST data, posed difficulties in visually demonstrating the concept of sparsity. Future work could explore adaptive mechanisms for sparsity promotion, tailored to the specific characteristics of different datasets.

Moreover, the selection of tuning parameters, including the number of convolutional layers and the size of convolutional kernels, remains crucial for optimal model performance. Further research could investigate other automated or data-driven approaches for parameter selection to enhance the model's adaptability and robustness.

Nonlinear methods (Fukami et al. [2020]) also present a promising direction for future research. The current SAE-C model primarily focuses on linear transformations to estimate loading vectors. Exploring nonlinear approaches could potentially capture more complex spatial patterns in the data. However, estimating loading vectors in a nonlinear framework presents significant challenges. Traditional methods like Taylor series expansions might offer some solutions, but they often compromise the inherent nonlinearity of the model. Future work should aim to develop new techniques for estimating loading vectors that fully leverage the nonlinear capabilities of advanced neural network architectures, without resorting to linear approximations.

Additionally, future research could investigate the use of autoencoders with convolutions for nonlinear canonical correlation analysis. Due to time constraints, we were unable to complete simulations using this method, but we believe it holds significant potential. A detailed exploration of this approach is provided in the Chapter 6.2, highlighting its promise for enhancing the model's ability to capture complex patterns in the data. This offers a promising avenue for continued development and refinement of the SAE-C model.

6.2 Canonical Correlation Analysis

Canonical Correlation Analysis (CCA) (Guo and Wu [2019]) is a statistical method used to explore relationships between two sets of variables. It identifies linear combinations of variables from two datasets that are maximally correlated. Given two data matrices, $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)' \in \mathbb{R}^{n \times p}$ and $\boldsymbol{Y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)' \in \mathbb{R}^{n \times q}$, where n represents the number of observations and p and q are the numbers of variables respectively. The the mean-centered matrices are:

$$\tilde{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{1}_n \bar{\boldsymbol{x}}', \quad \tilde{\boldsymbol{Y}} = \boldsymbol{Y} - \boldsymbol{1}_n \bar{\boldsymbol{y}}',$$

where

$$\bar{\boldsymbol{x}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{x}_i, \quad \bar{\boldsymbol{y}} = \frac{1}{n} \sum_{i=1}^{n} \boldsymbol{y}_i,$$

and $\boldsymbol{1}_n$ is an $n \times 1$ vector with all elements equal to 1.

CCA seeks to solve the following optimization problem:

$$(\hat{\boldsymbol{w}}, \hat{\boldsymbol{v}}) = \arg\max_{(\boldsymbol{w},\boldsymbol{v})} \frac{\boldsymbol{w}'\tilde{\boldsymbol{X}}'\tilde{\boldsymbol{Y}}\boldsymbol{v}}{\sqrt{(\boldsymbol{w}'\boldsymbol{S}_X\boldsymbol{w})(\boldsymbol{v}'\boldsymbol{S}_Y\boldsymbol{v})}},$$

where $\boldsymbol{w} \in \mathbb{R}^p$ and $\boldsymbol{v} \in \mathbb{R}^q$ are weight vectors, and $\boldsymbol{S}_X$ and $\boldsymbol{S}_Y$ are the sample covariance matrices of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. The estimated vectors, $\hat{\boldsymbol{w}}$ and $\hat{\boldsymbol{v}}$, are called canonical loading vectors. The canonical scores for $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ are given by $\hat{\boldsymbol{w}}'(\boldsymbol{x}_i - \bar{\boldsymbol{x}}) = z_{X,i}$ and $\hat{\boldsymbol{v}}'(\boldsymbol{y}_i - \bar{\boldsymbol{y}}) = z_{Y,i}$, respectively.

6.2.1 Deep Canonical Correlation Analysis

Neural Networks for CCA

Deep Canonical Correlation Analysis (DCCA) (Benton et al. [2017]) extends the classical Canonical Correlation Analysis (CCA) by leveraging the power of neural networks to capture complex, non-linear relationships betwen two sets of variables. Traditional CCA is limited to linear transformations, which may not be sufficient for capturing intricate patterns in high-dimensional data. DCCA addresses this limitation by using neural networks to learn more flexible and powerful representations of the data.

DCCA employs two neural networks, one for each dataset, to transform the data into a lower-dimensional space where the transformed representations are maximally correlated. Given two datasets, $\boldsymbol{X}$ and $\boldsymbol{Y}$, represented by $f^X_{\text{NN}} : \mathbb{R}^p \to \mathbb{R}^{p'}$ parametrized by $\boldsymbol{\theta}_X$ and $f^Y_{\text{NN}} : \mathbb{R}^q \to \mathbb{R}^{q'}$ parametrized by $\boldsymbol{\theta}_Y$, the goal is to find transformation vectors $\boldsymbol{w}_X$ and $\boldsymbol{w}_Y$ that maximize the correlation between the outputs of the neural networks.

The trainable parameters $(\boldsymbol{\theta}_X, \boldsymbol{\theta}_Y)$ include weight matrices and bias vectors are for the neural networks f^X_{NN} and f^Y_{NN}, respectively. The objective is to optimize these parameters for the neural networks and transformation vectors, $\boldsymbol{w}_X \in \mathbb{R}^{p'}$ and $\boldsymbol{w}_Y \in \mathbb{R}^{q'}$, to achieve maximum correlation between the transformed outputs $\boldsymbol{w}'_X f^X_{\text{NN}}(\boldsymbol{x}_i|\boldsymbol{\theta}_X)$'s and $\boldsymbol{w}'_Y f^Y_{\text{NN}}(\boldsymbol{y}_i|\boldsymbol{\theta}_Y)$'s. This is achieved by minimizing the following loss function:

$$L_{\text{DCCA}}(\boldsymbol{\theta}_X, \boldsymbol{\theta}_Y, \boldsymbol{w}_X, \boldsymbol{w}_Y) = -\frac{\sum_{i=1}^{n} \boldsymbol{w}'_X f^X_{\text{NN}}(\boldsymbol{x}_i|\boldsymbol{\theta}_X) \boldsymbol{w}'_Y f^Y_{\text{NN}}(\boldsymbol{y}_i|\boldsymbol{\theta}_Y)}{\{\sum_{i=1}^{n}(\boldsymbol{w}'_X f^X_{\text{NN}}(\boldsymbol{x}_i|\boldsymbol{\theta}_X))^2\}^{1/2} \cdot \{\sum_{i=1}^{n}(\boldsymbol{w}'_Y f^Y_{\text{NN}}(\boldsymbol{y}_i|\boldsymbol{\theta}_Y))^2\}^{1/2}}. \tag{6.1}$$

By minimizing this loss, the neural networks learn to extract and correlate complex features from both datasets effectively. The optimized parameters $(\hat{\boldsymbol{\theta}}_X, \hat{\boldsymbol{\theta}}_Y, \hat{\boldsymbol{w}}_X, \hat{\boldsymbol{w}}_Y)$ result in deep canonical scores $\hat{\boldsymbol{w}}'_X f^X_{\text{NN}}(\boldsymbol{x}_i|\hat{\boldsymbol{\theta}}_X)$ and $\hat{\boldsymbol{w}}'_Y f^Y_{\text{NN}}(\boldsymbol{y}_i|\hat{\boldsymbol{\theta}}_Y)$ for each data point, $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$, respectively.

This approach enhances the capability of CCA by allowing it to uncover non-linear correlations, making it a powerful tool for analyzing complex, high-dimensional data where linear methods fall short.

Deep Canonically Correlated Autoencoders

Inspired by both CCA and reconstruction-based objectives, Wang et al. [2015] proposes a model that includes two autoencoders and optimizes the combination of canonical correlation between the learned code representations and the reconstruction loss of the autoencoders. These models are called Deep Cananically Correlated AutoEncoders (DCCAE).

The encoders, $f^X_{\text{E}} : \mathbb{R}^p \to \mathbb{R}^{d_1}$ parametrized by $\boldsymbol{\theta}^X_{\text{E}}$ and $f^Y_{\text{E}} : \mathbb{R}^q \to \mathbb{R}^{d_2}$ parametrized by $\boldsymbol{\theta}^Y_{\text{E}}$ are for $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. The decoders, $f^X_{\text{D}} : \mathbb{R}^{d_1} \to \mathbb{R}^p$ parametrized by $\boldsymbol{\theta}^X_{\text{D}}$ and $f^Y_{\text{D}} : \mathbb{R}^{d_2} \to \mathbb{R}^q$ parametrized by $\boldsymbol{\theta}^Y_{\text{D}}$ reconstruct the data. Each autoencoder, $f^X_{AE} = f^X_{\text{D}} \circ f^X_{\text{E}} : \mathbb{R}^p \to \mathbb{R}^p$ and $f^Y_{AE} = f^Y_{\text{D}} \circ f^Y_{\text{E}} : \mathbb{R}^q \to \mathbb{R}^q$, has trainable parameters $\boldsymbol{\theta}^X_{\text{E}}, \boldsymbol{\theta}^X_{\text{D}}, \boldsymbol{\theta}^Y_{\text{E}}, \boldsymbol{\theta}^Y_{\text{D}}$.

The transformation vectors $\boldsymbol{w}_X \in \mathbb{R}^{d_1}$ and $\boldsymbol{w}_Y \in \mathbb{R}^{d_2}$ are applied to the outputs of the encoders, $f^X_{\text{E}}(\boldsymbol{x}|\boldsymbol{\theta}^X_{\text{E}})$ and $f^Y_{\text{E}}(\boldsymbol{y}|\boldsymbol{\theta}^Y_{\text{E}})$, to maximize correlation. The parameters and transformation vectors $(\hat{\boldsymbol{\theta}}^X_{\text{E}}, \hat{\boldsymbol{\theta}}^Y_{\text{E}}, \hat{\boldsymbol{\theta}}^X_{\text{D}}, \hat{\boldsymbol{\theta}}^Y_{\text{D}}, \tilde{\boldsymbol{w}}_X, \tilde{\boldsymbol{w}}_Y)$ minimize the following loss function, which incorporates the loss function of DCCA (6.1):

$$\begin{aligned} & L_{\text{DCCAE}}(\boldsymbol{\theta}_{\text{E}}^{X}, \boldsymbol{\theta}_{\text{E}}^{Y}, \boldsymbol{\theta}_{\text{D}}^{X}, \boldsymbol{\theta}_{\text{D}}^{Y}, \boldsymbol{w}_X, \boldsymbol{w}_Y) \\ &\quad = L_{\text{DCCA}}(\boldsymbol{\theta}_{\text{E}}^{X}, \boldsymbol{\theta}_{\text{E}}^{Y}, \boldsymbol{w}_X, \boldsymbol{w}_Y) + \lambda \left(\frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{x}_i - f_{AE}^{X}(\boldsymbol{x}_i | \boldsymbol{\theta}_{\text{E}}^{X}, \boldsymbol{\theta}_{\text{D}}^{X})\|_2 \right. \\ &\qquad \left. + \frac{1}{n} \sum_{i=1}^{n} \|\boldsymbol{y}_i - f_{AE}^{Y}(\boldsymbol{y}_i | \boldsymbol{\theta}_{\text{E}}^{Y}, \boldsymbol{\theta}_{\text{D}}^{Y})\|_2 \right), \end{aligned} \tag{6.2}$$

where λ balances the trade-off between maximizing the correlation and minimizing the reconstruction loss. The first part of the loss function, L_{DCCA}, aims to maximize the correlation between the outputs of the encoders and the transformation vectors, while the second part minimizes the difference between the original inputs and their reconstructions. The terms

$$\tilde{\boldsymbol{w}}'_X f_{\text{E}}^{X}(\boldsymbol{x}_i | \hat{\boldsymbol{\theta}}_{\text{E}}^{X}) \text{ and } \tilde{\boldsymbol{w}}'_Y f_{\text{E}}^{Y}(\boldsymbol{y}_i | \hat{\boldsymbol{\theta}}_{\text{E}}^{Y})$$

represent the deep correlated codes for inputs $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$, respectively.

This dual objective allows the model to extract codes that not only maximize correlation between the datasets but also preserve the original information within each dataset. When $\lambda = 0$, the decoder parameters ($\boldsymbol{\theta}_{\text{D}}^{X}$, $\boldsymbol{\theta}_{\text{D}}^{Y}$) are not trained. Therefore, the deep correlated codes $\tilde{\boldsymbol{w}}'_X f_{\text{E}}^{X}(\boldsymbol{x}_i | \hat{\boldsymbol{\theta}}_{\text{E}}^{X})$ and $\tilde{\boldsymbol{w}}'_Y f_{\text{E}}^{Y}(\boldsymbol{y}_i | \hat{\boldsymbol{\theta}}_{\text{E}}^{Y})$ will be the same as those obtained from optimizing the loss function in (6.1).

6.2.2 Spatial CCA Using Autoencoders

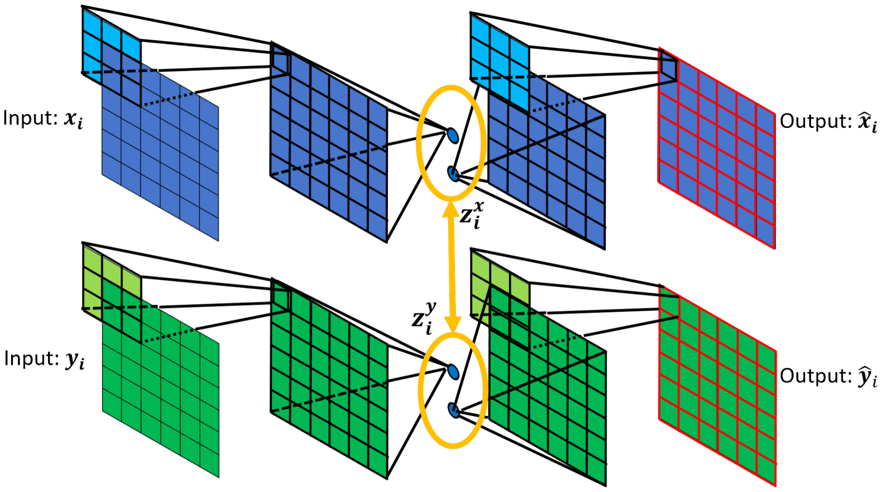

Input: $\boldsymbol{x}_i$
Output: $\hat{\boldsymbol{x}}_i$
$\boldsymbol{z}_i^x$
$\boldsymbol{z}_i^y$
Input: $\boldsymbol{y}_i$
Output: $\hat{\boldsymbol{y}}_i$

Figure 6.1: Model architecture of Deep Canonically Correlated AutoEncoders with Convolutions (DCCAE-C). Orange indicates the correlation loss, and red represents the reconstruction loss (Mean squared error). The architecture of the method includes parallel autoencoders with convolutions connected by the correlation loss between correlated codes ($\boldsymbol{z}_i^x$, $\boldsymbol{z}_i^y$).

To extend the framework of DCCAE (Chapter 6.2.1) to spatial data analysis, we introduce DCCAE with Convolutions (DCCAE-C). This variant incorporates convolutional layers to effectively handle data with inherent spatial structures, such as images or spatially correlated signals. Leveraging the localized feature extraction capabilities of CNNs, DCCAE-C enhances the performance of canonical correlation analysis in high-dimensional spatial domains.

As illustrated in Figure 6.1, the DCCAE-C model consists of two parallel autoencoders with convolutions, one for each dataset ($\boldsymbol{X}$ and $\boldsymbol{Y}$). These autoencoders are designed to maximize the correlation between their latent spaces while accurately reconstructing the original inputs, which contain spatial information. The encoders, $f_{\mathrm{E}}^X : \mathbb{R}^p \to \mathbb{R}^{d_1}$ parametrized by $\boldsymbol{\theta}_{\mathrm{E}}^X$ and $f_{\mathrm{E}}^Y : \mathbb{R}^q \to \mathbb{R}^{d_2}$ parametrized by $\boldsymbol{\theta}_{\mathrm{E}}^Y$ are for $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively. The decoders, $f_{\mathrm{D}}^X : \mathbb{R}^{d_1} \to \mathbb{R}^p$ parametrized by $\boldsymbol{\theta}_{\mathrm{D}}^X$ and $f_{\mathrm{D}}^Y : \mathbb{R}^{d_2} \to \mathbb{R}^q$ parametrized by $\boldsymbol{\theta}_{\mathrm{D}}^Y$ reconstruct the data.

Each autoencoder, $f_{AE}^X = f_{\mathrm{D}}^X \circ f_{\mathrm{E}}^X : \mathbb{R}^p \to \mathbb{R}^p$ and $f_{AE}^Y = f_{\mathrm{D}}^Y \circ f_{\mathrm{E}}^Y : \mathbb{R}^q \to \mathbb{R}^q$, has trainable parameters $\boldsymbol{\theta}_{\mathrm{E}}^{X}, \boldsymbol{\theta}_{\mathrm{D}}^{X}, \boldsymbol{\theta}_{\mathrm{E}}^{Y}, \boldsymbol{\theta}_{\mathrm{D}}^{Y}$, where

$$\begin{aligned}
\boldsymbol{\theta}_{\mathrm{E}}^{X} &= \{\boldsymbol{C}_{1}^{(X)}, ..., \boldsymbol{C}_{L}^{(X)}, \boldsymbol{B}_{1}^{(X)}, ..., \boldsymbol{B}_{L}^{(X)}\}, \\
\boldsymbol{\theta}_{\mathrm{D}}^{X} &= \{\boldsymbol{C}_{L+1}^{(X)}, ..., \boldsymbol{C}_{2L}^{(X)}, \boldsymbol{B}_{L+1}^{(X)}, ..., \boldsymbol{B}_{2L}^{(X)}\}, \\
\boldsymbol{\theta}_{\mathrm{E}}^{Y} &= \{\boldsymbol{C}_{1}^{(Y)}, ..., \boldsymbol{C}_{L}^{(Y)}, \boldsymbol{B}_{1}^{(Y)}, ..., \boldsymbol{B}_{L}^{(Y)}\}, \\
\boldsymbol{\theta}_{\mathrm{D}}^{Y} &= \{\boldsymbol{C}_{L+1}^{(Y)}, ..., \boldsymbol{C}_{2L}^{(Y)}, \boldsymbol{B}_{L+1}^{(Y)}, ..., \boldsymbol{B}_{2L}^{(Y)}\},
\end{aligned}$$

where $\boldsymbol{C}_{l}^{(X)}$ and $\boldsymbol{C}_{l}^{(Y)}$ are convolutional kernels trained on $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively, and $\boldsymbol{B}_{l}^{(X)}$ and $\boldsymbol{B}_{l}^{(Y)}$ are bias vectors trained on $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively.

The transformation vectors $\boldsymbol{w}_X \in \mathbb{R}^{d_1}$ and $\boldsymbol{w}_Y \in \mathbb{R}^{d_2}$ are applied to the outputs of the encoders, $f_{\mathrm{E}}^{X}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathrm{E}}^{X})$ and $f_{\mathrm{E}}^{Y}(\boldsymbol{y}|\boldsymbol{\theta}_{\mathrm{E}}^{Y})$, to maximize correlation. The parameters including $2L$ convolutional filters and bias matrices and two transformation vectors

$$(\hat{\boldsymbol{\theta}}_{\mathrm{E}}^{X}, \hat{\boldsymbol{\theta}}_{\mathrm{E}}^{Y}, \hat{\boldsymbol{\theta}}_{\mathrm{D}}^{X}, \hat{\boldsymbol{\theta}}_{\mathrm{D}}^{Y}, \tilde{\boldsymbol{w}}_X, \tilde{\boldsymbol{w}}_Y)$$

minimize the following loss function, which incorporates the loss function of DCCA (6.1). The loss function for DCCAE-C is the same as (6.2), but the fully connected layers are replaced by convolutional layers (3.3). The terms

$$\tilde{\boldsymbol{w}}_X' f_{\mathrm{E}}^{X}(\boldsymbol{x}_i|\hat{\boldsymbol{\theta}}_{\mathrm{E}}^{X}) \text{ and } \tilde{\boldsymbol{w}}_Y' f_{\mathrm{E}}^{Y}(\boldsymbol{y}_i|\hat{\boldsymbol{\theta}}_{\mathrm{E}}^{Y})$$

represent the deep correlated codes for inputs $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$, respectively.

The training of the DCCAE-C model involves gradient-based optimization techniques. This process alternates between optimizing the weights of the convolutional autoencoders to improve reconstruction and adjusting the parameters to maximize the sample correlation between the codes. By pursuing these dual objectives, the model captures the most correlated features between the datasets while maintaining high-fidelity data reconstruction. Notably, our method retains spatial information by incorporating convolutional layers, which are specifically designed to handle spatial dependencies in the data. The tuning parameters in this case are also the dimension of codes and the size of the convolutional kernels. Following the same manner of hyperparameter tuning as (3.8), we use validation data for parameter tuning.

Bibliography

Aman Agrawal, Alec M Chiu, Minh Le, Eran Halperin, and Sriram Sankararaman. Scalable probabilistic pca for large-scale genetic variation data. *PLoS genetics*, 16:e1008773, 2020.

Adrian Benton, Huda Khayrallah, Biman Gujral, Dee Ann Reisinger, Sheng Zhang, and Raman Arora. Deep generalized canonical correlation analysis. *arXiv preprint arXiv:1702.02519*, 2017.

Edwin KP Chong and Stanislaw H Żak. *An introduction to optimization*. John Wiley & Sons, 2013.

Kai Fukami, Taichi Nakamura, and Koji Fukagata. Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data. *Physics of Fluids*, 32, 2020.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.

Chenfeng Guo and Dongrui Wu. Canonical correlation analysis (cca) based multi-view learning: An overview. *arXiv preprint arXiv:1907.01693*, 2019.

Ian Jolliffe. Principal component analysis. *Encyclopedia of statistics in behavioral science*, 2005.

Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 33:6999–7019, 2021.

Umberto Michelucci. An introduction to autoencoders. *arXiv preprint arXiv:2201.03898*, 2022.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58:267–288, 1996.

KJ Waldron, Shih-Liang Wang, and SJ Bolin. A study of the jacobian matrix of serial manipulators. *Journal of Mechanisms, Transmissions, and Automation in Design*, 107:230–237, 1985.

Weiran Wang, Raman Arora, Karen Livescu, and Jeff Bilmes. On deep multi-view representation learning. In *International conference on machine learning*, pages 1083–1092. PMLR, 2015.

Daniel S Wilks. *Statistical methods in the atmospheric sciences*. Academic press, 2011.

Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9:611–629, 2018.

Junhai Zhai, Sufang Zhang, Junfen Chen, and Qiang He. Autoencoder and its various variants. In *2018 IEEE international conference on systems, man, and cybernetics (SMC)*, pages 415–419. IEEE, 2018.

Hui Zou, Trevor Hastie, and Robert Tibshirani. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15:265–286, 2006.