Deep Learning One-Class Classification With Support Vector Methods

Hayden D. Hampton

University of Central Florida
DEEP LEARNING ONE-CLASS CLASSIFICATION WITH SUPPORT VECTOR METHODS

by

HAYDEN DOUGLAS HAMPTON
B.S. University of Central Florida, 2019
M.S. University of Central Florida, 2021

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Statistics & Data Science
in the College of Sciences
at the University of Central Florida
Orlando, Florida

Spring Term
2024

Major Professor: Edgard M. Maboudou-Tchao
Through the specialized lens of one-class classification, anomalies—irregular observations that uncharacteristically diverge from normative data patterns—are comprehensively studied. This dissertation focuses on advancing boundary-based methods in one-class classification, a critical approach to anomaly detection. These methodologies delineate optimal decision boundaries, thereby facilitating a distinct separation between normal and anomalous observations. Encompassing traditional approaches such as One-Class Support Vector Machine and Support Vector Data Description, recent adaptations in deep learning offer a rich ground for innovation in anomaly detection. This dissertation proposes three novel deep learning methods for one-class classification, aiming to enhance the efficacy and accuracy of anomaly detection in an era where data volume and complexity present unprecedented challenges. The first two methods are designed for tabular data from a least squares perspective. Formulating these optimization problems within a least squares framework offers notable advantages. It facilitates the derivation of closed-form solutions for critical gradients that largely influence the optimization procedure. Moreover, this approach circumvents the prevalent issue of degenerate or uninformative solutions, a challenge often associated with these types of deep learning algorithms. The third method is designed for second-order tensors. This proposed method has certain computational advantages and alleviates the need for vectorization, which can lead to structural information loss when spatial or contextual relationships exist in the data structure. The performance of the three proposed methods are demonstrated with simulation studies and real-world datasets. Compared to kernel-based one-class classification methods, the proposed deep learning methods achieve significantly better performance under the settings considered.
This dissertation is dedicated to my grandfather, Harold Leroy Friedman (1924-2014).
ACKNOWLEDGMENTS

To my parents, Helen and Howie, thank you for your endless support and encouragement. I could not have achieved this without the both of you. My heartfelt thanks to my grandmother Eleanor for your unwavering belief in me and always recognizing my potential. I know in my heart that if my grandfather could see me now, he would be beaming with pride at what I’ve achieved. To my sister Holly, now that we both have embraced the title of doctor, I want to express how invaluable your guidance has been to me. Despite our differing passions, your support has been a fundamental part of my journey. To Uncle Mark, your support and guidance have been instrumental in my journey. The understanding we share has empowered you to offer just the right words at the right time, whether they were comforting or exactly what I needed to hear to realign my path. It’s hard to imagine reaching this point without your unwavering encouragement.

I want to extend my deepest gratitude to a number of individuals whose support has been pivotal in my academic journey. First and foremost, I wish to express my sincere thanks to Dr. Edgard M. Maboudou-Tchao for his invaluable guidance and encouragement. Your mentorship has played a pivotal role in both my professional and personal development throughout this journey. I am also profoundly thankful to my department for providing me with the funding opportunity that made it possible for me to pursue my Ph.D. degree. A special acknowledgement goes to Dr. Hsin-Hsiung Huang for his generosity in supporting me as a Research Assistant during the summer semesters. His pivotal mentorship was instrumental in shaping my academic and research interests, especially in fostering my enthusiasm for Bayesian and spatio-temporal data analysis. Additionally, I would like to extend my heartfelt appreciation to two past mentors, Dr. David M. Nickerson and Professor Joseph R. Pick. Their encouragement, patience, and eagerness to nurture an inquisitive mind has left an indelible mark on my academic career.
My friend and colleague, Michael Hopwood, deserves special recognition for our enriching discussions and collaborations in exploring new research directions over the course of my graduate studies. This journey was made all the more fulfilling with your friendship and camaraderie. To my research collaborator and lab mate, Emil Agbemade, working with you has been an enriching experience and an absolute pleasure. Additionally, I would like to acknowledge my late friend and fellow classmate, Eric Feit, who constantly challenged me to always put forward my very best work. His humility and virtue taught me the valuable lesson of speaking with purpose, recognizing that sometimes, less is indeed more.
# TABLE OF CONTENTS

LIST OF FIGURES .................................................. xiv

CHAPTER 1: INTRODUCTION ........................................ 1

CHAPTER 2: LITERATURE REVIEW .............................. 4

One-Class Classification ........................................ 4

Kernel-Based Support Vector Methods ......................... 7

One-Class Support Vector Machine ............................. 7

Least Squares One-Class Support Vector Machine .......... 12

Support Vector Data Description .............................. 15

Least Squares Support Vector Data Description ............. 20

Support Matrix Data Description .............................. 25

Deep Learning-Based Support Vector Methods ............... 33

One-Class Neural Network ....................................... 33

Deep Support Vector Data Description ....................... 34

CHAPTER 3: DEEP LEAST SQUARES SUPPORT VECTOR DATA DESCRIPTION .... 37
Introduction ................................................................. 37

Methodology ............................................................... 38

Neural Network with One Hidden Layer ............................... 41

Neural Network with Two Hidden Layers ............................ 44

Neural Network with $L$ Hidden Layers ............................... 47

DLS-SVDD Classification Boundary ................................. 49

Alternative Objective Function .................................... 54

Hyperparameter Tuning ................................................. 58

Simulation Study .......................................................... 61

Performance Evaluation ................................................. 61

Performance Settings ..................................................... 63

Uncorrelated Multivariate Normal Data ............................. 64

Low-dimensional setting: $p = 5$ and $N = 100$ ............... 65

Low-dimensional setting: $p = 10$ and $N = 100$ .............. 66

High-dimensional setting: $p = 100$ and $N = 40$ ............. 67

Correlated Multivariate Normal Data ............................... 68

Low-dimensional setting: $p = 5$ and $N = 100$ ............... 69
LIST OF FIGURES

Figure 3.1: Visualization of the simulated dataset comprised of the target class sampled from the Two Moons dataset and the outlier class uniformly sampled from a circle with radius $\frac{1}{5}$ in the input space. .............................. 50

Figure 3.2: Visualization of the DLS-SVDD decision boundary in the input space during model training after 15 epochs (left) and 100 epochs (right). ............... 51

Figure 3.3: Visualization of the DLS-SVDD decision boundary in the input space during model training after 950 epochs. ................................. 51

Figure 3.4: Visualization of the DLS-SVDD decision boundary in the input space during model training after 959 epochs (left) and 960 epochs (right). ........... 52

Figure 3.5: Visualization of the DLS-SVDD decision boundary in the input space during model training after 974 epochs (left) and 1000 epochs (right). .......... 53

Figure 3.6: Visualization of the DLS-SVDD decision boundary in the input space after training for 1023 epochs and reaching model convergence. .............. 53

Figure 3.7: DLS-SVDD simulation results for $p = 5$ with uncorrelated multivariate normal data. ................................................................. 65

Figure 3.8: DLS-SVDD simulation results for $p = 10$ with uncorrelated multivariate normal data. ................................................................. 66

Figure 3.9: DLS-SVDD simulation results for $p = 100$ with uncorrelated multivariate normal data. ................................................................. 67
Figure 3.10: DLS-SVDD simulation results for \( p = 5 \) with correlated multivariate normal data. .......................................................... 69

Figure 3.11: DLS-SVDD simulation results for \( p = 10 \) with correlated multivariate normal data. .......................................................... 70

Figure 3.12: DLS-SVDD simulation results for \( p = 100 \) with correlated multivariate normal data. .......................................................... 71

Figure 3.13: DLS-SVDD simulation results for \( p = 5 \) with multivariate skew-normal data. .......................................................... 74

Figure 3.14: DLS-SVDD simulation results for \( p = 10 \) with multivariate skew-normal data. .......................................................... 75

Figure 3.15: DLS-SVDD simulation results for \( p = 100 \) with multivariate skew-normal data. .......................................................... 76

Figure 3.16: DLS-SVDD simulation results for \( p = 5 \) with alternative activation functions. .......................................................... 78

Figure 3.17: DLS-SVDD simulation results for \( p = 10 \) with alternative activation functions. .......................................................... 79

Figure 3.18: DLS-SVDD simulation results for \( p = 5 \) with contaminated data. .......................................................... 81

Figure 3.19: DLS-SVDD simulation results for \( p = 10 \) with contaminated data. .......................................................... 82

Figure 3.20: DLS-SVDD simulation results for \( p = 100 \) with contaminated data. .......................................................... 83

Figure 3.21: OCC using LS-SVDD for the breast cancer dataset example. .......................................................... 87

Figure 3.22: OCC using SVDD for the breast cancer dataset example. .......................................................... 88
Figure 3.23: OCC using DLS-SVDD for the breast cancer dataset example. 89

Figure 4.1: Visualization of the simulated dataset comprised of the target class sampled from the Two Moons dataset and the outlier class uniformly sampled from a circle with radius $\frac{1}{5}$ in the input space. 104

Figure 4.2: Visualization of the DLS-OCSVM decision boundary in the input space during model training after 5 epochs. 105

Figure 4.3: Visualization of the DLS-OCSVM decision boundary in the input space during model training after 10 epochs (left) and 20 epochs (right). 106

Figure 4.4: Visualization of the DLS-OCSVM decision boundary in the input space during model training after 40 epochs (left) and 75 epochs (right). 106

Figure 4.5: Visualization of the DLS-OCSVM decision boundary in the input space during model training after 200 epochs. 107

Figure 4.6: DLS-OCSVM simulation results for $p = 5$ with uncorrelated multivariate normal data. 112

Figure 4.7: DLS-OCSVM simulation results for $p = 10$ with uncorrelated multivariate normal data. 113

Figure 4.8: DLS-OCSVM simulation results for $p = 20$ with uncorrelated multivariate normal data. 114

Figure 4.9: DLS-OCSVM simulation results for $p = 100$ with uncorrelated multivariate normal data. 115
Figure 4.10: DLS-OCSVM simulation results for $p = 5$ with alternative activation functions. .................................................. 117

Figure 4.11: DLS-OCSVM simulation results for $p = 10$ with alternative activation functions. .................................................. 118

Figure 4.12: DLS-OCSVM simulation results for $p = 20$ with alternative activation functions. .................................................. 119

Figure 4.13: DLS-OCSVM simulation results for $p = 100$ with alternative activation functions. .................................................. 120

Figure 4.14: OCC using LS-OCSVM for the breast cancer dataset example. .................. 123

Figure 4.15: OCC using SVDD for the breast cancer dataset example. .................. 124

Figure 4.16: OCC using DLS-OCSVM for the breast cancer dataset example. .................. 125

Figure 5.1: DSMDD and OC-DSMDD simulation results for dimensions $(n \times p) = (3, 5)$ with matrix-variate normal data. .................. 150

Figure 5.2: DSMDD and OC-DSMDD simulation results for dimensions $(n \times p) = (5, 5)$ with matrix-variate normal data. .................. 151

Figure 5.3: DSMDD and OC-DSMDD simulation results for dimensions $(n \times p) = (10, 20)$ with matrix-variate normal data. .................. 152

Figure 5.4: DSMDD and OC-DSMDD simulation results for dimensions $(n \times p) = (20, 10)$ with matrix-variate normal data. .................. 153

Figure 5.5: DSMDD and OC-DSMDD variance shift simulation results for $p = 5$, $m = 4.156$
Figure 5.6: DSMDD and OC-DSMDD correlation shift simulation results for $p = 5,$ $m = 4$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 157

Figure 5.7: DSMDD and OC-DSMDD variance and correlation shift simulation results for $p = 5,$ $m = 4$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 158

Figure 5.8: DSMDD and OC-DSMDD variance shift simulation results for $p = 10,$ $m = 7$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 159

Figure 5.9: DSMDD and OC-DSMDD correlation shift simulation results for $p = 10,$ $m = 7$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 160

Figure 5.10: DSMDD and OC-DSMDD variance and correlation shift simulation results for $p = 10,$ $m = 7$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 161

Figure 5.11: A sample of images of one individual in the ORL dataset. . . . . . . . . . . 167

Figure 5.12: OCC using SMDD for the ORL database of faces example. . . . . . . . . . 168

Figure 5.13: OCC using DSMDD for the ORL database of faces example. . . . . . . . . 169

Figure 5.14: OCC using SMDD for the aluminum smelting dataset example. . . . . . . . 172

Figure 5.15: OCC using DSMDD for the aluminum smelting dataset example. . . . . . . . 173

Figure 5.16: OCC using SMDD for the aluminum smelting dataset example. . . . . . . . 174

Figure 5.17: OCC using DSMDD for the aluminum smelting dataset example. . . . . . . . 175
CHAPTER 1: INTRODUCTION

Anomalous data may assume a variety of forms, but the presence of rare or unexpected events is ubiquitous and encountered across all scientific disciplines. Over the years, many people have attempted to provide a simple answer to the question “what is an anomaly?” Hawkins famously said, “An outlier is an observation which deviates so much from the other observations as to arouse suspicion that it was generated by a different mechanism” [19]. Generally speaking, anomalies are irregular observations that do not conform to the expected patterns found in the majority of the data.

The 19th century marked the inception of formal scholarly studies into anomaly detection [36]. However, informal inquiries may well have happened before this period. Nonetheless, anomaly detection has been the subject of extensive research across a multitude of application domains since that point in time. This positions the field of anomaly detection as a critical area of research for the foreseeable future.

Anomaly detection is crucial to understanding the world around us. As our world becomes more data-driven, dealing with large volumes of data is a common challenge. As this trend continues, the difficulty associated with detecting rare or unusual events of interest is increasing. For a variety of reasons, real-world data is typically contaminated by noise and prone to contain anomalies of various forms. Due to the generality of the problem setting, different types of frameworks have been developed to tackle the anomaly detection problem. Some of these frameworks have been developed for a particular domain, or for a specific type of data. In this dissertation, we consider a general framework for anomaly detection that is problem-agnostic, and can be applied to a vast number of real-world applications.
The broad applicability of anomaly detection in addressing diverse real-world challenges leads to numerous practical applications. Examples include fraud detection [41, 52, 55], medical diagnosis [6, 62], cybersecurity [5, 35, 40], disease outbreak detection [27, 51], industrial fault and damage detection [31, 34, 56], monitoring of infrastructure [2, 22], monitoring of stock markets [8, 63], event detection in the earth sciences [14, 47], and acoustic novelty detection [3, 33]. In this work, we refer to unusual events of interest as anomalies or outliers, but they have also been referred to as novelties, discordants, deviants, or abnormalities in the literature.

Anomalies can be further classified into different types. The most common types of anomalies typically encountered in practice are referred to as global outliers or point anomalies. These individual observations will be characteristically different from the majority of the data. In other words, these observations don’t follow the expected patterns or behavior of the data under consideration.

The second type of anomaly is called a contextual or conditional anomaly. For these types of anomalies, the context or environment is a critical factor. These observations might not seem significantly different from other data points upon direct examination, unlike what we would typically expect for point anomalies. However, when the data are viewed in the proper context, their anomalous behavior will come to light. In simple terms, when the data are viewed from a specific context, the behavior of the data will not follow expected data patterns and significantly deviate from normal behavior.

The last type of anomaly is known as a group or collective anomaly. Rather than focusing on individual anomalies, the objective now relates to detecting abnormal behavior within a group of related data. In this setting, an individual observation may appear normal when analyzed by itself. However, when considered as a group comprising the interconnected elements, abnormal behavior emerges.
In the scope of this dissertation, our primary emphasis is placed on the detailed examination and analysis of point anomalies. Contextual anomalies, which occur when data is anomalous within a specific context or situation, and collective anomalies, where a group of related data points collectively deviates from the expected behavior, while important in the broader field of anomaly detection, are not the focus of this research. This allows for a more concentrated and in-depth analysis of point anomalies, ensuring a thorough understanding and exploration of their characteristics, detection methodologies, and implications within the datasets under review.
CHAPTER 2: LITERATURE REVIEW

One-Class Classification

Binary classification is the most well-known type of classification problem. In this setting, a training set is available that consists of observations from two classes that have both a feature vector and a class label. In the test set, only the observations’ feature vectors are known, and the goal is to predict the class label. A special case of binary classification, known as one-class classification, arises when only data originating from a single class is available. Under this setting, we have no information about data that may originate from one of potentially many different types of classes. One-class classification was first coined by Moya et. al [42], where a neural network was used to construct a closed boundary around the training data. Since that time, one-class classification has developed into an active area of research that spans numerous scientific disciplines including statistics, machine learning, computer science, and engineering.

One-class classification is a special type of classification problem where data is assumed to originate from a single class, referred to as the target class. These observations are referred to as targets and follow some type of expected behavior. Alternatively, observations that deviate from the target class in some measurable way are referred to as outliers, or anomalies. The goal is to identify observations that deviate from the expected behavior of the target class. Hence, the one-class classification framework directly aligns with the goals of the anomaly detection task.

One-class classification methods can be classified into three main categories: density estimation, reconstruction methods, and boundary methods. While the end goal of one-class classification techniques is to identify anomalies, the three types of methods solve the problem using different approaches. Some of these approaches may assume additional underlying assumptions related to
their modeling approach or the data generating process, whereas others require minimal underlying assumptions.

Density estimation methods directly estimate the probability density function of the target class to discriminate between targets and outliers. These methods can be classified as either parametric or non-parametric approaches. Parametric-based approaches assume the underlying distribution is known, and discordancy tests are used to evaluate whether unseen observations are outliers. Examples of parametric-based density estimation methods include Gaussian models and Gaussian Mixture Models (GMMs) [30, 46]. A popular nonparametric density estimator is the Kernel Density Estimator (KDE) [45], which has certain theoretical advantages over histograms [54]. In low-dimensional settings, classical nonparametric density estimators achieve relatively good performance. However, in high-dimensional settings the curse of dimensionality dramatically hampers their performance. In order to achieve a consistent level of accuracy with these methods, the necessary sample size grows significantly with respect to the dimension of the feature space.

Reconstruction methods leverage a model’s ability to distinguish between normal and abnormal data patterns by comparing original input with reconstructed input. These methods operate under the assumption that there is prior knowledge about the underlying data generating process. With the underlying assumption that normal data will have low reconstruction error and anomalies will have higher-than-average reconstruction error, these techniques identify anomalies as instances with reconstruction errors exceeding a predefined threshold. Examples of reconstruction methods include Autoencoders [21, 25] and its variants [1, 61], Principal Component Analysis (PCA) [13, 18, 24, 44, 50] and its variants including Kernel PCA [23] and Robust PCA [7, 29, 43, 59].

Boundary methods aim to construct a classification boundary that separates the target class from outliers. Under this framework, there are no underlying assumptions regarding the nature of the outliers. The most prominent boundary-based kernel methods include the One-Class Support Vec-
tor Machine [49] and the Support Vector Data Description [53]. However, in complex, high-dimensional data environments, these kernel-based methods may face performance challenges. Recently, deep learning variants of these methods, such as One-Class Neural Network [9] and Deep Support Vector Data Description [48], have demonstrated superior performance in high-dimensional settings.

The following literature review contains two parts. The first section covers kernel-based support vector methods. The well-known One-Class Support Vector Machine and Support Vector Data Description, as well as their least squares reformulations, are reviewed. Additionally, we review Support Matrix Data Description, the matrix generalization of SVDD. The second section covers approaches that leverage neural networks for feature mapping, moving away from traditional kernel functions. In this section, we review One-Class Neural Network and Deep Support Vector Data Description, the deep learning analogues of OCSVM and SVDD, respectively.
Schölkopf et al. [49] proposed the One-Class Support Vector Machine (OCSVM), where a maximum margin hyperplane is used to separate the target class from outliers. Consider a training dataset \( X = \{x_1, x_2, \ldots, x_N\} \subseteq \mathcal{X} \), where \( N \) is the number of vectors and \( \mathcal{X} \subseteq \mathbb{R}^p \) is an original input space for which we want a description. A description refers to a model characterized by a classification boundary in the form of a separating hyperplane that distinguishes the target class from outliers. For this input space, we will define a corresponding output space \( \mathcal{F} \). Let \( \varphi(\cdot) : \mathcal{X} \mapsto \mathcal{F} \) denote a kernel function mapping the input space \( \mathcal{X} \) into a higher dimensional feature space \( \mathcal{F} \), where the feature space \( \mathcal{F} \) is a reproducing kernel Hilbert space (RKHS).

The goal is to separate the target class from the origin using a hyperplane by maximizing the distance between the hyperplane and the origin. This hyperplane is characterized by a normal vector \( \mathbf{w} \in \mathcal{F} \) and an offset \( \rho \in \mathbb{R} \). The formal mathematical representation of this model for finding the hyperplane is given by the following constrained optimization problem:

\[
\min_{\rho, \mathbf{w}, \xi} \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} - \rho + C \sum_{i=1}^{N} \xi_i \\
\text{subject to} \quad \mathbf{w}^T \varphi(x_i) \geq \rho - \xi_i, \quad i = 1, 2, \ldots, N \\
\text{and} \quad \xi_i \geq 0, \quad i = 1, 2, \ldots, N
\]

(2.1)

where \( \mathbf{w} \) is the normal vector perpendicular to the hyperplane, \( \rho \) is an offset or bias, \( \xi_i \) are the slack variables, and the parameter \( C > 0 \) controls the influence of the slack variables.
The conventional approach to solve the constrained optimization problem (2.1) is to use the Lagrange dual problem. Consequently, by incorporating the two sets of inequality constraints into the objective function using Lagrange multipliers, we obtain the following Lagrangian function:

$$
\mathcal{L}(\rho, w, \alpha, \gamma, \xi) = \frac{1}{2} w^T w - \rho + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left( w^T \varphi(x_i) - \rho + \xi_i \right) - \sum_{i=1}^{N} \gamma_i \xi_i \tag{2.2}
$$

where $\alpha_i \geq 0$ and $\gamma_i \geq 0$ are the Lagrange multipliers.

Necessary and sufficient conditions for any optimal solution are given by the Karush-Kuhn-Tucker (KKT) optimality conditions. The KKT conditions for OCSVM are obtained by taking the derivative of the Lagrangian with respect to the primal variables $\rho, w$, and $\xi$. Therefore, by differentiating (2.2) with respect to these variables and setting the derivatives equal to zero yields:

$$
\frac{\partial \mathcal{L}}{\partial \rho} = 0 \implies \sum_{i=1}^{N} \alpha_i = 1 \tag{2.3}
$$
$$
\frac{\partial \mathcal{L}}{\partial w} = 0 \implies w = \sum_{i=1}^{N} \alpha_i \varphi(x_i) \tag{2.4}
$$
$$
\frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \implies \alpha_i = C - \gamma_i \tag{2.5}
$$

For the OCSVM, the KKT conditions are summarized below.

(1) Stationarity

1. $1 - \sum_{i=1}^{N} \alpha_i = 0$

2. $w - \sum_{i=1}^{N} \alpha_i \varphi(x_i) = 0$

3. $C + \alpha_i - \gamma_i = 0$
(2) Primal Feasibility

1. \( \mathbf{w}^T \varphi(x_i) \geq \rho - \xi_i \)

(3) Dual Feasibility

1. \( 0 < \alpha_i < C \)

(4) Complementary Slackness

1. \( \alpha_i (\mathbf{w}^T \varphi(x_i) - \rho + \xi_i) = 0 \)

Considering the equations above and the non-negative nature of the Lagrange multipliers, we can safely remove the Lagrange multipliers \( \gamma_i \) from the optimization problem by imposing the following constraint:

\[
0 \leq \alpha_i \leq C, \quad i = 1, 2, ..., N
\]  

(2.6)

Then, the primal variables \( \rho, \mathbf{w}, \) and \( \xi \) can be eliminated using substitution, which yields the following Lagrangian equation exclusively in terms of the dual variables:

\[
\mathcal{L}(\alpha) = \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j \varphi(x_i)^T \varphi(x_j)
\]

(2.7)

subject to the constraints (2.6) and \( \sum_{i=1}^{N} \alpha_i = 1 \).

After \( \mathcal{L} \) has been minimized with respect to the primal variables \( \rho, \mathbf{w}, \) and \( \xi \) given \( \alpha \) and \( \gamma \), then it is maximized with respect to \( \alpha \). The maximization of (2.7) results in a set of dual variables \( \alpha^* \),
and the dual problem is given by the following:

\[
\min_{\alpha} \quad \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j \varphi(x_i)^T \varphi(x_j) \\
\text{subject to} \quad \sum_{i=1}^{N} \alpha_i = 1, \quad \sum_{i=1}^{N} \alpha_i = 1, \quad (2.8)
\]

and 

\[0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, N\]

There is no need to compute the features \(\varphi(x_i)\) when one knows how to compute the products directly using a valid Mercer kernel. Consequently, the inner product \(\varphi(x_i)^T \varphi(x_j)\) is replaced by a kernel function \(k(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)\), which maps the original non-linearly separable input space into a higher dimensional feature space where the features can now be linearly separated. The resulting dual problem can be expressed as follows:

\[
\min_{\alpha} \quad \frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \\
\text{subject to} \quad \sum_{i=1}^{N} \alpha_i = 1, \quad (2.9)
\]

and 

\[0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, N\]

This is a quadratic programming problem, and it can be solved efficiently using quadratic programming solver packages. Once the dual is maximized, the resulting Lagrange multipliers \(\alpha^*\) can be used to calculate the normal vector \(w\).

When a training vector \(\varphi(x_i)\) satisfies the condition \(w^T \varphi(x_i) > \rho - \xi_i\), then the inequality constraint is satisfied and the corresponding Lagrange multiplier will be zero (\(\alpha_i = 0\)). For training vectors satisfying the equality constraint \(w^T \varphi(x_i) = \rho - \xi_i\), then the constraint has to be enforced and the Lagrange multiplier will become larger than zero (\(\alpha_i > 0\)).
The three possible scenarios that may occur for a given training vector $\varphi(x_i)$ are shown below.

$$w^T \varphi(x_i) > \rho \implies \alpha_i = 0, \quad \gamma_i = C \quad (2.10)$$

$$w^T \varphi(x_i) = \rho \implies 0 < \alpha_i < C, \quad \gamma_i > 0 \quad (2.11)$$

$$w^T \varphi(x_i) < \rho \implies \alpha_i = C, \quad \gamma_i = 0 \quad (2.12)$$

Note that Equation (2.4) shows that the normal vector $w$ is a linear combination of the Lagrange multipliers. However, only training vectors $\varphi(x_i)$ with $\alpha_i > 0$ are needed in the data description. The training vectors $\varphi(x_i)$ corresponding to positive Lagrange multipliers $\alpha_i > 0$ are the support vectors. The support vectors that correspond to training vectors that lie directly on the hyperplane boundary in the feature space, with $0 < \alpha_i < C$, are the boundary support vectors. The other support vectors, with $\alpha_i = C$, are the vectors that lie between the hyperplane and the origin of the unit ball referred to as the non-boundary support vectors.

The next step is to compute the offset of the hyperplane $\rho$. There are two approaches to evaluating the value of $\rho$. One approach involves using a randomly selected non-zero support vector, denoted by $x_s$. An alternative approach is to use the set of all support vectors, denoted by $N_s$, and calculating the average across all support vectors as follows:

$$\rho = \frac{1}{N_s} \sum_{s \in S} \left( \sum_{i \in S} \alpha_i^s k(x_s, x_i) \right) \quad (2.13)$$

The equation for the hyperplane is given by:

$$f(x) = \sum_{i=1}^{N} \alpha_i k(x, x_i) - \rho \quad (2.14)$$

For a previously unseen test vector $z$, the hyperplane can be leveraged directly for discrimination based on which side of the hyperplane the observation falls.
So, a test vector \( z \) is classified as a target based on the following decision function:

\[
g(z) = sgn \left( \sum_{i=1}^{N} \alpha_i k(x, x_i) - \rho \right)
\]  

(2.15)

If \( g(z) = 1 \), then \( z \) is classified as a target. Otherwise, it is classified as an outlier.

**Least Squares One-Class Support Vector Machine**

Least Squares One-Class Support Vector Machine (LS-OCSVM), which was developed by Choi [11], is a reformulation of the standard One-Class Support Vector Machine (OCSVM) problem of Schölkopf et al. [49] using a quadratic error function and equality constraints. Consider a training dataset \( X = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X} \) where \( N \) is the number of vectors and \( \mathcal{X} \subseteq \mathbb{R}^p \) is an original input space for which we want a description. A description refers to a model characterized by a classification boundary in the form of a separating hyperplane that distinguishes the target class from outliers. For this input space, we will define a corresponding output space \( \mathcal{F} \). Let \( \varphi(\cdot) : \mathcal{X} \mapsto \mathcal{F} \) denote a kernel function mapping the input space \( \mathcal{X} \) into a higher dimensional feature space \( \mathcal{F} \), where the feature space \( \mathcal{F} \) is a reproducing kernel Hilbert space (RKHS).

The goal is to separate the target class from the origin using a hyperplane by maximizing the distance between the hyperplane and the origin such that the sum of squares of errors are minimized. This hyperplane is characterized by a normal vector \( w \in \mathcal{F} \) and an offset \( \rho \in \mathbb{R} \). The formal mathematical representation of this model for finding the hyperplane is given by the following constrained optimization problem:

\[
\begin{align*}
& \min_{\rho, w, \xi} \quad \frac{1}{2} w^T w - \rho + \frac{C}{2} \sum_{j=1}^{N} \xi_j^2 \\
& \text{subject to} \quad w^T \varphi(x_j) = \rho - \xi_j, \quad j = 1, 2, ..., N
\end{align*}
\]  

(2.16)
where \( w \) is the normal vector perpendicular to the hyperplane, \( \rho \) is an offset or bias, \( \xi_j \) are the error variables arising from a training vector \( x_j \) with respect to the hyperplane, and the parameter \( C > 0 \) is introduced to control the influence of the error variables.

The conventional approach to solve the constrained optimization problem (2.16) is to use the Lagrange dual problem. Consequently, by incorporating the set of equality constraints into the objective function using Lagrange multipliers, we obtain the following Lagrangian function:

\[
\mathcal{L}(\rho, w, \alpha, \xi) = \frac{1}{2} w^T w - \rho + \frac{C}{2} \sum_{j=1}^{N} \xi_j^2 - \sum_{j=1}^{N} \alpha_j \left( w^T \varphi(x_j) - \rho + \xi_j \right) \quad \text{(2.17)}
\]

where \( \alpha_j, j = 1, 2, ..., N \) are the Lagrange multipliers, which can be either positive or negative as a result of the equality constraints.

The solution for the LS-OCSVM can be obtained by taking the derivative of the Lagrangian with respect to the primal variables \( \rho, w, \) and \( \xi \). Therefore, by differentiating (2.17) with respect to these variables and setting the derivatives equal to zero yields:

\[
\frac{\partial \mathcal{L}}{\partial \rho} = 0 \implies \sum_{j=1}^{N} \alpha_j = 1 \quad \text{(2.18)}
\]

\[
\frac{\partial \mathcal{L}}{\partial w} = 0 \implies w = \sum_{j=1}^{N} \alpha_j \varphi(x_j) \quad \text{(2.19)}
\]

\[
\frac{\partial \mathcal{L}}{\partial \xi_j} = 0 \implies \alpha_j = C \xi_j \quad \text{(2.20)}
\]

Next, the primal variables \( w \) and \( \xi \) can be eliminated using substitution, which yields the following system of linear equations for \( \alpha_j \) and \( \rho \):

\[
\begin{cases}
\sum_{j=1}^{N} \alpha_j k(x_i, x_j) - \rho + \frac{\alpha_j}{C} \\
\sum_{j=1}^{N} \alpha_j = 1
\end{cases}
\quad \text{(2.21)}
\]
Then, we can express the problem (2.21) in matrix form as follows:

\[
\begin{bmatrix}
0 & e^T \\
e & H
\end{bmatrix}
\begin{bmatrix}
-\rho \\
\alpha
\end{bmatrix}
= \begin{bmatrix}
1 \\
0
\end{bmatrix}
\tag{2.22}
\]

where \( K \) is a \( N \times N \) Gram matrix with \((i, j)th\) entry \( K_{ij} = k(x_i, x_j) \), \( I_N \) is a \( N \times N \) identity matrix, \( H = K + \frac{1}{C} I_N \), \( e \) is a \( N \times 1 \) vector of ones, and \( \alpha \) is a \( N \times 1 \) vector of Lagrange multipliers.

The analytic solution of the dual problem for LS-OCSVM is given by:

\[\rho = \frac{1}{e^T H^{-1} e}\]  
\[\tag{2.23}\]

and

\[\alpha = \frac{H^{-1}}{e^T H^{-1} e} e\]  
\[\tag{2.24}\]

Consequently, the errors \( \xi \) associated with the training set vectors relative to the hyperplane can be calculated using:

\[\xi = \frac{1}{C e^T H^{-1} e} e\]  
\[\tag{2.25}\]

The equation for the hyperplane is given by:

\[f(x) = \sum_{j=1}^{N} \alpha_j k(x, x_j) - \rho\]  
\[\tag{2.26}\]

Unlike OCSVM, LS-OCSVM does not have a natural decision function due to the quadratic objective. Rather, the proximity to the hyperplane was suggested by Choi [11] as a discriminative measure. However, using a distance measure that is based on the proximity to the hyperplane can result in very large values. To mitigate this problem, Maboudou-Tchao [38] introduced a standard-
ized measure of proximity, which circumvents the complication of dealing with large values, and this standardized measure is employed in our research.

For a previously unseen test vector $z$, the kernel distance statistic $d$ is computed as follows:

$$d = \frac{\left( \sum_{j=1}^{N} \alpha_j k(x_j, z) - \rho \right)^2}{\alpha^T \alpha}$$  \hspace{1cm} (2.27)

To classify the test vector $z$ as either a target or an outlier, an upper limit threshold based on the desired Type I error rate is required. The threshold, denoted by $h$, can be obtained using the bootstrap simulation technique summarized in Algorithm 1. The test vector $z$ is classified as a target if the condition $d \leq h$ is satisfied. Otherwise, it is classified as an outlier.

**Algorithm 1 LS-OCSVM Bootstrap Algorithm for Threshold $h$**

**Input:** Training dataset $X = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X}$ with $x_j \in \mathbb{R}^P$, $B$ bootstrap samples, Lagrange multipliers $\alpha^*$, and hyperplane offset $\rho^*$ of a trained model.

**Output:** A threshold $h$.

1: for $x_j \in \mathcal{X}$ do  
2: \hspace{1cm} find $d_j = \frac{\left( \sum_{j=1}^{N} \alpha_j^* k(x, x_j) - \rho^* \right)^2}{\alpha^{*T} \alpha^*}$  
3: end for  
4: for $b \leftarrow 1, B$ do  
5: \hspace{1cm} (i) Draw a bootstrap sample of size $N$ from the set of $N$ $d$ statistics  
6: \hspace{1cm} (ii) If $\alpha$ is the desired Type I error, determine the $100 \times (1 - \alpha)^{th}$ percentile value  
7: end for  
8: Obtain the threshold $h$ by taking an average of $B$ $100 \times (1 - \alpha)^{th}$ percentile values  
9: return $h$

**Support Vector Data Description**

Tax and Duin [53] introduced the Support Vector Data Description (SVDD), where a minimum volume hypersphere is used to enclose the target class, effectively isolating these observations from outliers. Consider a training dataset $X = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X}$ where $N$ is the number of
vectors and $\mathcal{X} \subseteq \mathbb{R}^p$ is an original input space for which we want a description. A description refers to a model characterized by a closed boundary in the form of a minimum volume hypersphere that distinguishes the target class from outliers. For this input space, we will define a corresponding output space $\mathcal{F}$. Let $\varphi(\cdot) : \mathcal{X} \mapsto \mathcal{F}$ denote a kernel function mapping the input space $\mathcal{X}$ into a higher dimensional feature space $\mathcal{F}$, where $\mathcal{F}$ is a reproducing kernel Hilbert space (RKHS).

The goal is to encapsulate the target class using a minimum volume hypersphere characterized by its center $a \in \mathcal{F}$ and radius $R > 0$ that contains all, or most, of the target class vectors. The formal mathematical representation of this model for finding the hypersphere is given by the following constrained optimization problem:

$$
\min_{R, a, \xi} \quad R^2 + C \sum_{i=1}^{N} \xi_i
$$

subject to

$$
||\varphi(x_i) - a||^2 \leq R^2 + \xi_i, \quad i = 1, 2, \ldots, N
$$

and

$$
\xi_i \geq 0, \quad i = 1, 2, \ldots, N
$$

where $a$ is the center of the hypersphere, $R$ is the radius of the hypersphere, $\xi_i$ are the slack variables, and $C > 0$ is a parameter used to control the influence of the slack variables.

The conventional approach to solve the constrained optimization problem (2.28) is to use the Lagrange dual problem. Consequently, by incorporating the two sets of inequality constraints into the objective function using Lagrange multipliers, we obtain the following Lagrangian function:

$$
\mathcal{L}(R, a, \alpha, \gamma, \xi) = R^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left( R^2 + \xi_i - ||\varphi(x_i) - a||^2 \right) - \sum_{i=1}^{N} \gamma_i \xi_i
$$

where $\alpha_i \geq 0$ and $\gamma_i \geq 0$ the Lagrange multipliers.
Necessary and sufficient conditions for any optimal solution are given by the Karush-Kuhn-Tucker (KKT) optimality conditions. The KKT conditions for the SVDD are obtained by taking the derivative of the Lagrangian with respect to the primal variables \( R, a, \) and \( \xi \). Therefore, by differentiating \((2.29)\) with respect to these variables and setting the derivatives equal to zero yields:

\[
\frac{\partial L}{\partial R} = 0 \implies \sum_{i}^{N} \alpha_i = 1 \tag{2.30}
\]
\[
\frac{\partial L}{\partial a} = 0 \implies a = \sum_{i=1}^{N} \alpha_i \varphi(x_i) \tag{2.31}
\]
\[
\frac{\partial L}{\partial \xi_i} = 0 \implies \alpha_i = C - \gamma_i \tag{2.32}
\]

For the SVDD, the KKT conditions are summarized below.

(1) Stationarity

1. \( 1 - \sum_{i=1}^{N} \alpha_i = 0 \)
2. \( a - \sum_{i=1}^{N} \alpha_i \varphi(x_i) = 0 \)
3. \( C - \alpha_i - \gamma_i = 0 \)

(2) Primal Feasibility

1. \( \| \varphi(x_i) - a \|^2 \leq R^2 + \xi_i \)

(3) Dual Feasibility

1. \( 0 < \alpha_i < C \)

(4) Complementary Slackness

1. \( \alpha_i (\| \varphi(x_i) - a \|^2 - R^2 - \xi_i) = 0 \)
Considering the equations above and the non-negative nature of the Lagrange multipliers, we can safely remove the Lagrange multipliers $\gamma_i$ from the optimization problem by imposing the following constraint:

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, N$$

(2.33)

Then, the primal variables $R$, $a$, and $\xi$ can be eliminated using substitution, which yields the following Lagrangian equation exclusively in terms of the dual variables:

$$L(\alpha) = \sum_{i=1}^{N} \alpha_i \varphi(x_i)^T \varphi(x_i) - \sum_{i,j=1}^{N} \alpha_i \alpha_j \varphi(x_i)^T \varphi(x_j)$$

(2.34)

subject to the constraints (2.33) and $\sum_{i=1}^{N} \alpha_i = 1$.

After $L$ has been minimized with respect to the primal variables $R$, $a$, and $\xi$ given $\alpha$ and $\gamma$, then it is maximized with respect to $\alpha$. The maximization of (2.34) results in a set of dual variables $\alpha^*$, and the dual problem is given by the following:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i \varphi(x_i)^T \varphi(x_i) - \sum_{i,j=1}^{N} \alpha_i \alpha_j \varphi(x_i)^T \varphi(x_j)$$

subject to $\sum_{i=1}^{N} \alpha_i = 1$  

(2.35)

and $0 \leq \alpha_i \leq C, \quad i = 1, 2, \ldots, N$

There is no need to compute the features $\varphi(x_i)$ when one knows how to compute the products directly using a valid Mercer kernel. Consequently, the inner product $\varphi(x_i)^T \varphi(x_j)$ is replaced by a kernel function $k(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$, which maps the training data from the original input space to a bounded spherical region in the feature space, effectively isolating outliers beyond this

18
closed boundary. The resulting dual problem can be expressed as follows:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i k(x_i, x_i) - \sum_{i,j=1}^{N} \alpha_i \alpha_j k(x_i, x_j)$$

subject to $\sum_{i=1}^{N} \alpha_i = 1$ (2.36)

and $0 \leq \alpha_i \leq C$, $i = 1, 2, ..., N$

This is a quadratic programming problem and can be solved efficiently using quadratic programming solver packages. Once the dual is maximized, the resulting Lagrange multipliers $\alpha^*$ can be used to calculate the center of the hypersphere $a$.

When a training vector $\varphi(x_i)$ satisfies the condition $||\varphi(x_i) - a||^2 < R^2 + \xi$, then the inequality constraint is satisfied and the corresponding Lagrange multiplier will be zero ($\alpha_i = 0$). For training vectors satisfying the equality constraint $||\varphi(x_i) - a||^2 = R^2 + \xi$, then the constraint has to be enforced and the Lagrange multiplier will become larger than zero ($\alpha_i > 0$). The three possible scenarios that may occur for a given training vector $\varphi(x_i)$ are shown below.

$$||\varphi(x_i) - a||^2 < R^2 \implies \alpha_i = 0, \gamma_i = 0 \quad (2.37)$$

$$||\varphi(x_i) - a||^2 = R^2 \implies 0 < \alpha_i < C, \gamma_i > 0 \quad (2.38)$$

$$||\varphi(x_i) - a||^2 > R^2 \implies \alpha_i = C, \gamma_i = 0 \quad (2.39)$$

Note that Equation (2.31) shows that the hypersphere center $a$ is a linear combination of the Lagrange multipliers. However, only training vectors $\varphi(x_i)$ with $\alpha_i > 0$ are needed in the data description. The training vectors $\varphi(x_i)$ corresponding to positive Lagrange multipliers $\alpha_i > 0$ are the support vectors. The support vectors that correspond to training vectors that lie directly on the hypersphere boundary in the feature space, with $0 < \alpha_i < C$, are the boundary support vectors.
The other support vectors, with \( \alpha_i = C \), are referred to as the non-boundary support vectors.

The next step is to compute the radius of the hypersphere \( R \). There are two approaches to evaluating the value of \( R \). By definition, the radius is the Euclidean distance from the center of the hypersphere \( a \) to any of the support vectors on the boundary in the kernel space. Support vectors which fall outside the data description \((\alpha_i = C)\) are excluded. An alternative approach is to use the set of all boundary support vectors, denoted by \( N_s \), and calculating the average squared kernel distance from the center of the hypersphere to the boundary support vectors given by the following equation:

\[
R^2 = \frac{1}{N_s} \sum_{s=1}^{N_s} \left( k(x_s, x_s) - 2 \sum_{i=1}^{N_s} \alpha_i k(x_s, x_i) + \sum_{i,j=1}^{N_s} \alpha_i \alpha_j k(x_i, x_j) \right) \tag{2.40}
\]

for any \( x_i \in SV_{<C} \), the set of support vectors which have \( \alpha_i < C \).

For a previously unseen test vector \( z \), the kernel distance statistic \( d \) is computed as follows:

\[
d = K(z, z) - 2 \sum_{i=1}^{N} \alpha_i K(x_i, z) + \sum_{i,j=1}^{N} \alpha_i \alpha_j K(x_i, x_j) \tag{2.41}
\]

The test vector \( z \) is classified as a target if the condition \( d \leq R^2 \) is satisfied. Otherwise, it is classified as an outlier.

**Least Squares Support Vector Data Description**

Least Squares Support Vector Data Description (LS-SVDD), which was developed by Guo et al. [17], is a reformulation of the standard Support Vector Data Description (SVDD) proposed by Tax and Duin [53] using a quadratic error function and equality constraints. Consider a training dataset \( X = \{x_1, x_2, \ldots, x_N\} \subseteq \mathcal{X} \) where \( N \) is the number of vectors and \( \mathcal{X} \subseteq \mathbb{R}^p \) is an original input.
space for which we want a description. A description refers to a model characterized by a closed boundary in the form of a minimum volume hypersphere that distinguishes the target class from outliers. For this input space, we will define a corresponding output space $\mathcal{F}$. Let $\varphi(\cdot) : \mathcal{X} \mapsto \mathcal{F}$ denote a kernel function mapping the input space $\mathcal{X}$ into a higher dimensional feature space $\mathcal{F}$, where $\mathcal{F}$ is a reproducing kernel Hilbert space (RKHS).

The goal is to encapsulate the target class in a minimum volume hypersphere characterized by its center $a \in \mathcal{F}$ and radius $R > 0$ such that the sum of squares of errors are minimized. The formal mathematical representation of this model for finding the hypersphere is given by the following constrained optimization problem:

$$
\min_{R, a, \xi} R^2 + \frac{C}{2} \sum_{j=1}^{N} \xi_j^2
$$

subject to $||\varphi(x_j) - a||^2 = R^2 + \xi_j, \ j = 1, 2, \ldots, N$ \hspace{1cm} (2.42)

where $a$ is the center of the hypersphere, $R$ is the radius of the hypersphere, $\xi_j$ are the error variables realized by a training vector $x_j$ with respect to the hypersphere, and the parameter $C > 0$ is introduced to control the influence of the error variables.

The conventional approach to solve the constrained optimization problem (2.42) is to use the Lagrange dual problem. Consequently, by incorporating the set of equality constraints into the objective function using Lagrange multipliers, we obtain the following Lagrangian function:

$$
\mathcal{L}(R, a, \alpha, \xi) = R^2 + C \sum_{j=1}^{N} \xi_j^2 - \sum_{j=1}^{N} \alpha_j \left( R^2 + \xi_j - ||\varphi(x_j) - a||^2 \right) \hspace{1cm} (2.43)
$$

where $\alpha_j, j = 1, 2, \ldots, N$ are the Lagrange multipliers, which can assume either positive or negative values as a result of the equality constraints.
The solution for the LS-SVDD can be obtained by taking the derivative of the Lagrangian with respect to the primal variables $R$, $a$, and $\xi$. Therefore, by differentiating (2.43) with respect to these variables and setting the derivatives equal to zero yields:

\[
\frac{\partial L}{\partial R} = 0 \implies \sum_{j=1}^{N} \alpha_j = 1 \\
\frac{\partial L}{\partial a} = 0 \implies a = \sum_{j=1}^{N} \alpha_j \varphi(x_j) \\
\frac{\partial L}{\partial \xi_j} = 0 \implies \alpha_j = 2C \xi_j
\]

Next, the primal variables $R$, $a$, and $\xi$ can be eliminated using substitution, which yields the LS-SVDD dual problem:

\[
\begin{align*}
\text{maximize} & \quad \sum_{j=1}^{N} \alpha_j \varphi(x_j)^T \varphi(x_j) - \sum_{i,j=1}^{N} \alpha_i \alpha_j \left( \varphi(x_i)^T \varphi(x_j) + \frac{1}{2C} \delta_{ij} \right) \\
\text{subject to} & \quad \sum_{j=1}^{N} \alpha_j = 1
\end{align*}
\]

where $\delta_{ij}$ denotes the Kronecker Delta function as shown below.

\[
\delta_{ij} = \begin{cases} 
1 & \text{if } i \neq j \\
0 & \text{if } i = j
\end{cases}
\]

Kernel-based support vector methods for one-class classification, such as One-Class Support Vector Machine and Support Vector Data Description, were originally solved using quadratic programming solvers. However, the implementation of these methods may present difficulties, particularly when confronted with complex high-dimensional data due to the increasing computational costs associated with using quadratic programming solvers. To reduce the computational burden,
Maboudou-Tchao [39] demonstrated that the least squares variant of the Support Vector Data Description yields a closed-form solution, effectively overcoming the computational issues.

The Lagrangian of the LS-SVDD dual problem is:

\[
L(\alpha) = \sum_{j=1}^{N} \alpha_j \varphi(x_j)^T \varphi(x_j) - \sum_{i,j=1}^{N} \alpha_i \alpha_j \left( \varphi(x_i)^T \varphi(x_j) + \frac{1}{2C} \delta_{ij} \right) + \gamma \left( \sum_{j=1}^{N} \alpha_j - 1 \right) \quad (2.48)
\]

There is no need to compute the features \( \varphi(x_i) \) when one knows how to compute the products directly using a valid Mercer kernel. Consequently, the inner product \( \varphi(x_i)^T \varphi(x_j) \) is replaced by a kernel function \( k(x_i, x_j) = \varphi(x_i)^T \varphi(x_j) \), which maps the training data from the original input space to a bounded spherical region in the feature space, effectively isolating outliers beyond this closed boundary. The resulting dual problem can be expressed as follows:

\[
L(\alpha) = \sum_{j=1}^{N} \alpha_j k(x_j, x_j) - \sum_{i,j=1}^{N} \alpha_i \alpha_j \left( k(x_i, x_j) + \frac{1}{2C} \delta_{ij} \right) + \gamma \left( \sum_{j=1}^{N} \alpha_j - 1 \right) \quad (2.49)
\]

The dual problem can be equivalently expressed in matrix form as follows:

\[
L(\alpha) = -\alpha^T H \alpha + k^T \alpha + \gamma (e^T \alpha - 1) \quad (2.50)
\]

where \( K \) is a \( N \times N \) Gram matrix with \( (i, j)th \) entry \( K_{ij} = k(x_i, x_j) \), \( k \) denotes a vector with entries \( k(x_j, x_j) \), \( j = 1, 2, ..., N \), \( I_N \) is a \( N \times N \) identity matrix, \( H = K + \frac{1}{C} I_N \), \( e \) is a \( N \times 1 \) vector of ones, and \( \alpha \) is a \( N \times 1 \) vector of Lagrange multipliers.

Note that the problem (2.50) is now expressed as an unconstrained optimization problem with respect to the variable \( \alpha \). Hence, any conventional approach used for solving unconstrained convex optimization problems can be applied to solve this problem.
By differentiating (2.50) with respect to $\alpha$ and setting the derivative equal to zero, we obtain:

$$\frac{\partial L(\alpha)}{\partial \alpha} = 0 \implies \alpha = \frac{1}{2} H^{-1} (k + \gamma e) \quad (2.51)$$

To find $\gamma$, we differentiate (2.50) with respect to $\gamma$ and setting the derivative equal to zero yields:

$$\frac{\partial L(\alpha)}{\partial \gamma} = 0 \implies \gamma = 2 \frac{-e^T H^{-1} k}{e^T H^{-1} e} \quad (2.52)$$

By replacing $\gamma$ in (2.51) with (2.52), we can derive the following closed-form solution of the dual problem for LS-SVDD:

$$\alpha = \frac{1}{2} H^{-1} \left( k + \frac{2 - e^T H^{-1} k}{e^T H^{-1} e} e \right) \quad (2.53)$$

Consequently, the errors $\xi$ associated with the training set vectors relative to the hypersphere can be computed using:

$$\xi = \frac{\alpha}{C} = \frac{1}{2C} H^{-1} \left( k + \frac{2 - e^T H^{-1} k}{e^T H^{-1} e} e \right) \quad (2.54)$$

Using the closed-form solution for $\alpha$, the radius of the hypersphere is computed as follows:

$$R^2 = \frac{1}{N} \sum_{s=1}^{N} \left( k(x_s, x_s) - 2 \sum_{j=1}^{N} \alpha_j k(x_s, x_j) + \sum_{j,k=1}^{N} \alpha_j \alpha_k k(x_j, x_k) \right) \quad (2.55)$$

For a previously unseen test vector $z$, the kernel distance statistic $d$ is computed as follows:

$$d = K(z, z) - 2 \sum_{j=1}^{N} \alpha_j K(x_j, z) + \sum_{j,k}^{N} \alpha_j \alpha_k K(x_j, x_k) \quad (2.56)$$

The test vector $z$ is classified as a target if the condition $d \leq R^2$ is satisfied. Otherwise, it is
classified as an outlier.

**Support Matrix Data Description**

Support Matrix Data Description (SMDD), proposed by Maboudou-Tchao [37], is a one-class classification method for data whose native form are matrices. SMDD is a matrix generalization of SVDD that operates in matrix space, thereby eliminating the need to vectorize second-order tensors. Consider a training dataset \( X = \{ X_1, X_2, \ldots, X_N \} \subseteq \mathcal{X} \) where \( N \) is the number of matrices and \( \mathcal{X} \subseteq \mathbb{R}^{n \times p} \) is an original input space for which we want a description. A description refers to a model characterized by a closed boundary in the form of a minimum volume hypersphere, or a “matrix sphere,” that distinguishes the target class from outliers. For this input space, we will define a corresponding output space \( \mathcal{F} \). Let \( \varphi(\cdot) : \mathcal{X} \mapsto \mathcal{F} \) denote a kernel function mapping the input space \( \mathcal{X} \) into a higher dimensional feature space \( \mathcal{F} \), where \( \mathcal{F} \) is a reproducing kernel Hilbert space (RKHS).

The goal is to encapsulate the target class using a minimum volume hypersphere characterized by its center \( A \in \mathcal{F} \), which is a matrix, and radius \( R > 0 \) that contains all, or most, of the target class matrices. The formal mathematical representation of this model for finding the hypersphere is given by the following constrained optimization problem:

\[
\begin{align*}
\min_{R,A,\xi} & \quad R^2 + C \sum_{i=1}^{N} \xi_i \\
\text{subject to} & \quad ||\varphi(X_i) - A||_F^2 \leq R^2 + \xi_i, \quad i = 1, 2, \ldots, N \\
\text{and} & \quad \xi_i \geq 0, \quad i = 1, 2, \ldots, N
\end{align*}
\]  

(2.57)

where \( A \) denotes the center of the hypersphere, \( R \) is the radius of the hypersphere, \( \xi_i \) are the slack variables, the tuning parameter \( C > 0 \) regulates the trade-off between the target class errors and
the volume of the hypersphere by means of the slack variables, and $|| \cdot ||_F$ denotes the Frobenius norm.

To ensure the theoretical results of SMDD are consistent with a convex optimization problem, first we redefine the radius as follows:

$$\bar{R} = R^2$$

As pointed out by Chang et al. [10] in the case of vector-based SVDD, and Maboudou-Tchao [37] in the case of matrix-based SMDD, this is due to the following mathematical expression in the optimization problem being concave with respect to the radius $R$, thereby resulting in the optimization problem being non-convex.

$$||\varphi(X_i) - A||_F^2 - R^2 - \xi_i$$

The primal problem can be reformulated into the following constrained optimization problem:

$$\min_{\bar{R}, C, \xi} \bar{R} + C \sum_{i=1}^{N} \xi_i$$

subject to

$$||\varphi(X_i) - A||_F^2 \leq \bar{R} + \xi_i, \quad i = 1, 2, ..., N$$

and

$$\xi_i \geq 0, \quad i = 1, 2, ..., N$$

and

$$\bar{R} \geq 0$$

where we have added an additional inequality constraint forcing the radius $\bar{R}$ to be non-negative.
As a result, the following expression is now linear in terms of both the radius $\bar{R}$ and the slack variables $\xi_i$, and it is convex with respect to the hypersphere center $A$.

$$||\varphi(X_i) - A||_F^2 - \bar{R} - \xi_i = tr (\varphi(X_i)^T \varphi(X_i)) - 2tr (\varphi(X_i)^T A) + tr (A^T A) - \bar{R} - \xi_i$$

(2.61)

While the typical approach to solve the constrained optimization problem (2.60) is to use the Lagrange dual problem, a challenge arises when obtaining the optimal solution due to the non-negativity constraint $\bar{R} \geq 0$. However, Maboudou-Tchao [39] shows that the solution to the dual problem can be obtained by considering the following two cases $(i) \ C > 1/N$ and $(ii) \ C \leq 1/N$ separately.

First, let’s consider the case where $C > 1/N$. By incorporating the inequality constraints into the objective function using Lagrange multipliers, we obtain the following Lagrangian function:

$$L(\bar{R}, A, \alpha, \gamma, \xi) = \bar{R} + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i (\bar{R} + \xi_i - ||\varphi(X_i) - A||_F^2) - \sum_{i=1}^{N} \gamma_i \xi_i$$

$$= \bar{R} \left(1 - \sum_{i=1}^{N} \alpha_i \right) + \sum_{i=1}^{N} \xi_i (C - \alpha_i - \gamma_i) + \sum_{i=1}^{N} \alpha_i (||\varphi(X_i) - A||_F^2)$$

(2.62)

where $\alpha_i \geq 0$ and $\gamma_i \geq 0$ are the Lagrange multipliers.

Necessary and sufficient conditions for any optimal solution are given by the Karush-Kuhn-Tucker (KKT) optimality conditions. The KKT conditions for the SMDD are obtained by taking the derivative of the Lagrangian with respect to the primal variables $\bar{R}$, $A$, and $\xi$. 

27
Therefore, by differentiating (2.62) with respect to these variables and setting the derivatives equal to zero yields:

\[
\frac{\partial L}{\partial \bar{R}} = 0 \implies \sum_{i=1}^{N} \alpha_i = 1
\]

(2.63)

\[
\frac{\partial L}{\partial A} = 0 \implies A = \sum_{i=1}^{N} \alpha_i \varphi(X_i)
\]

(2.64)

\[
\frac{\partial L}{\partial \xi_i} = 0 \implies C - \alpha_i - \gamma_i = 0
\]

(2.65)

Considering the equations above and the non-negative nature of the Lagrange multipliers, we can safely remove the Lagrange multipliers \(\gamma_i\) from the optimization problem by imposing the following constraint:

\[
0 \leq \alpha_i \leq C, \quad i = 1, 2, ..., N
\]

(2.66)

Then, the primal variables \(\bar{R}\), \(A\), and \(\xi\) can be eliminated using substitution, which yields the following Lagrangian equation exclusively in terms of the dual variables:

\[
L(\alpha) = \sum_{i=1}^{N} \alpha_i \text{tr} \left( \varphi(X_i)^T \varphi(X_i) \right) - \sum_{i,j=1}^{N} \alpha_i \alpha_j \text{tr} \left( \varphi(X_i)^T \varphi(X_j) \right)
\]

(2.67)

subject to the constraints (2.66) and \(\sum_{i=1}^{N} \alpha_i = 1\).

After \(L\) has been minimized with respect to the primal variables \(\bar{R}, A,\) and \(\xi\) given \(\alpha\) and \(\gamma\), then it is maximized with respect to \(\alpha\). The maximization of (2.67) results in a set of dual variables \(\alpha^*\),
and the dual problem is given by:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i tr \left( \varphi(X_i)^T \varphi(X_i) \right) - \sum_{i,j=1}^{N} \alpha_i \alpha_j tr \left( \varphi(X_i)^T \varphi(X_j) \right)$$

subject to \( \sum_{i=1}^{N} \alpha_i = 1 \), \( \text{subject to} \) \( \sum_{i=1}^{N} \alpha_i = 1 \), \( 0 \leq \alpha_i \leq C, \) \( i = 1, 2, ..., N \)

There is no need to compute the features \( \varphi(X_i) \) when one knows how to compute the products directly using a valid Mercer kernel. Consequently, the inner products \( tr \left( \varphi(X_i)^T \varphi(X_j) \right) \) is replaced by a kernel function \( k(X_i, X_j) = tr \left( \varphi(X_i)^T \varphi(X_j) \right) \), which maps the training data from the original input space to a bounded spherical region in the feature space, effectively isolating outliers beyond this closed boundary.

To ensure Mercer’s theorem is satisfied, we consider the power-Euclidean distance since it yields a positive-definite Gaussian kernel [26]. The power-Euclidean metric between two matrices \( X_i \) and \( X_j \) is expressed as follows:

$$d(X_i, X_j) = \frac{1}{\alpha_0} ||X_i - X_j||_F$$

Therefore, in this work, we use the radial basis function (RBF) kernel, induced by the power-Euclidean metric with \( \alpha_0 = 1 \), defined by:

$$K(X_i, X_j) = \exp \left( -\frac{||X_i - X_j||_F}{\sigma^2} \right)$$

where \( \sigma > 0 \) is the width of the Gaussian kernel that controls the complexity of the SMDD classification boundary.
The resulting dual problem can be expressed as follows:

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i k(X_i, X_i) - \sum_{i,j=1}^{N} \alpha_i \alpha_j k(X_i, X_j)$$

subject to $$\sum_{i=1}^{N} \alpha_i = 1,$$ (2.71)

and $$0 \leq \alpha_i \leq C, \quad i = 1, 2, ..., N$$

This is a quadratic programming problem and can be solved efficiently using quadratic programming solver packages. Once the dual is maximized, the resulting Lagrange multipliers $$\alpha^*$$ can be used to calculate the center of the hypersphere $$A$$.

When a training matrix $$\varphi(X_i)$$ satisfies the condition $$||\varphi(X_i) - A||^2 < \bar{R} + \xi_i$$, then the inequality constraint is satisfied and the corresponding Lagrange multiplier will be zero ($$\alpha_i = 0$$). For training matrices satisfying the equality constraint $$||\varphi(X_i) - A||^2 = \bar{R} + \xi_i$$, then the constraint has to be enforced and the Lagrange multiplier will become larger than zero ($$\alpha_i > 0$$). The three possible scenarios that may occur for a given training matrix $$\varphi(X_i)$$ are shown below.

$$||\varphi(X_i) - a||^2 < \bar{R} \implies \alpha_i = 0, \quad \gamma_i = 0$$ (2.72)

$$||\varphi(X_i) - a||^2 = \bar{R} \implies 0 < \alpha_i < C, \quad \gamma_i > 0$$ (2.73)

$$||\varphi(X_i) - a||^2 > \bar{R} \implies \alpha_i = C, \quad \gamma_i = 0$$ (2.74)

Note that Equation (2.64) shows that the hypersphere center $$A$$ is a linear combination of the Lagrange multipliers. However, only training matrices $$\varphi(X_i)$$ with $$\alpha_i > 0$$ are needed in the data description. The training matrices $$\varphi(X_i)$$ corresponding to positive Lagrange multipliers $$\alpha_i > 0$$ are the “support matrices” (SM). The support matrices that correspond to training matrices that lie directly on the hypersphere boundary in the feature space, with $$0 < \alpha_i < C$$, are the boundary
support matrices. The other support matrices, with $\alpha_i = C$, are referred to as the non-boundary support matrices.

Next, let’s consider the second case where $C \leq 1/N$. Maboudou-Tchao [39] proves that in this scenario, the radius $\tilde{R}$ can be removed from the optimization problem resulting in the following primal problem:

$$\min_{A, \xi} \quad C \sum_{i=1}^{N} \xi_i$$

subject to $0 \leq ||\varphi(X_i) - A||_F^2 \leq \xi_i, \quad i = 1, 2, ..., N$ \hspace{1cm} (2.75)

By incorporating the set of inequality constraints into the objective function using Lagrange multipliers, we obtain the following Lagrangian function:

$$\mathcal{L}(A, \alpha, \xi) = C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i (\xi_i - ||\varphi(X_i) - A||_F^2)$$ \hspace{1cm} (2.76)

where $\alpha_i \geq 0$ are the Lagrange multipliers.

The KKT conditions for the SMDD are obtained by taking the derivative of the Lagrangian with respect to the primal variables $A$ and $\xi$. Therefore, by differentiating (2.76) with respect to these variables and setting the derivatives equal to zero yields:

$$\frac{\partial \mathcal{L}}{\partial A} = 0 \implies A = \frac{\sum_{i=1}^{N} \varphi(X_i)}{N} \hspace{1cm} (2.77)$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = 0 \implies C = \alpha_i \hspace{1cm} (2.78)$$

Therefore, in the case where $C \leq 1/N$, the optimal solution is independent of the tuning parameter $C$ and the Lagrange multipliers $\alpha$. 

31
The next step is to compute the squared radius of the hypersphere $\bar{R}$. There are two approaches to evaluating the value of $\bar{R}$. By definition, the squared radius $\bar{R}$ is the squared Euclidean distance from the center of the hypersphere $A$ to any of the support matrices on the boundary in the kernel space. Support matrices which fall outside the data description ($\alpha_i = C$) are excluded. An alternative approach is to use the set of all boundary support matrices, denoted by $N_s$, and calculating the average squared kernel distance from the center of the hypersphere to the boundary support matrices given by the following equation:

$$R^2 = \frac{1}{N_s} \sum_{s=1}^{N_s} \left( k(\mathbf{X}_s, \mathbf{X}_s) - 2 \sum_{i=1}^{N_s} \alpha_i k(\mathbf{X}_s, \mathbf{X}_i) + \sum_{i,j=1}^{N_s} \alpha_i \alpha_j k(\mathbf{X}_i, \mathbf{X}_j) \right)$$  \hspace{1cm} (2.79)

for any $\mathbf{X}_i \in SM_{<C}$, the set of support matrices which have $\alpha_i < C$.

For a previously unseen test matrix $Z$, the kernel distance statistic $d$ is computed as follows:

$$d = k(\mathbf{Z}, \mathbf{Z}) - 2 \sum_{i=1}^{N} \alpha_i k(\mathbf{X}_i, \mathbf{Z}) + \sum_{i,j=1}^{N} \alpha_i \alpha_j k(\mathbf{X}_i, \mathbf{X}_j)$$  \hspace{1cm} (2.80)

The test matrix $Z$ is classified as a target if the condition $d \leq \bar{R}$ is satisfied. Otherwise, it is classified as an outlier.
Deep Learning-Based Support Vector Methods

Considering that the distribution of the target class is often non-spherical in its native form, employing kernels to map training data into a higher-dimensional feature space is one approach to finding flexible decision boundaries for the one-class classification task. As an alternative approach to using kernels, effective boundaries can also be obtained using deep neural networks by projecting training data into latent space to learn useful feature representations of the target class.

One-Class Neural Network

One-Class Neural Network (OC-NN), proposed by Chalapathy et al. [9], is a method used for one-class classification that combines the ability of neural networks to extract useful feature representations of the data coupled with a maximum margin separating hyperplane loss function. OC-NN integrates a One-Class Support Vector Machine (OCSVM) equivalent loss into the neural network architecture.

Consider a training dataset \( X = \{x_1, x_2, \ldots, x_N\} \subseteq \mathcal{X} \), where \( N \) is the number of vectors and \( \mathcal{X} \subseteq \mathbb{R}^p \) is an original input space. For this input space, we will define a corresponding output space \( \mathcal{F} \subseteq \mathbb{R}^d \). Let \( \phi(\cdot; \mathcal{V}) \) be a simple feed forward neural network that maps \( \mathcal{X} \mapsto \mathcal{F} \) with \( L = 2 \) layers. The network consists of 1 hidden layer and a final output layer with a single output neuron. The architecture is characterized by a set of weight matrices \( \mathcal{V} = \{V^{[1]}, V^{[2]}\} \), corresponding to the weights \( V^{[\ell]} \) of layer \( \ell = \{1, 2\} \), respectively. In other words, \( \phi(x; \mathcal{V}) \) denotes the feature representation of \( x \in \mathcal{X} \) given by network \( \phi \) with parameters \( \mathcal{V} \).

The goal is to jointly learn the network parameters \( \mathcal{V} \) while separating the target class from the origin using a hyperplane that maximizes the distance between the hyperplane and the origin. This hyperplane is characterized by a normal vector \( \mathbf{w} \in \mathcal{F} \) and an offset \( \rho \in \mathbb{R} \). The formal
mathematical representation of this model for jointly learning the network parameters $V$ while finding the maximum hyperplane is given by the following unconstrained optimization problem:

$$
\min_{\rho, w, V^{[1]}} \frac{1}{2} w^T w - \rho + \frac{1}{\nu N} \sum_{i=1}^{N} \max (0, \rho - w^T \phi(x_i; V)) + \frac{1}{2} ||V^{[1]}||_F^2
$$

(2.81)

where $w$ is the normal vector perpendicular to the hyperplane, $\rho$ is the offset or bias, and the parameter $\nu \in (0, 1)$ is a tuning parameter introduced to regulate the trade off between maximizing the distance of the hyperplane from the origin and the number of training observations permitted to cross the hyperplane (Type I error). The last term in the objective introduces a weight decay regularization component applied to the hidden layer’s weight matrix, where $|| \cdot ||_F$ denotes the Frobenius norm.

Deep Support Vector Data Description

Deep Support Vector Data Description (DSVDD), proposed by Ruff et al. [48], combines the powerful feature extraction capability of deep neural networks with Support Vector Data Description for the one-class classification task. In this end-to-end framework, the goal is to encompass the training data inside a hypersphere with minimum volume while simultaneously optimizing the neural network weight parameters.

Consider a training dataset $X = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X}$, where $N$ is the number of vectors and $\mathcal{X} \subseteq \mathbb{R}^p$ is an original input space for which we want a description. A description refers to a model characterized by a closed boundary in the form of minimum volume hypersphere that distinguishes the target class from outliers. For this input space, we will define a corresponding output space $\mathcal{F} \subseteq \mathbb{R}^d$. Let $\phi(\cdot; W)$ be a neural network that maps $\mathcal{X} \mapsto \mathcal{F}$ with $L \in \mathbb{N}$ layers. The network consists of $L - 1$ hidden layers and a final output layer, and the architecture is characterized by a
set of weight matrices $\mathcal{W} = \{W^{[1]}, W^{[2]}, \ldots, W^{[L]}\}$, corresponding to the weights $W^{[\ell]}$ of layer $\ell \in \{1, 2, \ldots, L\}$, respectively. In other words, $\phi(x; \mathcal{W})$ denotes the feature representation of $x \in \mathcal{X}$ given by network $\phi$ with weight parameters $\mathcal{W}$.

The goal is to jointly learn the network parameters $\mathcal{W}$ while encapsulating the target class using a minimum volume hypersphere characterized by its center $c \in \mathcal{F}$ and radius $R > 0$ that contains all, or most, of the training set vectors. The formal mathematical representation of this model for finding the hypersphere is given by the following unconstrained optimization problem:

$$
\min_{R, \mathcal{W}} \quad R^2 + \frac{1}{\nu N} \sum_{i=1}^{N} \max \left\{ 0, \|\phi(x_i; \mathcal{W}) - c\|^2 - R^2 \right\} + \frac{\lambda}{2} \sum_{\ell=1}^{L} \|W^{[\ell]}\|^2_F
$$

(2.82)

where $c$ is the center of the hypersphere, $R$ is the radius of the hypersphere, and the parameter $\nu \in (0, 1]$ is introduced to regulate the error associated with the misclassification of the target class and the hypersphere’s volume, thereby providing control over the allowed proportion of outliers in a model. The last term in the objective introduces a weight decay regularization component applied to the network weights $\mathcal{W}$ with a tuning parameter $\lambda > 0$, where $\|\cdot\|_F$ denotes the Frobenius norm.

As suggested by Ruff et al. [48], we do not treat the hypersphere center $c$ as a learnable parameter to avoid hypersphere collapse, which results in the algorithm learning degenerate or uninformative solutions. Instead, the center is fixed as the average feature representations obtained from a small training batch upon an initial forward pass through the network $\phi(\cdot; \mathcal{W})$.

It is common to find the network parameters $\mathcal{W}$ and the hypersphere radius living on different scales. Therefore, an alternating minimization/block coordinate descent approach can be used for model training. Under this procedure, the network weight parameters $\mathcal{W}$ are trained for some $k \in \mathbb{N}$ epochs while the hypersphere radius $R$ is held fixed. Then, after every $k$—th epoch, a line search technique is employed to solve for the radius using the network weight parameters $\mathcal{W}$ of
the latest update.

For a previously unseen test vector $z$, the distance statistic $d$ is computed as follows:

$$d = ||\phi(z; \mathcal{W}^*) - c||^2$$

(2.83)

where $\mathcal{W}^*$ are the network weights of a trained model. The test vector $z$ is classified as a target if the condition $d \leq R^2$ is satisfied. Otherwise, it is classified as an outlier.
CHAPTER 3: DEEP LEAST SQUARES SUPPORT VECTOR DATA DESCRIPTION

Introduction

The success of deep learning in fields such as computer vision, natural language processing, and speech recognition, has created a strong perception that deep learning is a superior modeling approach compared to classical techniques regardless of the data problem. Given the recent achievements in the aforementioned fields using deep learning, one area that has seen much less advancement is the effectiveness of deep learning techniques on tabular data that lacks spatial or temporal structure in unsupervised settings. Since heterogeneous tabular data are one of the most commonly used forms of data in real-world settings, it is critical that the performance of deep learning-based anomaly detection algorithms be effective on this type of data. The focus on this chapter is to evaluate a novel deep learning one-class classification technique on tabular data that does not possess spatial or temporal characteristics. We propose Deep Least Squares Support Vector Data Description (DLS-SVDD), a novel one-class classifier that incorporates both deep learning and Least Squares Support Vector Data Description (LS-SVDD) in an end-to-end framework.
Methodology

Deep Least Squares Support Vector Data Description (DLS-SVDD) is a method used for one-class classification that combines the ability of deep neural networks to capture useful feature representations of the data coupled with the one-class objective to detect anomalies in complex datasets. DLS-SVDD integrates Least Squares Support Vector Data Description (LS-SVDD) into the neural network architecture, and jointly trains a neural network while minimizing the volume of a data-enclosing hypersphere. DLS-SVDD is a reformulation of Deep Support Vector Data Description (DSVDD), proposed by Ruff et al. [48], in a least squares sense.

Consider a training dataset $X = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X}$, where $N$ is the number of vectors and $\mathcal{X} \subseteq \mathbb{R}^p$ is an original input space for which we want a description. A description refers to a model characterized by a closed boundary in the form of a minimum volume hypersphere that distinguishes the target class from outliers. For this input space, we will define a corresponding output space $\mathcal{F} \subseteq \mathbb{R}^d$. Let $\phi(\cdot; \mathcal{W}, \mathcal{B})$ be a neural network that maps $\mathcal{X} \mapsto \mathcal{F}$ with $L \in \mathbb{N}$ layers. The network consists of $L - 1$ hidden layers and a final output layer, and the architecture is characterized by a set of weight matrices $\mathcal{W} = \{W^{[1]}, W^{[2]}, ..., W^{[L]}\}$ and bias vectors $\mathcal{B} = \{b^{[1]}, b^{[2]}, ..., b^{[L]}\}$, corresponding to the weights $W^{[\ell]}$ and biases $b^{[\ell]}$ of layer $\ell \in \{1, 2, ..., L\}$, respectively. In other words, $\phi(x; \mathcal{W}, \mathcal{B})$ denotes the feature representation of $x \in \mathcal{X}$ given by network $\phi$ with weights $\mathcal{W}$ and biases $\mathcal{B}$.

The goal is to jointly learn the network parameters $\mathcal{W}, \mathcal{B}$ while encapsulating the target class using a minimum volume hypersphere characterized by its center $c \in \mathcal{F}$ and radius $R > 0$ that contains all, or most, of the training set vectors. The formal mathematical representation of this model for
finding the smallest hypersphere is given by the following constrained optimization problem:

$$\min_{R, c, \xi, W, B} R^2 + \frac{1}{2\nu N} \sum_{i=1}^{N} \xi_i^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||W[\ell]||_F^2;$$

subject to $||\phi(x_i; W, B) - c||^2 = R^2 + \xi_i, \ i = 1, 2, ..., N$ (3.1)

where $c$ is the center of the hypersphere, $R$ is the radius of the hypersphere, $\xi_i$ are the error variables, and the parameter $\nu > 0$ is introduced to control the influence of the error variables, consequently regulating the error associated with the target class and the hypersphere’s volume. The last term in the objective introduces a weight decay regularization component applied to the network weights $W$ with a tuning parameter $\lambda > 0$, where $|| \cdot ||_F$ denotes the Frobenius norm.

The DLS-SVDD constrained optimization problem (3.1) can equivalently be expressed as the following unconstrained optimization problem:

$$\min_{R, c, W, B} R^2 + \frac{1}{2\nu N} \sum_{i=1}^{N} \left(||\phi(x_i; W, B) - c||^2 - R^2\right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||W[\ell]||_F^2$$ (3.2)

Note that this optimization problem uses the $\ell_2$ norm, which is smooth and differentiable, instead of the hinge loss used in the DSVDD optimization problem (2.82). Consequently, the optimization problem (3.1) is easily solved. We can define the loss function, denoted by $\mathcal{L}$, as follows:

$$\mathcal{L}(R, c, W, B) = R^2 + \frac{1}{2\nu N} \sum_{i=1}^{N} \left(||\phi(x_i; W, B) - c||^2 - R^2\right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||W[\ell]||_F^2$$ (3.3)

To optimize this problem, we use stochastic gradient descent (SGD). Define the DLS-SVDD loss function for SGD, denoted by $\mathcal{L}_s$, at a random point $x_*$ as follows:

$$\mathcal{L}_s(R, c, W, B) = R^2 + \frac{1}{2\nu} \left(||\phi(x_*; W, B) - c||^2 - R^2\right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||W[\ell]||_F^2$$ (3.4)
Theorem 3.0.1 \( R^2 \) is obtained using the following closed-form solution:

\[
R^2 = ||\phi(x^*; \mathcal{W}, \mathcal{B}) - c||^2 - \nu
\]  

(3.5)

The proof for Theorem 3.0.1 can be found in Appendix B. Note that Equation (3.5) defines the hypersphere radius based solely on the hypersphere center and neural network weights and biases. This closed-form solution is one of the differentiating factors of DLS-SVDD compared to other one-class classification boundary techniques. During training, the radius can be updated using the analytic solution every epoch, or on a predefined schedule, providing additional model flexibility. This approach differs from DSVDD, which uses a line search technique to solve for the radius, as well as LS-SVDD, which has a non-sparse solution and uses all of the observations in the training set to solve for the radius.

A given test point \( z \in \mathcal{X} \) is in the target class if the Euclidean distance of the point \( z \) to the center of the hypersphere is less than the radius. In other words, the test point \( z \) will be in the target class if the following condition is satisfied:

\[
s(z) = ||\phi(z; \mathcal{W}^*, \mathcal{B}^*) - c^*||^2 \leq R^{*2}
\]  

(3.6)

where \( \mathcal{W}^*, \mathcal{B}^*, R^{*2}, \) and \( c^* \) are the network weights, network biases, radius squared, and hypersphere center of a trained model, respectively. Furthermore, we denote the anomaly score for a given test point \( z \) by \( s(z) \).

The SGD DLS-SVDD training procedure is summarized in Algorithm 2.
Algorithm 2 DLS-SVDD SGD Algorithm

**Input:** Training dataset $X = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X}$ with $x_i \in \mathbb{R}^p$ and hyperparameters $\lambda, \nu$.

**Output:** Hypersphere radius $R^*$, hypersphere center $c^*$, network weights $W^*$, and network biases $B^*$ of a trained model.

1: **Initialize:**
   - Hypersphere radius $R$, network weights $W$, network biases $B$, and hypersphere center $c$ as the average of a small batch of training data after an initial forward pass through the network $\phi(\cdot; W, B)$.

2: **for** each epoch **do**
   3: Randomly sample a training observation $x \in \mathcal{X}$.
   4: Calculate $\phi(x; W, B)$ by forward propagation through DLS-SVDD model.
   5: Calculate $L_s$ using Equation (3.4).
   6: Update the network weights, biases, and hypersphere center using the Adam algorithm.
   7: Solve for the radius $R$ using Equation (3.5).

8: **end for**

9: **return** $R^*, c^*, W^*$ and $B^*$

In the following section, we systematically derive the mathematical equations for DLS-SVDD using a simple artificial neural network with one hidden layer. This is followed by an analogous derivation adapted to a more complex model with two hidden layers. We conclude by generalizing these derivations to apply to arbitrarily deep neural networks with $L \in \mathbb{N}$ hidden layers.

**Neural Network with One Hidden Layer**

For the mathematical derivations, we design a simple feed forward neural network with one hidden layer. This network has rectified linear unit (ReLU) activation functions in the hidden layer, denoted by $g_1(\cdot)$, and linear activation functions in the output layer, denoted by $g_2(\cdot)$.

Let $Z^{[1]}$ denote the weighted input of the first hidden layer. The output of the first hidden layer, denoted by $A^{[1]}$, is obtained after applying $g_1(\cdot)$ in an element-wise manner to the weighted input $Z^{[1]}$. For the output layer, the weighted input is denoted by $Z^{[2]}$, and the network output, defined as $\phi(\cdot; W, B) = A^{[2]}$, is obtained after $g_2(\cdot)$ is applied element-wise to $Z^{[2]}$. 
During the forward propagation phase, an observation $x_*$ is initially processed at the input layer, then passed through the hidden layer, and finally is directed to the output layer. The mathematical functions detailing this progression are shown below.

$$Z^{[1]} = W^{[1]}x_* + b^{[1]}$$  \hspace{2cm} (3.7)

$$A^{[1]} = g_1(Z^{[1]})$$  \hspace{2cm} (3.8)

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$  \hspace{2cm} (3.9)

$$\phi(x_*; W, B) = A^{[2]} = g_2(Z^{[2]})$$  \hspace{2cm} (3.10)

Through the application of the chain rule from differential calculus, backpropagation methodically propagates error signals backward through the network from the output layer to the input layer, thereby facilitating weight adjustments across the entire network. By initially defining the errors that are propagated backwards through each layer, a concise representation of the gradients for the neural network’s weights and biases can be obtained. The proofs for Proposition 3.0.1 and Proposition 3.0.2, respectively, can be found in Appendix B.

Define $\delta^{[2]}$ as the error propagated backwards through the output layer’s activation function $g_2$ as follows:

$$\delta^{[2]} = \frac{2}{\nu} \left( ||\phi(x_*; W, B) - c||^2 - R^2 \right) (\phi(x_*; W, B) - c) \circ g_2'(Z^{[2]})$$  \hspace{2cm} (3.11)

where $\circ$ denotes the Hadamard product.

Next, define $\delta^{[1]}$ as the error propagated backwards through the hidden layer’s activation function $g_1$ as follows:

$$\delta^{[1]} = W^{[2]T} \delta^{[2]} \circ g_1'(Z^{[1]})$$  \hspace{2cm} (3.12)
The gradients for the hypersphere center and neural network weight matrices, respectively, are presented in Proposition 3.0.1.

**Proposition 3.0.1**

\[
\frac{\partial L_s}{\partial c} = \frac{2}{\nu} (||\phi(x_\star; W, B) - c||^2 - R^2) (c - \phi(x_\star; W, B)) \tag{3.13}
\]

\[
\frac{\partial L_s}{\partial W^{[2]}} = \delta^{[2]} A^{[1]T} + \lambda W^{[2]} \tag{3.14}
\]

\[
\frac{\partial L_s}{\partial W^{[1]}} = \delta^{[1]} x_\star^T + \lambda W^{[1]} \tag{3.15}
\]

Furthermore, the gradients for the two bias vectors are presented in Proposition 3.0.2.

**Proposition 3.0.2**

\[
\frac{\partial L_s}{\partial b^{[2]}} = \delta^{[2]} \tag{3.16}
\]

\[
\frac{\partial L_s}{\partial b^{[1]}} = \delta^{[1]} \tag{3.17}
\]
The SGD uses the following iterative updates for neural network layers $\ell = \{1, 2\}$:

\[
W^{[\ell]}_{(t+1)} = W^{[\ell]}_{(t)} - \eta_1 \frac{\partial L_s}{\partial W^{[\ell]}_{(t)}} \\
b^{[\ell]}_{(t+1)} = b^{[\ell]}_{(t)} - \eta_2 \frac{\partial L_s}{\partial b^{[\ell]}_{(t)}} \\
c_{(t+1)} = c_{(t)} - \eta_3 \frac{\partial L_s}{\partial c_{(t)}} \\
R^2_{(t+1)} = \| \phi(x_*; W_{(t+1)}, B_{(t+1)}) - c_{(t+1)} \|^2 - \nu
\]

where $\eta_1$, $\eta_2$, and $\eta_3$ are defined as the learning rates for the network weights, network biases, and hypersphere center, respectively.

*Neural Network with Two Hidden Layers*

In this section, we consider the mathematical derivations for an artificial neural network with two hidden layers. This network has rectified linear unit (ReLU) activation functions in the hidden layers, denoted by $g_i(\cdot)$, $i = 1, 2$, and linear activation functions in the output layer, denoted by $g_3(\cdot)$.

Let $Z^{[1]}$ denote the weighted input of the first hidden layer. The output of the first hidden layer, denoted by $A^{[1]}$, is obtained after applying $g_1(\cdot)$ in an element-wise manner to the weighted input $Z^{[1]}$. Similarly, $Z^{[2]}$ represents the weighted input of the second hidden layer. The output of the second hidden layer, denoted by $A^{[2]}$, is obtained after applying $g_2(\cdot)$ element-wise to the weighted input $Z^{[2]}$. For the output layer, the weighted input is denoted by $Z^{[3]}$, and the network output, defined as $\phi(\cdot; W, B) = A^{[3]}$, is obtained after $g_3(\cdot)$ is applied element-wise to $Z^{[3]}$. 

44
During the forward propagation phase, an observation $x_*$ is initially processed at the input layer, then sequentially passed through each of the hidden layers, and finally is directed to the output layer. The mathematical functions detailing this progression are shown below.

\[
Z^{[1]} = W^{[1]} x_* + b^{[1]} \quad (3.18)
\]

\[
A^{[1]} = g_1 (Z^{[1]}) \quad (3.19)
\]

\[
Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]} \quad (3.20)
\]

\[
A^{[2]} = g_2 (Z^{[2]}) \quad (3.21)
\]

\[
Z^{[3]} = W^{[3]} A^{[2]} + b^{[3]} \quad (3.22)
\]

\[
\phi(x_*; W, B) = A^{[3]} = g_3 (Z^{[3]}) \quad (3.23)
\]

Through the application of the chain rule from differential calculus, backpropagation methodically propagates error signals backward through the network from the output layer to the input layer, thereby facilitating weight adjustments across the entire network. By initially defining the errors that are propagated backwards through each layer, a concise representation of the gradients for the neural network’s weights and biases can be obtained. The proofs for Propositions 3.0.3 and 3.0.4, respectively, can be found in Appendix B.

Define $\delta^{[3]}$ as the error propagated backwards through the output layer’s activation function $g_3$ as follows:

\[
\delta^{[3]} = \frac{2}{\nu} \left( ||\phi(x_*; W, B) - c||^2 - R^2 \right) \left( \phi(x_*; W, B) - c \right) \circ g'_3 (Z^{[3]}) \quad (3.24)
\]

where $\circ$ denotes the Hadamard product.
Next, define $\delta^{[2]}$ as the error propagated backwards through the second hidden layer’s activation function $g_2$ as follows:

$$\delta^{[2]} = W^{[3]}T \delta^{[3]} \circ g'_2(Z^{[2]}) \quad (3.25)$$

Similarly, define $\delta^{[1]}$ as the error propagated backwards through the first hidden layer’s activation function $g_1$ as follows:

$$\delta^{[1]} = W^{[2]}T \delta^{[2]} \circ g'_1(Z^{[1]}) \quad (3.26)$$

The gradients for the hypersphere center and neural network weight matrices, respectively, are presented in Proposition 3.0.3.

**Proposition 3.0.3**

$$\frac{\partial L_s}{\partial c} = \frac{2}{\nu}(||\phi(x^*_s;W,B) - c||^2 - R^2)(c - \phi(x^*_s;W,B)) \quad (3.27)$$

$$\frac{\partial L_s}{\partial W^{[3]}} = \delta^{[3]}A^{[2]T} + \lambda W^{[3]} \quad (3.28)$$

$$\frac{\partial L_s}{\partial W^{[2]}} = \delta^{[2]}A^{[1]T} + \lambda W^{[2]} \quad (3.29)$$

$$\frac{\partial L_s}{\partial W^{[1]}} = \delta^{[1]}x^*_T + \lambda W^{[1]} \quad (3.30)$$
Furthermore, the gradients for the three bias terms are presented in Proposition 3.0.4.

**Proposition 3.0.4**

\[
\frac{\partial L_s}{\partial b[3]} = \delta^{[3]} \tag{3.31}
\]
\[
\frac{\partial L_s}{\partial b[2]} = \delta^{[2]} \tag{3.32}
\]
\[
\frac{\partial L_s}{\partial b[1]} = \delta^{[1]} \tag{3.33}
\]

The SGD uses the following iterative updates for neural network layers \(\ell = \{1, 2, 3\}\):

\[
W^{[\ell]}(t+1) = W^{[\ell]}(t) - \eta_1 \frac{\partial L_s}{\partial W^{[\ell]}(t)}
\]
\[
b^{[\ell]}(t+1) = b^{[\ell]}(t) - \eta_2 \frac{\partial L_s}{\partial b^{[\ell]}(t)}
\]
\[
c(t+1) = c(t) - \eta_3 \frac{\partial L_s}{\partial c(t)}
\]
\[
R^2_{(t+1)} = \|\phi(x_*; W(t+1), B(t+1)) - c(t+1)\|^2 - \nu
\]

where \(\eta_1, \eta_2, \) and \(\eta_3\) are defined as the learning rates for the network weights, network biases, and hypersphere center, respectively.

**Neural Network with \(L\) Hidden Layers**

It is straightforward to generalize the model to incorporate deeper network architectures. Consider a neural network with \(L \in \mathbb{N}\) hidden layers \((\ell = 1, 2, ..., L)\) and one output layer \((\ell = L + 1)\). The network architecture consists of a set of weight matrices \(\mathcal{W} = \{W^{[1]}, W^{[2]}, ..., W^{[L+1]}\}\) and
bias vectors $B = \{b^{[1]}, b^{[2]}, ..., b^{[L+1]}\}$, corresponding to the weights $W^{[\ell]}$ and biases $b^{[\ell]}$ of layer $\ell \in \{1, 2, ..., L + 1\}$, respectively.

The generalized forward propagation equations for neural network layers $\ell = 1, 2, ..., L + 1$ can be expressed as follows:

\[
Z^{[\ell]} = W^{[\ell]}A^{[\ell-1]} + b^{[\ell]}
\]

\[
A^{[\ell]} = g_\ell(Z^{[\ell]})
\]

where the network input is defined as $A^{[0]} = x^*$.

For output layer $\ell = L + 1$, the error equation can be expressed as follows:

\[
\delta^{[L+1]} = \frac{2}{\nu} \left( ||\phi(x_*; W, B) - c||^2 - R^2 \right) (\phi(x_*; W, B) - c) \circ g'_{L+1}(Z^{[L+1]})
\]

where $\circ$ denotes the Hadamard product.

For hidden layers $\ell = 1, 2, ..., L$, the error equations can be expressed as follows:

\[
\delta^{[\ell]} = W^{[\ell+1]T} \delta^{[\ell+1]} \circ g'_\ell(Z^{[\ell]})
\]

For neural network layers $\ell = 1, 2, ..., L, L + 1$, the gradients of the weight and bias parameters, respectively, are succinctly represented using the equations shown below.

\[
\frac{\partial L_s}{\partial W^{[\ell]}} = \delta^{[\ell]}A^{[\ell-1]T} + \lambda W^{[\ell]}
\]

\[
\frac{\partial L_s}{\partial b^{[\ell]}} = \delta^{[\ell]}
\]
In this section, we demonstrate the flexibility of the DLS-SVDD decision boundary to adapt to real data and assume an irregular shape using a specially adapted “two moons” dataset for practical demonstration. These atypical boundary forms are adept at discerning shifts within the dataset, shifts that elliptical forms may not consistently detect due to their inherent geometric constraints. Using a modified version of the popular “two moons” dataset, we enhance the complexity of the one-class classification task by designating both moons as the target class while introducing a set of outlier observations that are uniformly sampled from a circular region situated between the two moons.

We employ a series of illustrative diagrams to capture the step-by-step evolution of the DLS-SVDD decision boundary, focusing on the moments of meaningful transformation throughout the training procedure. These visualizations highlight how the decision boundary iteratively adjusts to more accurately encapsulate the data points of the target class while excluding outliers.

The target class consists of 200 observations that are sampled from two interleaving half-circles with an added Gaussian noise that has standard deviation $\sigma = 0.04$. The outlier class is comprised of 50 uniformly sampled observations from a circle with radius $\frac{1}{5}$ that is centered at the mean of the target class. A visual representation of the data in the input space is shown in Figure 3.1.
Figure 3.1: Visualization of the simulated dataset comprised of the target class sampled from the Two Moons dataset and the outlier class uniformly sampled from a circle with radius $\frac{1}{5}$ in the input space.

The model is training exclusively on the data from the target class. In the early training stages of the DLS-SVDD model, the initial classification boundary for the target class takes an elliptical form, erroneously including the observations from the outlier class. The classification boundary after 15 epochs and 100 epochs, respectively, is shown in Figure 3.2. We can observe that after 15 epochs, the boundary assumes a shape that is starting to appear elliptical with a positive degree tilt. After 100 epochs, the boundary takes on a more defined elliptical shape. However, the boundary now has a negative degree tilt.
Figure 3.2: Visualization of the DLS-SVDD decision boundary in the input space during model training after 15 epochs (left) and 100 epochs (right).

With continued training, a notable shift towards a more accurate data description of the target class begins to manifest after 950 epochs, marked by the emergence of an opening at the lower base of the boundary as shown in Figure 3.3.

Figure 3.3: Visualization of the DLS-SVDD decision boundary in the input space during model training after 950 epochs.
Upon reaching 959 epochs, the opening in the lower extremity of the decision boundary is observed to be altering its shape, moving closer to the outlier class that lies in the region between the two moons containing the target class. Furthermore, we observe the elimination of the opening that was initially made at the bottom of the boundary. After 960 epochs, the classification boundary continues to evolve, now showcasing an expanding gap approaching the outlier class. The classification boundary after 959 and 960 epochs, respectively, is shown in Figure 3.4.

![Figure 3.4: Visualization of the DLS-SVDD decision boundary in the input space during model training after 959 epochs (left) and 960 epochs (right).](image)

At this stage, the DLS-SVDD model demonstrates its capability in learning rich feature representations that provide highly flexible decision boundaries, thereby facilitating the detection of anomalies that elliptical shapes may fail to capture. After epoch 974, we observe the emerging gap within the central area of the decision boundary beginning to encapsulate the outlier class. Following epoch 1000, this opening has effectively isolated almost all outliers from the target class. The classification boundary after 974 and 100 epochs, respectively, is shown in Figure 3.5.
Figure 3.5: Visualization of the DLS-SVDD decision boundary in the input space during model training after 974 epochs (left) and 1000 epochs (right).

The model has converged to a local minimum after 1023 epochs, resulting in a distinct separation between the target class and the outlier class, which is depicted in Figure 3.6.

Figure 3.6: Visualization of the DLS-SVDD decision boundary in the input space after training for 1023 epochs and reaching model convergence.
Alternative Objective Function

The proposed method can be simplified with an alternative approach. In the situation where the training set is assumed to contain only target class observations, which is consistent with the one-class assumption, we propose a simplified objective. In this simplified form, we remove the radius term from the optimization problem since all data representations are assumed to lie within the hypersphere boundary. The goal is still to enclose the training data in a minimum volume hypersphere with center $c \in \mathcal{F}$, except now this is achieved solely by minimizing the sum of squared errors originating from the target class representations with respect to the hypersphere.

Consider a training dataset $X = \{x_1, x_2, \ldots, x_N\} \subseteq \mathcal{X}$, where $N$ is the number of vectors and $\mathcal{X} \subseteq \mathbb{R}^p$ is an original input space for which we want a description. A description refers to a model characterized by a closed boundary in the form of a minimum volume hypersphere that distinguishes the target class from outliers. For this input space, we will define a corresponding output space $\mathcal{F} \subseteq \mathbb{R}^d$. Let $\phi(\cdot; \mathcal{W}, \mathcal{B})$ be a neural network that maps $\mathcal{X} \mapsto \mathcal{F}$ with $L \in \mathbb{N}$ layers. The network consists of $L - 1$ hidden layers and a final output layer, and the architecture is characterized by a set of weight matrices $\mathcal{W} = \{W^{[1]}, W^{[2]}, \ldots, W^{[L]}\}$ and bias vectors $\mathcal{B} = \{b^{[1]}, b^{[2]}, \ldots, b^{[L]}\}$, corresponding to the weights $W^{[\ell]}$ and biases $b^{[\ell]}$ of layer $\ell \in \{1, 2, \ldots, L\}$, respectively. In other words, $\phi(x; \mathcal{W}, \mathcal{B})$ denotes the feature representation of $x \in \mathcal{X}$ given by network $\phi$ with weights $\mathcal{W}$ and biases $\mathcal{B}$. We define the Deep Least Squares One-Class Support Vector Data Description (DLS-OCSSVDD) objective as follows:

$$
\min_{c, \mathcal{W}, \mathcal{B}} \frac{1}{2N} \sum_{i=1}^{N} \left( \|\phi(x_i; \mathcal{W}, \mathcal{B}) - c\|^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} \|W^{[\ell]}\|_F^2 \tag{3.40}
$$

where $c$ is the center of the hypersphere, and the last term in the objective introduces a weight decay regularization component applied to the network weights $\mathcal{W}$ with a tuning parameter $\lambda > 0$. 

54
where $|| \cdot ||_F$ denotes the Frobenius norm.

Define the DLS-OCSVDD loss function for SGD, denoted by $L_{s_1}$, at a random point $x_*$ as follows:

$$
L_{s_1}(c, W, B) = \frac{1}{2} \left( ||\phi(x_*; W, B) - c||^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||W^{[\ell]}||^2_F.
$$

(3.41)

For the mathematical derivations, consider a simple feed forward neural network with a single hidden layer as was previously outlined in the Methodology Section for DLS-SVDD. The proofs for Proposition 3.0.5 and Proposition 3.0.6, respectively, can be found in Appendix C.

Define $\delta^{[2]}_1$ as the error propagated backwards through the output layer’s activation function $g_2$ as follows:

$$
\delta^{[2]}_1 = 2 \left( ||\phi(x_i; W, B) - c||^2 \right) (\phi(x_*; W, B) - c) \circ g'_2(Z^{[2]})
$$

(3.42)

where $\circ$ denotes the Hadamard product.

Next, define $\delta^{[1]}$ as the error propagated backwards through the hidden layer’s activation function $g_1$ as follows:

$$
\delta^{[1]}_1 = W^{[2]^T} \delta^{[2]}_1 \circ g'_1(Z^{[1]})
$$

(3.43)

The gradients for the hypersphere center and neural network weight matrices, respectively, are presented in Proposition 3.0.5.
Proposition 3.0.5

\[
\frac{\partial L_{s1}}{\partial c} = 2 \left\| \phi(x_*, W, B) - c \right\|^2 \left( c - \phi(x_*, W, B) \right) \tag{3.44}
\]

\[
\frac{\partial L_{s1}}{\partial W^{[2]}} = \delta_1^{[2]} A^{[1]T} + \lambda W^{[2]} \tag{3.45}
\]

\[
\frac{\partial L_{s1}}{\partial W^{[1]}} = \delta_1^{[1]} x_*^T + \lambda W^{[1]} \tag{3.46}
\]

Furthermore, the gradients for the two bias terms in the neural network are presented in Proposition 3.0.6.

Proposition 3.0.6

\[
\frac{\partial L_{s1}}{\partial b^{[2]}} = \delta_1^{[2]} \tag{3.47}
\]

\[
\frac{\partial L_{s1}}{\partial b^{[1]}} = \delta_1^{[1]} \tag{3.48}
\]

The SGD uses the following iterative updates for neural network layers \( \ell = \{1, 2\} \):

\[
W^{[\ell]}_{(t+1)} = W^{[\ell]}_{(t)} - \eta_1 \frac{\partial L_{s1}}{\partial W^{[\ell]}_{(t)}}
\]

\[
b^{[\ell]}_{(t+1)} = b^{[\ell]}_{(t)} - \eta_2 \frac{\partial L_{s1}}{\partial b^{[\ell]}_{(t)}}
\]

\[
c_{(t+1)} = c_{(t)} - \eta_3 \frac{\partial L_{s1}}{\partial c_{(t)}}
\]

where \( \eta_1, \eta_2, \) and \( \eta_3 \) are defined as the learning rates for the network weights, network biases, and hypersphere center, respectively.
In this simplified approach, we no longer have a radius term as part of the model. Hence, we cannot detect outliers using the radius. Instead we can use a threshold $h$ obtained using bootstrap simulation.

A given test point $z \in \mathcal{X}$ is in the target class if the distance of the point $z$ to the center of the hypersphere is less than a threshold $h$. In other words, the test point $z$ is classified as a target if the following condition holds:

$$s_1(z) = ||\phi(z; \mathcal{W}^*, \mathcal{B}^*) - \mathbf{c}^*||^2 \leq h$$  \hspace{1cm} (3.49)$$

where $\mathcal{W}^*$, $\mathcal{B}^*$, and $\mathbf{c}^*$ are the network weights, network biases, and hypersphere center of a trained model, respectively. Furthermore, $s_1(z)$ denotes the anomaly score for a given test point $z$.

The SGD DLS-OCSVDD training procedure is summarized in Algorithm 3.

**Algorithm 3 DLS-OCSVDD SGD Algorithm**

**Input**: Training dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subseteq \mathcal{X}$ with $\mathbf{x}_i \in \mathbb{R}^p$ and hyperparameter $\lambda$.

**Output**: Hypersphere center $\mathbf{c}^*$, network weights $\mathcal{W}^*$, and network biases $\mathcal{B}^*$ of a trained model.

1. **Initialize**:  
   Network weights $\mathcal{W}$, network biases $\mathcal{B}$, and hypersphere center $\mathbf{c}$ as the average of a small batch of targets after forward pass through network $\phi(\cdot; \mathcal{W}, \mathcal{B})$.

2. **for** each epoch **do**
   3. Randomly sample a training observation $\mathbf{x} \in \mathcal{X}$.
   4. Calculate $\phi(\mathbf{x}; \mathcal{W}, \mathcal{B})$ by forward propagation through DLS-OCSVDD model.
   5. Calculate $L_{s_1}$ using Equation (3.41).
   6. Update the network weights, biases, and hypersphere center using the Adam algorithm.

7. **end for**

8. **return** $\mathbf{c}^*$, $\mathcal{W}^*$, and $\mathcal{B}^*$
Hyperparameter Tuning

The training scheme for DLS-SVDD provides users with considerable flexibility in choosing suitable hyperparameters \( \nu \) and \( \lambda \). However, there are no outliers available during the training procedure under the one-class assumption. Therefore, conventional tuning procedures used for binary classification will typically fail in this setting. The two types of model tuning procedures commonly used in one-class classification scenarios are pseudo outlier generation-based methods and heuristics-based methods. For the hyperparameter tuning of DLS-SVDD, we suggest the border-edge pattern selection (BEPS) algorithm proposed by Wang et al. [57].

BEPS is an approach for hyperparameter tuning that is specifically designed for one-class classification problems, where only observations originating from the target class are available during model training. Under this self-adaptive data shifting procedure, a negative shifting mechanism is used by incorporating edge pattern detection (EPD) [32] along with the calculation of negative data density gradients [15].

With this approach, the training set can be used to generate two new datasets to facilitate training the one-class classifier. The first is a pseudo outlier dataset resulting from negative shifting. The second is a pseudo target class dataset obtained from positive shifting. First, the EPD dataset is found using the target class observations. Then, the pseudo outlier training set using the EPD dataset is generated. Lastly, the pseudo target class is generated. The steps for the BEPS algorithm are outlined in Algorithm 4.

After the creation of the pseudo target class and pseudo outlier datasets, training of the DLS-SVDD model proceeds exclusively with the pseudo targets. In the testing phase, the model’s accuracy in classifying both pseudo targets and pseudo outliers is assessed. The average accuracy from these two measures is then computed. The model’s hyperparameters are selected based on achieving the
maximal average accuracy. The DLS-SVDD hyperparameter selection procedure is summarized in Algorithm 5.

**Algorithm 4** Border Edge Pattern Selection (BEPS) Algorithm.

**Input:** A training dataset $X_{\text{target}} = \{x_1, x_2, \ldots, x_N\} \subseteq \mathcal{X}$ with $x_i \in \mathbb{R}^p$.

**Output:** Edge pattern dataset $X_{\text{edge}}$, pseudo outlier dataset $X_{\text{outlier}}$, and pseudo target class dataset $X_{\text{target}}^*$.

---

**Phase 1 – Edge Pattern Detection**

1: Initialize:
   - Calculate number of nearest neighbors $k = \lceil 5 \log_{10} N \rceil$
   - Set threshold $T = 0.1$
   - Set $X_{\text{edge}} \neq \emptyset$

2: for $i = 1, 2, \ldots, N$ do
   3: Calculate $k$-nn direction vectors $v_{ij} = \frac{(x_i - x_j)}{||x_i - x_j||}$, $j = 1, 2, \ldots, k$
   4: Approximate normal vector $n_i = \sum_{j=1}^{k} v_{ij}$
   5: Calculate $\theta_{ij} = v_{ij}^T \cdot n_i$, $j = 1, 2, \ldots, k$
   6: Calculate $l_i = \frac{1}{k} \sum_{j=1}^{k} I(\theta_{ij} \geq 0)$
   7: if $l_i \geq 1 - T$ then
      8: $X_{\text{edge}} = X_{\text{edge}} \cup x_i$
   9: end if
10: end for
11: return $X_{\text{edge}} = \{x_1, x_2, \ldots, x_{N_1}\}$ where $N_1 \leq N$.

**Phase 2 – Pseudo Outlier Class Generation**

12: Initialize:
   - Set $X_{\text{outlier}} \neq \emptyset$

13: for $i = 1, 2, \ldots, N_1$ do
   14: Calculate the negative shifting magnitude: $\ell_{ns}^{(i)} = \frac{1}{k} \sum_{j=1}^{k} ||x_i - x_j||$
   15: end for
16: Calculate the average negative shifting magnitude: $\bar{\ell}_{ns} = \frac{1}{N_1} \sum_{i=1}^{N_1} \ell_{ns}^{(i)}$
17: for $i = 1, 2, \ldots, N_1$ do
   18: Calculate negative shifting direction: $-\nabla p(x_i) \approx n_i = \sum_{j=1}^{k} \frac{x_i - x_j}{||x_i - x_j||}$
   19: $x_{\text{outlier}}^{(i)} = x_i + \frac{n_i}{||n_i||} \cdot \bar{\ell}_{ns}$
   20: $X_{\text{outlier}} = X_{\text{outlier}} \cup x_{\text{outlier}}^{(i)}$
21: end for
22: return $X_{\text{outlier}}$

**Phase 3 – Pseudo Target Class Generation**

23: Initialize:
   - $X_{\text{target}}^* \neq \emptyset$

---

59
for $i = 1, 2, ..., N$ do
25: Calculate positive shifting direction: $\nabla p(x_i) \cong -n_i = -\sum_{j=1}^{k} \frac{x_i - x_j}{||x_i - x_j||}$
26: for $j = 1, 2, ..., k$ do
27: if $j = 1$ then
28: $x_{ij}^{min} = \frac{\nabla p(x_i)}{||\nabla p(x_i)||} \cdot (x_{ij} - x_i)$
29: else
30: $projection_{ij} = \frac{\nabla p(x_i)}{||\nabla p(x_i)||} \cdot (x_{ij} - x_i)$
31: if $projection_{ij} < x_{ij}^{min}$ then
32: $x_{ij}^{min} = projection_{ij}$
33: end if
34: end if
35: end for
36: $x_{target}^{(i)} = x_i + \frac{\nabla p(x_i)}{||\nabla p(x_i)||} \cdot (x_{ij}^{min} - x_i) \cdot \frac{\nabla p(x_i)}{||\nabla p(x_i)||}$
37: $X^\ast_{target} = X^\ast_{target} \cup x_{target}^{(i)}$
38: end for
39: return $X^\ast_{target}$

Algorithm 5 DLS-SVDD Hyperparameter Selection.

Input: Training dataset $X_{target} = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X}$ with $x_i \in \mathbb{R}^p$ and hyperparameter range $\nu_{range}, \lambda_{range}$.

Output: Optimal hyperparameter combination ($\nu_{opt}, \lambda_{opt}$).

1: Implement EPD in Phase 1 of BEPS Algorithm 4.
2: Generate pseudo outlier set $X_{outlier}$ in Phase 2 of BEPS Algorithm 4.
3: Generate pseudo target class training set $X^\ast_{target}$ in Phase 3 of BEPS Algorithm 4.
4: Set $Error_{best} = \infty$.
5: for each hyperparameter combination ($\nu, \lambda$) from $\nu_{range}, \lambda_{range}$ do
6: Train a DLS-SVDD model with hyperparameter ($\nu, \lambda$).
7: Estimate the error rate on the outlier class $Error_{outlier}$ by $X_{outlier}$.
8: Estimate the error rate on the target class $Error_{target}$ by $X^\ast_{target}$.
9: Calculate the current overall error rate.
10: $Error = 0.5 \cdot Error_{outlier} + 0.5 \cdot Error_{target}$
11: if $Error_{best} > Error$ then
12: $(\nu_{opt}, \lambda_{opt}) = (\nu, \lambda)$
13: end if
14: end for
15: return $(\nu_{opt}, \lambda_{opt})$
We evaluate the performance of the proposed method with other existing methods using an evaluation scheme based on the probability of a Type II error. Note that for DLS-SVDD, one can choose to use the radius $R^2$ to detect outliers. However, if the user is more interested in controlling the Type I error rates, it is suitable to use a threshold, $h$, based on the probability of Type I error instead.

In this context, we are interested in two distinct components. The first is a distance measure that quantifies the spatial separation between a new observation $z$ to the target class. The second is a discernment threshold, denoted by $h$, intended to serve as a comparative benchmark against the aforementioned distance metric.

A previously unseen observation will be classified as a target if the distance measure is smaller than a threshold $h$. In this framework, the absence of observations originating from the outlier class during the training stage prevents the assessment of the Type II error rate. Therefore, only the target class observations that are erroneously categorized as outliers (Type I error) can be controlled. In order to evaluate the performance of DLS-SVDD, we can use the probability of Type I error, denoted by $\alpha$, and the probability of Type II error, denoted by $\beta$.

Consider the availability of a training set, consisting of $p$-dimensional vectors for $i = 1, 2, ..., N$ assumed to originate from the target class. For each newly encountered vector $z$, the distance statistic $d$ is computed as follows:

$$d = ||\phi(z; W^*, B^*) - c^*||^2$$ (3.50)
where $\mathcal{W}^*$, $B^*$, and $c^*$ are the network weights, network biases, and hypersphere center of a trained model, respectively. The vector $z$ is classified as a target if the condition $d \leq h$ is satisfied, where the threshold $h$ is chosen to achieve a specified Type I error rate $\alpha$. Otherwise, it is classified as an outlier.

As previously mentioned, the radius $R^2$ can be used as a discernment threshold to compare against the distance from an unseen observation to the target class. However, it is not possible to control the Type I error if the radius is used. Instead, a threshold $h$ can be used based on the Type I error rate. Values of $h$ are obtained by using the bootstrap simulation technique summarized in Algorithm 6.

**Algorithm 6 DLS-SVDD Bootstrap Algorithm for Threshold $h$**

**Input**: Training dataset $X = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X}$ with $x_i \in \mathbb{R}^p$, $B$ bootstrap samples, network weights $\mathcal{W}^*$, network biases $B^*$, and hypersphere center $c^*$ of a trained model.

**Output**: A threshold $h$.

1: for $x_i \in \mathcal{X}$ do
2: find $d_i = ||\phi(x_i; \mathcal{W}^*, B^*) - c^*||^2$
3: end for
4: for $b \leftarrow 1, B$ do
5:   (i) Draw a bootstrap sample of size $N$ from the set of $N$ $d$ statistics.
6:   (ii) If $\alpha$ is the desired Type I error, determine the $100 \times (1 - \alpha)^{th}$ percentile value.
7: end for
8: Calculate the threshold $h$ by taking an average of $B$ $100 \times (1 - \alpha)^{th}$ percentile values.
9: return $h$
To assess the performance of DLS-SVDD, we evaluate the probability of a Type II error following a data distribution shift, considering both low- and high-dimensional scenarios with normal and non-normal distributions. In the first scenario, we consider a data distribution shift in the mean vector using uncorrelated data generated from the multivariate normal distribution. The second scenario examines a change in the mean vector using correlated multivariate normal data. In the third scenario, we consider a data distribution shift in the mean vector generated from the multivariate skew-normal distribution. In the fourth scenario, we compare the model performance of DLS-SVDD following a change in the mean vector using different activation functions in the hidden layers. In the final scenario, we evaluate the robustness of DLS-SVDD when outliers are present in the training set by varying the level of contamination.

The network architecture under consideration for DLS-SVDD is an artificial neural network with 2 hidden layers that has bias incorporated in both hidden layers and the output layer. The hidden layers and output layer each have 15 neurons, respectively. Leaky ReLU activation functions with leakiness $\alpha = 0.1$ are used in the hidden layers, and linear activation functions are used in the output layer. The hypersphere center $c$ is initialized as the average feature representation of a small batch of training data after performing an initial forward pass through the network. The hypersphere radius $R$ is initialized by randomly choosing a value in the range $[0.1, 1]$. DLS-SVDD is trained using stochastic gradient descent (SGD) and Adam optimizer, proposed by Kingma and Ba [28], to optimize the network weights and biases as well as the hypersphere center $c$, which we treat as a learnable parameter that is updated every epoch. The radius is also updated every epoch using the closed-form solution from Equation (3.5).

All models are implemented using Python 3.10.12, PyTorch-2.0.1, and CUDA 11.8. The experiments are carried out on a PC with an NVIDIA GeForce RTX 3080 GPU and 128 GB of RAM.
Uncorrelated Multivariate Normal Data

In the first scenario, we evaluate the probability of a Type II error following a data distribution shift using uncorrelated multivariate normal data. We compare the model performance of DLS-SVDD against DSVDD, LS-SVDD, and SVDD.

The target set is generated from a multivariate normal distribution with the mean vector set to zero $\mu_0 = 0$ and identity covariance matrix $\Sigma_0 = I_p$. We investigate both low-dimensional and high-dimensional scenarios. For the low-dimensional setting, we consider dimensions $p = 5$ and $p = 10$ where $N = 100$ is the number of observations originating from the target class. For the high-dimensional setting, we consider $p = 100$ dimensions where $N = 40$ is the number of observations originating from the target class.

After the model was trained on data originating solely from the target class, the evaluation of anomalous observations was considered by changing the zero mean vector $\mu_0 = 0$ from the target set’s underlying distribution to a vector having $\delta$ in its first component while the other vector components remained unchanged. If the data generating process has a distribution shift from $N(\mu_0, \Sigma_0)$ to $N(\mu_1, \Sigma_0)$, we are interested in the speed at which the algorithm can detect the change. Consequently, the performance following any step change in the mean can be modeled by changing a single component of the mean vector as follows: set the target set’s mean vector to $\mu_0 = 0$ and the covariance matrix to $\Sigma_0 = I_p$. Thereafter, by adding a shift $\delta$ to the first component of each vector we can explore all possible step changes in the mean vector.
Low-dimensional setting: $p = 5$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 5$ and $N = 100$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.7 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DLS-SVDD</th>
<th>DSVDD</th>
<th>LS-SVDD</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.92</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.91</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.90</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.87</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.85</td>
<td>0.85</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.83</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.81</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.9</td>
<td>0.77</td>
<td>0.78</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td>1.0</td>
<td>0.74</td>
<td>0.75</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td>1.5</td>
<td>0.57</td>
<td>0.58</td>
<td>0.84</td>
<td>0.84</td>
</tr>
<tr>
<td>2.0</td>
<td>0.36</td>
<td>0.38</td>
<td>0.73</td>
<td>0.73</td>
</tr>
<tr>
<td>2.5</td>
<td>0.20</td>
<td>0.21</td>
<td>0.57</td>
<td>0.58</td>
</tr>
<tr>
<td>3.0</td>
<td>0.09</td>
<td>0.10</td>
<td>0.40</td>
<td>0.41</td>
</tr>
<tr>
<td>3.5</td>
<td>0.03</td>
<td>0.04</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>4.0</td>
<td>0.01</td>
<td>0.01</td>
<td>0.12</td>
<td>0.13</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.05</td>
<td>0.05</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.02</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Figure 3.7: DLS-SVDD simulation results for $p = 5$ with uncorrelated multivariate normal data.

The LS-SVDD and SVDD exhibit similar levels of performance, a pattern also seen between DLS-SVDD and DSVDD. However, DLS-SVDD and DSVDD stand out by demonstrating superior performance over LS-SVDD and SVDD.
Low-dimensional setting: $p = 10$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 10$ and $N = 100$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.8 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DLS-SVDD</th>
<th>DSVDD</th>
<th>LS-SVDD</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.92</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.91</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.89</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.88</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.86</td>
<td>0.85</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.84</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.81</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.9</td>
<td>0.78</td>
<td>0.79</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>1.0</td>
<td>0.75</td>
<td>0.76</td>
<td>0.92</td>
<td>0.93</td>
</tr>
<tr>
<td>1.5</td>
<td>0.58</td>
<td>0.61</td>
<td>0.87</td>
<td>0.88</td>
</tr>
<tr>
<td>2.0</td>
<td>0.39</td>
<td>0.44</td>
<td>0.79</td>
<td>0.81</td>
</tr>
<tr>
<td>2.5</td>
<td>0.23</td>
<td>0.27</td>
<td>0.67</td>
<td>0.70</td>
</tr>
<tr>
<td>3.0</td>
<td>0.11</td>
<td>0.14</td>
<td>0.51</td>
<td>0.56</td>
</tr>
<tr>
<td>3.5</td>
<td>0.05</td>
<td>0.06</td>
<td>0.35</td>
<td>0.40</td>
</tr>
<tr>
<td>4.0</td>
<td>0.02</td>
<td>0.03</td>
<td>0.21</td>
<td>0.25</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.01</td>
<td>0.11</td>
<td>0.14</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.04</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Figure 3.8: DLS-SVDD simulation results for $p = 10$ with uncorrelated multivariate normal data.

The performance of DLS-SVDD and DSVDD exceeds that of LS-SVDD and SVDD. Moreover, in the range of $1.5 \leq \delta \leq 3.0$, DLS-SVDD outperforms DSVDD by a small margin. Similarly, LS-SVDD marginally outperforms SVDD when $2.0 \leq \delta \leq 5.0$. 

66
High-dimensional setting: $p = 100$ and $N = 40$

In this section we present simulation results for a distribution shift where $p = 100$ and $N = 40$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate $15,000$ independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.9 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DLS-SVDD</th>
<th>DSVDD</th>
<th>LS-SVDD</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.92</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.91</td>
<td>0.90</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.89</td>
<td>0.88</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.6</td>
<td>0.88</td>
<td>0.87</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.86</td>
<td>0.85</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.84</td>
<td>0.84</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.9</td>
<td>0.75</td>
<td>0.75</td>
<td>0.93</td>
<td>0.92</td>
</tr>
<tr>
<td>1.0</td>
<td>0.64</td>
<td>0.63</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>1.5</td>
<td>0.51</td>
<td>0.50</td>
<td>0.88</td>
<td>0.87</td>
</tr>
<tr>
<td>2.0</td>
<td>0.37</td>
<td>0.37</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td>2.5</td>
<td>0.26</td>
<td>0.26</td>
<td>0.77</td>
<td>0.77</td>
</tr>
<tr>
<td>3.0</td>
<td>0.16</td>
<td>0.16</td>
<td>0.69</td>
<td>0.68</td>
</tr>
<tr>
<td>3.5</td>
<td>0.09</td>
<td>0.09</td>
<td>0.59</td>
<td>0.58</td>
</tr>
<tr>
<td>4.0</td>
<td>0.04</td>
<td>0.04</td>
<td>0.48</td>
<td>0.47</td>
</tr>
<tr>
<td>4.5</td>
<td>0.04</td>
<td>0.04</td>
<td>0.48</td>
<td>0.47</td>
</tr>
<tr>
<td>5.0</td>
<td>0.04</td>
<td>0.04</td>
<td>0.48</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Figure 3.9: DLS-SVDD simulation results for $p = 100$ with uncorrelated multivariate normal data.

The DLS-SVDD and DSVDD stand out for their better performance compared to LS-SVDD and SVDD. Furthermore, the performance of DLS-SVDD and DSVDD is nearly indistinguishable, paralleling the similarity in performance between LS-SVDD and SVDD.
In the second scenario, we evaluate the probability of a Type II error following a data distribution shift using correlated multivariate normal data. We compare the model performance of DLS-SVDD against DSVDD, LS-SVDD, and SVDD.

The target set is generated from a multivariate normal distribution with the mean vector set to zero $\mu_0 = 0$ and covariance matrix $\Sigma_0$ such that $\sigma_{ii} = 2$ for $i = 1, 2, \ldots, p$ and $\sigma_{ij} = 0.7^{|i-j|}$, $i \neq j$.

We investigate both low-dimensional and high-dimensional scenarios. For the low-dimensional setting, we consider dimensions $p = 5$ and $p = 10$ where $N = 100$ is the number of observations originating from the target class. For the high-dimensional setting, we consider $p = 100$ dimensions where $N = 40$ is the number of observations originating from the target class.

After the model was trained on data originating solely from the target class, the evaluation of anomalous observations was considered by changing the zero mean vector $\mu_0 = 0$ from the target set’s underlying distribution to a vector having $\delta$ in its first component while the other vector components remained unchanged. If the data generating process has a distribution shift from $N(\mu_0, \Sigma_0)$ to $N(\mu_1, \Sigma_0)$, we are interested in the speed at which the algorithm can detect the change. Consequently, the performance following any step change in the mean can be modeled by changing a single component of the mean vector as follows: set the target set’s mean vector to $\mu_0 = 0$ and the covariance matrix to $\Sigma_0$. Thereafter, by adding a shift $\delta$ to the first component of each vector we can explore all possible step changes in the mean vector.
Low-dimensional setting: \( p = 5 \) and \( N = 100 \)

In this section we present simulation results for a distribution shift where \( p = 5 \) and \( N = 100 \). For this scenario, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.10 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>DLS-SVDD</th>
<th>DSVDD</th>
<th>LS-SVDD</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.93</td>
<td>0.92</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.4</td>
<td>0.92</td>
<td>0.91</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.5</td>
<td>0.90</td>
<td>0.89</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.6</td>
<td>0.89</td>
<td>0.88</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>0.7</td>
<td>0.87</td>
<td>0.87</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>0.8</td>
<td>0.86</td>
<td>0.85</td>
<td>0.93</td>
<td>0.92</td>
</tr>
<tr>
<td>0.9</td>
<td>0.84</td>
<td>0.83</td>
<td>0.92</td>
<td>0.91</td>
</tr>
<tr>
<td>1.0</td>
<td>0.74</td>
<td>0.73</td>
<td>0.89</td>
<td>0.86</td>
</tr>
<tr>
<td>1.5</td>
<td>0.62</td>
<td>0.61</td>
<td>0.83</td>
<td>0.80</td>
</tr>
<tr>
<td>2.0</td>
<td>0.48</td>
<td>0.46</td>
<td>0.76</td>
<td>0.71</td>
</tr>
<tr>
<td>2.5</td>
<td>0.33</td>
<td>0.33</td>
<td>0.66</td>
<td>0.60</td>
</tr>
<tr>
<td>3.0</td>
<td>0.22</td>
<td>0.21</td>
<td>0.54</td>
<td>0.47</td>
</tr>
<tr>
<td>3.5</td>
<td>0.12</td>
<td>0.12</td>
<td>0.41</td>
<td>0.34</td>
</tr>
<tr>
<td>4.0</td>
<td>0.06</td>
<td>0.07</td>
<td>0.29</td>
<td>0.23</td>
</tr>
<tr>
<td>4.5</td>
<td>0.03</td>
<td>0.03</td>
<td>0.19</td>
<td>0.14</td>
</tr>
<tr>
<td>5.0</td>
<td>0.03</td>
<td>0.03</td>
<td>0.19</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Figure 3.10: DLS-SVDD simulation results for \( p = 5 \) with correlated multivariate normal data.

The performance between DLS-SVDD and DSVDD is nearly indistinguishable, both significantly outperforming LS-SVDD and SVDD. Additionally, SVDD achieves better performance compared to LS-SVDD in the range of \( 1.5 \leq \delta \leq 5.0 \).
In this section we present simulation results for a distribution shift where \( p = 10 \) and \( N = 100 \). For this scenario, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.11 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>DLS-SVDD</th>
<th>DSVDD</th>
<th>LS-SVDD</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.92</td>
<td>0.92</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.91</td>
<td>0.91</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.90</td>
<td>0.89</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.89</td>
<td>0.89</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.88</td>
<td>0.87</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.9</td>
<td>0.86</td>
<td>0.86</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>1.0</td>
<td>0.86</td>
<td>0.84</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>1.5</td>
<td>0.77</td>
<td>0.76</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>2.0</td>
<td>0.67</td>
<td>0.64</td>
<td>0.87</td>
<td>0.86</td>
</tr>
<tr>
<td>2.5</td>
<td>0.56</td>
<td>0.50</td>
<td>0.82</td>
<td>0.80</td>
</tr>
<tr>
<td>3.0</td>
<td>0.43</td>
<td>0.37</td>
<td>0.75</td>
<td>0.72</td>
</tr>
<tr>
<td>3.5</td>
<td>0.31</td>
<td>0.25</td>
<td>0.64</td>
<td>0.61</td>
</tr>
<tr>
<td>4.0</td>
<td>0.21</td>
<td>0.16</td>
<td>0.53</td>
<td>0.49</td>
</tr>
<tr>
<td>4.5</td>
<td>0.13</td>
<td>0.09</td>
<td>0.41</td>
<td>0.37</td>
</tr>
<tr>
<td>5.0</td>
<td>0.07</td>
<td>0.04</td>
<td>0.29</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Figure 3.11: DLS-SVDD simulation results for \( p = 10 \) with correlated multivariate normal data.

The SVDD has a marginal performance lead over LS-SVDD between \( 2.5 \leq \delta \leq 5.0 \), a pattern also observed as DSVDD narrowly outperforms DLS-SVDD when \( 2.0 \leq \delta \leq 5.0 \). Additionally, DLS-SVDD and DSVDD achieve better overall performance compared to LS-SVDD and SVDD.
High-dimensional setting: $p = 100$ and $N = 40$

In this section we present simulation results for a distribution shift where $p = 100$ and $N = 40$. For the high-dimensional simulations, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0, 5.5, 6.0, ..., 9.5, 10.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.12 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DLS-SVDD</th>
<th>DSVDD</th>
<th>LS-SVDD</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.93</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.92</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.92</td>
<td>0.93</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.91</td>
<td>0.92</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.7</td>
<td>0.91</td>
<td>0.92</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.90</td>
<td>0.91</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.9</td>
<td>0.89</td>
<td>0.91</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>1.0</td>
<td>0.89</td>
<td>0.90</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>1.5</td>
<td>0.85</td>
<td>0.86</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>2.0</td>
<td>0.81</td>
<td>0.81</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>2.5</td>
<td>0.75</td>
<td>0.76</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>3.0</td>
<td>0.69</td>
<td>0.68</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td>3.5</td>
<td>0.61</td>
<td>0.61</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>4.0</td>
<td>0.54</td>
<td>0.52</td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td>4.5</td>
<td>0.47</td>
<td>0.44</td>
<td>0.87</td>
<td>0.87</td>
</tr>
<tr>
<td>5.0</td>
<td>0.38</td>
<td>0.36</td>
<td>0.84</td>
<td>0.84</td>
</tr>
<tr>
<td>5.5</td>
<td>0.31</td>
<td>0.29</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>6.0</td>
<td>0.24</td>
<td>0.22</td>
<td>0.76</td>
<td>0.75</td>
</tr>
<tr>
<td>6.5</td>
<td>0.19</td>
<td>0.16</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>7.0</td>
<td>0.13</td>
<td>0.12</td>
<td>0.64</td>
<td>0.63</td>
</tr>
<tr>
<td>7.5</td>
<td>0.10</td>
<td>0.08</td>
<td>0.58</td>
<td>0.57</td>
</tr>
<tr>
<td>8.0</td>
<td>0.07</td>
<td>0.05</td>
<td>0.50</td>
<td>0.49</td>
</tr>
<tr>
<td>8.5</td>
<td>0.04</td>
<td>0.03</td>
<td>0.43</td>
<td>0.42</td>
</tr>
<tr>
<td>9.0</td>
<td>0.02</td>
<td>0.02</td>
<td>0.35</td>
<td>0.34</td>
</tr>
<tr>
<td>9.5</td>
<td>0.02</td>
<td>0.01</td>
<td>0.27</td>
<td>0.26</td>
</tr>
<tr>
<td>10.0</td>
<td>0.01</td>
<td>0.01</td>
<td>0.20</td>
<td>0.19</td>
</tr>
</tbody>
</table>

Figure 3.12: DLS-SVDD simulation results for $p = 100$ with correlated multivariate normal data.

DLS-SVDD and DSVDD achieve similar performance results, paralleled by the similarity between LS-SVDD and SVDD. Also, both DLS-SVDD and DSVDD outperforms LS-SVDD and SVDD.
**Multivariate Skew-Normal Data**

**Definition 3.0.1** (Azzalini and Capitanio [4]) A \( p \)-dimensional random vector \( z \) is said to have a multivariate skew-normal distribution, denoted by \( z \sim SN_p(\Omega, \gamma) \), if it is continuous with density function

\[
f(z) = 2\Phi_p(z; \Omega)\phi(\gamma^T z), \quad z \in \mathbb{R}^p,
\]

where \( \phi_p(z; \Omega) \) is the \( p \)-dimensional normal density with zero mean and correlation matrix \( \Omega \), \( \Phi(\cdot) \) is the \( N(0, 1) \) distribution function, and \( \gamma \) is a \( p \)-dimensional vector of shape parameters.

The cumulant generating function is

\[
K(t) = \frac{1}{2} t^T \Omega t + \log \{2\Phi(\omega^T t)\}
\]

where

\[
\omega = \frac{1}{(1 + \gamma^T \Omega \gamma)^{1/2}} \Omega \gamma
\]

Hence the mean vector and covariance matrix are

\[
\mu_z = E[z] = \left(\frac{2}{\pi}\right)^{1/2} \omega, \quad Cov[z] = \Omega - \mu_z \mu_z^T = \Omega - \left(\frac{2}{\pi}\right) \omega^T \omega \tag{3.51}
\]

If we let \( x = z + \mu - \mu_z = z + \mu - (2/\pi)^{1/2} \omega \), \( x \) has a multivariate skew-normal distribution with

\[
\mu_x = E[x] = \mu, \quad Cov[x] = \Omega - (2/\pi)\omega \omega^T \tag{3.52}
\]
In the third scenario, we evaluate the probability of a Type II error following a data distribution shift using multivariate skew-normal data. We compare the model performance of DLS-SVDD against DSVDD, LS-SVDD, and SVDD.

The target set is generated from a multivariate skew-normal distribution with the mean vector set to zero $\mu_0 = 0$, identity covariance matrix $\Omega_0 = I_p$, and shape parameter $\gamma_0 = (\gamma, \gamma, \ldots, \gamma)^T$ with $\gamma = 2$. We investigate both low-dimensional and high-dimensional scenarios. For the low-dimensional setting, we consider dimensions $p = 5$ and $p = 10$ where $N = 100$ is the number of observations originating from the target class. For the high-dimensional setting, we consider $p = 100$ dimensions where $N = 40$ is the number of observations originating from the target class.

After the model was trained on data originating solely from the target class, the evaluation of anomalous observations was considered by changing the zero mean vector $\mu_0 = 0$ from the target set’s underlying distribution to a vector having $\delta$ in its first component while the other vector components remained unchanged.
Low-dimensional setting: \( p = 5 \) and \( N = 100 \)

In this section we present simulation results for a distribution shift where \( p = 5 \) and \( N = 100 \). For this scenario, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.13 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>Type II Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DLS-SVDD</td>
</tr>
<tr>
<td>0.0</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
</tr>
<tr>
<td>0.4</td>
<td>0.90</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
</tr>
<tr>
<td>0.6</td>
<td>0.86</td>
</tr>
<tr>
<td>0.7</td>
<td>0.85</td>
</tr>
<tr>
<td>0.8</td>
<td>0.83</td>
</tr>
<tr>
<td>0.9</td>
<td>0.80</td>
</tr>
<tr>
<td>1.0</td>
<td>0.78</td>
</tr>
<tr>
<td>1.5</td>
<td>0.65</td>
</tr>
<tr>
<td>2.0</td>
<td>0.49</td>
</tr>
<tr>
<td>2.5</td>
<td>0.32</td>
</tr>
<tr>
<td>3.0</td>
<td>0.20</td>
</tr>
<tr>
<td>3.5</td>
<td>0.10</td>
</tr>
<tr>
<td>4.0</td>
<td>0.04</td>
</tr>
<tr>
<td>4.5</td>
<td>0.02</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Figure 3.13: DLS-SVDD simulation results for \( p = 5 \) with multivariate skew-normal data.

The DLS-SVDD exhibits comparable performance to DSVDD with negligible differences, a pattern also seen between LS-SVDD and SVDD, which have nearly indistinguishable performance results. Also, DLS-SVDD and DSVDD has better performance compared to LS-SVDD and SVDD.
Low-dimensional setting: $p = 10$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 10$ and $N = 100$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.14 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DLS-SVDD</th>
<th>DSVDD</th>
<th>LS-SVDD</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.92</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.91</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.4</td>
<td>0.90</td>
<td>0.90</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.88</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.6</td>
<td>0.86</td>
<td>0.86</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.7</td>
<td>0.85</td>
<td>0.85</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.8</td>
<td>0.83</td>
<td>0.82</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td>0.9</td>
<td>0.81</td>
<td>0.80</td>
<td>0.92</td>
<td>0.91</td>
</tr>
<tr>
<td>1.0</td>
<td>0.77</td>
<td>0.77</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>1.5</td>
<td>0.62</td>
<td>0.62</td>
<td>0.86</td>
<td>0.85</td>
</tr>
<tr>
<td>2.0</td>
<td>0.45</td>
<td>0.43</td>
<td>0.77</td>
<td>0.76</td>
</tr>
<tr>
<td>2.5</td>
<td>0.29</td>
<td>0.27</td>
<td>0.66</td>
<td>0.63</td>
</tr>
<tr>
<td>3.0</td>
<td>0.16</td>
<td>0.14</td>
<td>0.51</td>
<td>0.49</td>
</tr>
<tr>
<td>3.5</td>
<td>0.07</td>
<td>0.06</td>
<td>0.35</td>
<td>0.33</td>
</tr>
<tr>
<td>4.0</td>
<td>0.03</td>
<td>0.02</td>
<td>0.21</td>
<td>0.19</td>
</tr>
<tr>
<td>4.5</td>
<td>0.01</td>
<td>0.01</td>
<td>0.11</td>
<td>0.10</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.05</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Figure 3.14: DLS-SVDD simulation results for $p = 10$ with multivariate skew-normal data.

There is a clear similarity in performance between DLS-SVDD and DSVDD, a pattern that can also be seen between LS-SVDD and SVDD. Additionally, DLS-SVDD and DSVDD demonstrate superior performance compared to LS-SVDD and SVDD.
High-dimensional setting: \( p = 100 \) and \( N = 40 \)

In this section we present simulation results for a distribution shift where \( p = 100 \) and \( N = 40 \). For the high-dimensional simulations, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.15 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>DLS-SVDD</th>
<th>DSVDD</th>
<th>LS-SVDD</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.92</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.4</td>
<td>0.91</td>
<td>0.91</td>
<td>0.95</td>
<td>0.94</td>
</tr>
<tr>
<td>0.5</td>
<td>0.91</td>
<td>0.90</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.6</td>
<td>0.90</td>
<td>0.89</td>
<td>0.95</td>
<td>0.93</td>
</tr>
<tr>
<td>0.7</td>
<td>0.88</td>
<td>0.88</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>0.8</td>
<td>0.88</td>
<td>0.87</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>0.9</td>
<td>0.86</td>
<td>0.86</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>1.0</td>
<td>0.85</td>
<td>0.84</td>
<td>0.94</td>
<td>0.93</td>
</tr>
<tr>
<td>1.5</td>
<td>0.77</td>
<td>0.76</td>
<td>0.93</td>
<td>0.92</td>
</tr>
<tr>
<td>2.0</td>
<td>0.66</td>
<td>0.65</td>
<td>0.91</td>
<td>0.90</td>
</tr>
<tr>
<td>2.5</td>
<td>0.53</td>
<td>0.54</td>
<td>0.88</td>
<td>0.87</td>
</tr>
<tr>
<td>3.0</td>
<td>0.41</td>
<td>0.42</td>
<td>0.84</td>
<td>0.82</td>
</tr>
<tr>
<td>3.5</td>
<td>0.30</td>
<td>0.31</td>
<td>0.78</td>
<td>0.77</td>
</tr>
<tr>
<td>4.0</td>
<td>0.20</td>
<td>0.21</td>
<td>0.70</td>
<td>0.69</td>
</tr>
<tr>
<td>4.5</td>
<td>0.11</td>
<td>0.13</td>
<td>0.61</td>
<td>0.60</td>
</tr>
<tr>
<td>5.0</td>
<td>0.06</td>
<td>0.07</td>
<td>0.49</td>
<td>0.50</td>
</tr>
</tbody>
</table>

Figure 3.15: DLS-SVDD simulation results for \( p = 100 \) with multivariate skew-normal data.

The LS-SVDD and SVDD are not as adept at detecting larger shifts compared to the capabilities of DLS-SVDD and DSVDD. For LS-SVDD and SVDD, the Type II error rate does not fall below 50% until \( \delta = 5.0 \) or higher, whereas this is achieved at \( \delta = 3.0 \) for DLS-SVDD and DSVDD.
In this subsection, we evaluate the performance of DLS-SVDD with different activation functions. Note that Ruff et al. [48] suggested that the activation functions within the neural network should be unbounded functions to prevent saturated inputs with all common signs, which would behave like a bias term and collapse as discussed in Proposition 2 of their paper. This means that according to Ruff et al. [48] an unbounded term (or bounded only by zero), such as ReLU, should be used as an activation function within the neural network.

Consequently, we run some simulations using several activation functions within the neural network to see if our proposal will collapse with activation functions other than ReLU. We generate from a multivariate normal distribution with the mean vector set to zero $\mu_0 = 0$ and identity covariance matrix $\Sigma_0 = I_p$. We consider dimensions $p = 5$ and $p = 10$ where $N = 100$ is the number of observations originating from the target class. We compare the performances of DLS-SVDD based on the probability of Type II error using ReLU against logistic sigmoid and hyperbolic tangent activation functions.
In this section we present simulation results for a distribution shift where \( p = 5 \) and \( N = 100 \) for DLS-SVDD using ReLU, logistic sigmoid, and hyperbolic tangent activation functions. The Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.16 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>ReLU</th>
<th>Sigmoid</th>
<th>Tanh</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.92</td>
<td>0.93</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.89</td>
<td>0.92</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.88</td>
<td>0.90</td>
</tr>
<tr>
<td>0.6</td>
<td>0.85</td>
<td>0.86</td>
<td>0.89</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.84</td>
<td>0.87</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.81</td>
<td>0.85</td>
</tr>
<tr>
<td>0.9</td>
<td>0.77</td>
<td>0.79</td>
<td>0.82</td>
</tr>
<tr>
<td>1.0</td>
<td>0.74</td>
<td>0.76</td>
<td>0.80</td>
</tr>
<tr>
<td>1.5</td>
<td>0.57</td>
<td>0.59</td>
<td>0.64</td>
</tr>
<tr>
<td>2.0</td>
<td>0.36</td>
<td>0.40</td>
<td>0.45</td>
</tr>
<tr>
<td>2.5</td>
<td>0.20</td>
<td>0.23</td>
<td>0.28</td>
</tr>
<tr>
<td>3.0</td>
<td>0.09</td>
<td>0.11</td>
<td>0.13</td>
</tr>
<tr>
<td>3.5</td>
<td>0.03</td>
<td>0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>4.0</td>
<td>0.01</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Figure 3.16: DLS-SVDD simulation results for \( p = 5 \) with alternative activation functions.

The ReLU achieves marginally better performance than that of Sigmoid. Both of these activation functions, however, exceed Tanh in terms of performance.
Performance for $p = 10$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 10$ and $N = 100$ for DLS-SVDD using ReLU, logistic sigmoid, and hyperbolic tangent activation functions. The Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.17 shown below.

![Type II Error Rate Table](image)

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>ReLU</th>
<th>Sigmoid</th>
<th>Tanh</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.90</td>
<td>0.91</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.89</td>
<td>0.90</td>
</tr>
<tr>
<td>0.6</td>
<td>0.86</td>
<td>0.87</td>
<td>0.88</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.86</td>
<td>0.87</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.84</td>
<td>0.85</td>
</tr>
<tr>
<td>0.9</td>
<td>0.78</td>
<td>0.82</td>
<td>0.83</td>
</tr>
<tr>
<td>1.0</td>
<td>0.75</td>
<td>0.80</td>
<td>0.81</td>
</tr>
<tr>
<td>1.5</td>
<td>0.58</td>
<td>0.67</td>
<td>0.69</td>
</tr>
<tr>
<td>2.0</td>
<td>0.39</td>
<td>0.51</td>
<td>0.55</td>
</tr>
<tr>
<td>2.5</td>
<td>0.23</td>
<td>0.36</td>
<td>0.42</td>
</tr>
<tr>
<td>3.0</td>
<td>0.11</td>
<td>0.23</td>
<td>0.28</td>
</tr>
<tr>
<td>3.5</td>
<td>0.05</td>
<td>0.14</td>
<td>0.19</td>
</tr>
<tr>
<td>4.0</td>
<td>0.02</td>
<td>0.07</td>
<td>0.12</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.03</td>
<td>0.07</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.01</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Figure 3.17: DLS-SVDD simulation results for $p = 10$ with alternative activation functions.

In terms of performance, ReLU is clearly superior to both Sigmoid and Tanh, with Sigmoid also outperforming Tanh.
Robustness

In real-world applications, the assumption that training data is comprised exclusively of instances originating from a single class of data may not hold true. Consequently, the DLS-SVDD model might be trained with contaminated data that contains both target class instances as well as outliers. To evaluate the robustness of DLS-SVDD, we simulate training datasets with different degrees of contamination. Define the degree of contamination in the training set by $\epsilon$, where $\epsilon$ is a value between 0 and 1. For instance, at a contamination level of $\epsilon = 0.1$, the composition of the training set includes 90% target class instances and 10% outliers. We compare the scenario with no contamination against three varying levels of contamination, specifically $\epsilon = 0.1$, 0.25, and 0.4.

We conduct a series of simulations across varying levels of contamination to investigate the impact of outliers on the performance of the DLS-SVDD model. We consider contamination configurations with target-to-outlier ratios of 90:10, 75:25, and 60:40, thereby generating training sets tailored to each distinct level of contamination. The proportion of the training dataset representing the target class is simulated from a multivariate normal distribution with the mean vector set to zero $\mu_0 = 0$ and identity covariance matrix $\Sigma_0 = I_p$. The outlier data is simulated from the same distribution, with the exception of the mean vector now having a value of 2 in its leading component and all subsequent components at zero, and the identity covariance matrix. After the model was trained on the $\epsilon-$contaminated training set, the evaluation of anomalous observations was considered by changing the zero mean vector $\mu_0$ from the target set’s underlying distribution to a vector having $\delta$ in its first component while the other vector components remained unchanged.

We investigate both low-dimensional and high-dimensional scenarios. For the low-dimensional settings, we consider dimensions $p = 5$ and $p = 10$, where $N = 100$ is the number of observations in the training set. For the high dimensional setting, we consider $p = 100$ dimensions where $N = 40$ is the number of observations in the training set.
Low-dimensional setting: $p = 5$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 5$ and $N = 100$ using contaminated training data. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.18 shown below.

Figure 3.18: DLS-SVDD simulation results for $p = 5$ with contaminated data.

At a contamination level of zero ($\epsilon = 0.0$), the Type II error rate falls below 50% at $\delta = 2.0$. For mild contamination levels of $\epsilon = 0.10$ and $\epsilon = 0.25$, this benchmark is achieved at $\delta = 3.0$. However, for contamination level $\epsilon = 0.40$, the Type II error only drops below this rate at $\delta = 4.0$. 

81
Low-dimensional setting: \( p = 10 \) and \( N = 100 \)

In this section we present simulation results for a distribution shift where \( p = 10 \) and \( N = 100 \) using contaminated training data. For this scenario, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \).

A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.19 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \epsilon = 0.0 )</th>
<th>( \epsilon = 0.1 )</th>
<th>( \epsilon = 0.25 )</th>
<th>( \epsilon = 0.4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.93</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.86</td>
<td>0.93</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.92</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.90</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.9</td>
<td>0.78</td>
<td>0.90</td>
<td>0.91</td>
<td>0.95</td>
</tr>
<tr>
<td>1.0</td>
<td>0.75</td>
<td>0.89</td>
<td>0.91</td>
<td>0.95</td>
</tr>
<tr>
<td>1.5</td>
<td>0.58</td>
<td>0.82</td>
<td>0.86</td>
<td>0.92</td>
</tr>
<tr>
<td>2.0</td>
<td>0.39</td>
<td>0.71</td>
<td>0.79</td>
<td>0.88</td>
</tr>
<tr>
<td>2.5</td>
<td>0.23</td>
<td>0.59</td>
<td>0.71</td>
<td>0.82</td>
</tr>
<tr>
<td>3.0</td>
<td>0.11</td>
<td>0.44</td>
<td>0.60</td>
<td>0.72</td>
</tr>
<tr>
<td>3.5</td>
<td>0.05</td>
<td>0.30</td>
<td>0.48</td>
<td>0.61</td>
</tr>
<tr>
<td>4.0</td>
<td>0.02</td>
<td>0.19</td>
<td>0.36</td>
<td>0.48</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.11</td>
<td>0.26</td>
<td>0.35</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.05</td>
<td>0.16</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Figure 3.19: DLS-SVDD simulation results for \( p = 10 \) with contaminated data.

When there is no contamination (\( \epsilon = 0.0 \)), the Type II error rate drops below 50% at \( \delta = 2.0 \).

When \( \epsilon \) increases to 0.10, this benchmark is met at \( \delta = 3.0 \), and at \( \epsilon = 0.25 \) it occurs at \( \delta = 3.5 \).

However, for \( \epsilon = 0.40 \), this rate is not achieved until \( \delta = 4.0 \).
High-dimensional setting: \( p = 100 \) and \( N = 40 \)

In this section we present simulation results for a distribution shift where \( p = 100 \) and \( N = 40 \) using contaminated training data. For this scenario, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 3.20 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>( \epsilon = 0.0 )</th>
<th>( \epsilon = 0.1 )</th>
<th>( \epsilon = 0.25 )</th>
<th>( \epsilon = 0.4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.4</td>
<td>0.91</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.5</td>
<td>0.90</td>
<td>0.92</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.6</td>
<td>0.88</td>
<td>0.92</td>
<td>0.93</td>
<td>0.92</td>
</tr>
<tr>
<td>0.7</td>
<td>0.87</td>
<td>0.92</td>
<td>0.93</td>
<td>0.92</td>
</tr>
<tr>
<td>0.8</td>
<td>0.87</td>
<td>0.91</td>
<td>0.93</td>
<td>0.92</td>
</tr>
<tr>
<td>0.9</td>
<td>0.84</td>
<td>0.91</td>
<td>0.93</td>
<td>0.92</td>
</tr>
<tr>
<td>1.0</td>
<td>0.75</td>
<td>0.90</td>
<td>0.92</td>
<td>0.90</td>
</tr>
<tr>
<td>1.5</td>
<td>0.65</td>
<td>0.88</td>
<td>0.91</td>
<td>0.89</td>
</tr>
<tr>
<td>2.0</td>
<td>0.52</td>
<td>0.85</td>
<td>0.91</td>
<td>0.87</td>
</tr>
<tr>
<td>2.5</td>
<td>0.40</td>
<td>0.83</td>
<td>0.90</td>
<td>0.84</td>
</tr>
<tr>
<td>3.0</td>
<td>0.28</td>
<td>0.80</td>
<td>0.89</td>
<td>0.82</td>
</tr>
<tr>
<td>3.5</td>
<td>0.18</td>
<td>0.77</td>
<td>0.88</td>
<td>0.79</td>
</tr>
<tr>
<td>4.0</td>
<td>0.11</td>
<td>0.73</td>
<td>0.87</td>
<td>0.77</td>
</tr>
<tr>
<td>4.5</td>
<td>0.06</td>
<td>0.68</td>
<td>0.85</td>
<td>0.73</td>
</tr>
</tbody>
</table>

Figure 3.20: DLS-SVDD simulation results for \( p = 100 \) with contaminated data.

Without contamination, the Type II error rate drops below 50% at \( \delta = 3.0 \). However, any degree of contamination in high-dimensional settings significantly degrades DLS-SVDD’s performance such that none of the contaminated models reach a Type II error rate under 50% by \( \delta = 5.0 \).
Summary of Simulation Results

In this section, we evaluated the performance of DLS-SVDD against competing methods across a variety of scenarios, spanning both low-dimensional and high-dimensional settings with normal and non-normal distributions. In the subsequent discussion, we present a comprehensive summary of the results obtained from the simulation study.

First, we considered simulations with uncorrelated multivariate normal data. The results indicated that the performance disparities observed between the kernel-based methods LS-SVDD and SVDD were similarly reflected between the deep learning methods DLS-SVDD and DSVDD. For instance, in the cases of dimensions $p = 5$ and $p = 100$, respectively, the performance of DLS-SVDD and DSVDD was almost indistinguishable, a pattern that was also seen with LS-SVDD and SVDD. Furthermore, in the $p = 10$ scenario, DLS-SVDD demonstrated marginally better performance compared to DSVDD, and similarly, LS-SVDD narrowly outperformed SVDD. In other words, the performance patterns between the kernel-based methods, whether they marginally outperformed or exhibited negligible differences, were consistently mirrored in the deep learning models.

The second set of simulations involved correlated multivariate normal data. Unlike in the previous set of simulations, the performance differences between the kernel-based methods were not consistently mirrored in the deep learning methods. For example, in the $p = 5$ case, LS-SVDD had better performance than SVDD, whereas DLS-SVDD and DSVDD had comparable performance results. However, in the $p = 10$ scenario, we observe SVDD outperforming LS-SVDD, and similarly, DSVDD exceeding the performance of DLS-SVDD. Therefore, the results for the uncorrelated multivariate normal data do not indicate that the discrepancies in performance between the kernel-based methods necessarily transfer over to the deep learning methods. However, there were instances where this was observed to occur.
The next set of simulations we explored focused on multivariate skew-normal data. Negligible performance differences between DLS-SVDD and DSVDD were observed, and this pattern held for LS-SVDD and SVDD as well. Additionally, the kernel-based methods struggled to achieve good performance in the high-dimensional setting, whereas the deep learning methods showed much better capabilities.

The fourth set of simulations we considered involved the use of different activation functions in the hidden layers of the network. In both dimension settings considered, ReLU achieved the best overall performance. In low-dimensional settings, using the Sigmoid activation function may be considered as an alternative option to ReLU given their comparable performance outcomes. However, using Tanh would not be recommended given it is the lowest performing activation function out of the three considered.

In the final set of simulations, we tested the robustness of the DLS-SVDD model. In low-dimensional settings, we observed a degradation in performance that appears relatively consistent for contamination levels of $\epsilon = 0.10$ and $\epsilon = 0.25$. However, there is a significant drop in performance once the contamination level reaches $\epsilon = 0.40$. This trend of deteriorating model performance as the contamination level $\epsilon$ increases is also evident in the high-dimensional setting, however the model struggles to achieve good performance with any level of contamination in high dimensions. Therefore, it is not recommended to use DLS-SVDD when working with high-dimensional data that is contaminated if the outliers cannot be removed from the training set prior to model training.
We illustrate the real-world application of DLS-SVDD and compare its performance against LS-SVDD and SVDD using the ‘breast cancer dataset.’ The dataset was created by Dr. William H. Wolberg [58], a physician at the University of Wisconsin. The breast cancer dataset is publicly available and can be found in UCI Machine Learning Repository. There are two target classes in the dataset, labeled as 2 or 4. The dataset consists of nine numerical features measured on 683 patients: 444 diagnosed with benign cancer (target class 2) and 239 diagnosed with malignant cancer (target class 4).

The goal is to train a model with data exclusively from patients with benign cancer, and predict whether unseen patients have either benign or malignant cancer. We use the first 80 observations of the target class 2 data (benign cancer group) as our training set. This yields \( p = 9 \) variables with \( N = 80 \) observations. The distribution of this data is unknown and demonstrates the application of the proposed method on data that is not multivariate normal.

The test set is formed by using the last five observations of the target class 2 data (benign cancer group) and the first eight observations of the target class 4 data (malignant cancer group). This yields a total of 13 observations in the test set.

We applied DLS-SVDD, LS-SVDD, and SVDD to the 13 observations in the test set and compare their performance in detecting outliers (malignant cancer group). For DLS-SVDD, we use the same neural network architecture outlined in the Performance Settings Section. For LS-SVDD, we compute the \( \alpha \)'s using Equation (2.53). In the case of SVDD, the Lagrange multipliers are obtained using a quadratic programming solver. The dual problem is optimized in this manner due to the nonexistence of a closed-form solution, a distinction from LS-SVDD. The results for LS-SVDD and SVDD are shown in Figures 3.21 and 3.22, respectively. Calculating the distance statistic \( d \)
using Equation (3.50) for DLS-SVDD, the results are shown in Figure 3.23.

While all of the models were able to correctly classify the targets and outliers respectively, there are some notable differences. The first five observations, which belong to the target class 2 data (benign cancer group), are correctly classified by LS-SVDD. However, there is some variation between these observations with respect to their distance to the boundary. Specifically, the fourth observation is much closer to the boundary than the other observations. Furthermore, while all of the outliers are properly classified, they have very close proximity to the boundary.

![Figure 3.21: OCC using LS-SVDD for the breast cancer dataset example.](image-url)
In general, we observe a similar pattern for SVDD. The observations from the target class exhibit a structure akin to LS-SVDD, except with a global reduction in proximity to the boundary. Moreover, all of the outlier observations are positioned closer to the boundary, and the variability in distances between the outlier observations and the boundary is reduced. As a result, the boundary is located in closer proximity to the outlier observations compared to LS-SVDD.

Figure 3.22: OCC using SVDD for the breast cancer dataset example.
Alternatively, DLS-SVDD has very little variation in the first five target observations with respect to their distance to the boundary. The outliers are also farther away from the boundary relative to the target class compared to LS-SVDD and SVDD. This demonstrates the effectiveness of the proposed one-class classifier in a real-world application when dealing with data from an unknown distribution.

Figure 3.23: OCC using DLS-SVDD for the breast cancer dataset example.
CHAPTER 4: DEEP LEAST SQUARES ONE-CLASS SUPPORT VECTOR MACHINE

Introduction

In recent years, there has been increasing interest in addressing the challenges associated with severe class imbalance in the classification task. In various practical scenarios, the only available data for model training is related to normal operating conditions without outliers. This data can be considered as originating from the primary class, referred to as the target class, which constitutes the only available information during model training. This problem setting has underscored the motivation behind a reevaluation of prevailing classification paradigms, and one-class classification has gained prominence as a strategic solution to this common issue.

One-class classification has emerged as a critical approach to this pervasive problem by focusing exclusively on learning the characteristics of the training set, whose observations originate from the target class, to facilitate the development of a specialized one-class classifier for outlier detection. In this setting, previously unseen observations originating from one of potentially many alternative classes are effectively treated as outliers. These outlier observations are characteristically different than the target class, and the distinction in the underlying behavior of the data can be measured in various ways. This approach holds significant importance in real-world settings marked by the challenges of outlier characterization, which may be due to various factors such as financial constraints or practical limitations.
Deep Least Squares One-Class Support Vector Machine (DLS-OCSVM) serves as a new approach for one-class classification, combining the capability of deep neural networks to derive informative feature representations from the data with a quadratic one-class objective. DLS-OCSVM integrates Least Squares Support Vector Machine (LS-OCSVM) into the neural network architecture, and jointly trains a neural network while finding a separating hyperplane which has the maximum distance from the origin and minimized sum of squared errors.

Consider a training dataset $X = \{x_1, x_2, ..., x_N\} \subseteq \mathcal{X}$, where $N$ is the number of vectors and $\mathcal{X} \subseteq \mathbb{R}^p$ is an original input space. For this input space, we will define a corresponding output space $\mathcal{F} \subseteq \mathbb{R}^d$. Let $\phi(\cdot; \mathcal{V}, \mathcal{B})$ be a neural network that maps $\mathcal{X} \mapsto \mathcal{F}$ with $L \in \mathbb{N}$ layers. The network consists of $L-1$ hidden layers and a final output layer, and the architecture is characterized by a set of weight matrices $\mathcal{V} = \{V^{[1]}, V^{[2]}, ..., V^{[L]}\}$ and bias vectors $\mathcal{B} = \{b^{[1]}, b^{[2]}, ..., b^{[L]}\}$, corresponding to the weights $V^{[\ell]}$ and biases $b^{[\ell]}$ of layer $\ell \in \{1, 2, ..., L\}$, respectively. In other words, $\phi(x; \mathcal{V}, \mathcal{B})$ denotes the feature representation of $x \in \mathcal{X}$ given by network $\phi$ with weights $\mathcal{V}$ and biases $\mathcal{B}$.

The goal is to construct a classification boundary and separate the target class from the origin using a hyperplane by maximizing the distance between the hyperplane and the origin while simultaneously minimizing the sum of squares errors. This classification boundary is a hyperplane characterized by a normal vector $w \in \mathcal{F}$ and an offset $\rho \in \mathbb{R}$. The mathematical formulation of this model for jointly learning the network parameters $\mathcal{V}$ while finding the maximum hyperplane
is given by the following optimization problem:

$$\min_{\rho, \mathbf{w}, \mathbf{\xi}, \mathbf{V}} \frac{1}{2} \mathbf{w}^T \mathbf{w} - \rho + \frac{1}{2\nu N} \sum_{i=1}^{N} \xi_i^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||\mathbf{V}^{[\ell]}||_F^2$$

subject to \( \mathbf{w}^T \phi(\mathbf{x}_i; \mathbf{V}, \mathbf{B}) = \rho - \xi_i, \quad i = 1, 2, ..., N \) \hspace{1cm} (4.1)

where \( \xi_i \) are the error variables resulting from a training vector \( \mathbf{x}_i \) as it relates to the hyperplane, and the parameter \( \nu > 0 \) is introduced to control the influence of the error variables, consequently regulating the error associated with the target class and the decision boundary. The last term in the objective is a standard weight decay regularizer on the network weights \( \mathbf{V} \) with tuning parameter \( \lambda > 0 \), where \( || \cdot ||_F \) denotes the Frobenius norm.

The DLS-OCSVM constrained optimization problem (4.1) can be equivalently expressed as the following unconstrained optimization problem:

$$\min_{\rho, \mathbf{w}, \mathbf{V}, \mathbf{B}} \frac{1}{2} \mathbf{w}^T \mathbf{w} - \rho + \frac{1}{2\nu N} \sum_{i=1}^{N} (\rho - \mathbf{w}^T \phi(\mathbf{x}_i; \mathbf{V}, \mathbf{B}))^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||\mathbf{V}^{[\ell]}||_F^2$$ \hspace{1cm} (4.2)

Note that in this optimization problem, the \( \ell_2 \) norm is employed—a smooth and differentiable alternative to the hinge loss used in the OC-NN optimization problem (2.81). As a result, the solution to the optimization problem (4.2) can be readily obtained. We can define the loss function, denoted by \( \mathcal{L} \), as follows:

$$\mathcal{L}(\rho, \mathbf{w}, \mathbf{V}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \rho + \frac{1}{2\nu N} \sum_{i=1}^{N} (\rho - \mathbf{w}^T \phi(\mathbf{x}_i; \mathbf{V}, \mathbf{B}))^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||\mathbf{V}^{[\ell]}||_F^2$$ \hspace{1cm} (4.3)
To optimize this problem, we use stochastic gradient descent (SGD). Define the DLS-OCSVM loss function for SGD, denoted by $L_s$, at a random point $x_*$ as follows:

$$L_s (\rho, w, \mathcal{V}) = \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, \mathcal{V}, \mathcal{B}) \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||\mathcal{V}[\ell]||_F^2$$ \hspace{1cm} (4.4)

**Theorem 4.0.1** $\rho$ is obtained using the following closed-form solution:

$$\rho = \nu + w^T \phi(x_*, \mathcal{V}, \mathcal{B})$$ \hspace{1cm} (4.5)

Note that Equation (4.5) defines the hyperplane offset based solely on the hyperplane normal vector and neural network weights. This closed-form solution is one of the differentiating factors of DLS-OCSVM compared to other OCC boundary techniques. During training, the offset can be updated using the analytic solution every epoch, or on a predefined schedule, providing additional model flexibility. This approach differs from OC-NN, which finds the offset by solving for the quantile value of the scores, as well as LS-OCSVM, which has a non-sparse solution and uses all of the observations in the training set to solve for the offset.

The separating hyperplane of DLS-OCSVM is defined as follows:

$$f(x) = w^T \phi(x; \mathcal{V}, \mathcal{B}) - \rho = 0$$ \hspace{1cm} (4.6)

In this context, the Euclidean distance of an arbitrary point $x$ to the hyperplane can be computed as:

$$dist(x) = \frac{|f(x)|}{||w||_2}$$ \hspace{1cm} (4.7)
Note that DLS-OCSVM, similar to LS-OCSVM, does not have a natural decision function due to the quadratic objective. Hence, a discriminant function that distinguishes between targets and outliers will require some thresholding approach. Xing and Li [60] proposed to use the maximum distance between the training samples and the hyperplane as a threshold. However, we may be interested in placing progressively larger weights on objects farther from the hyperplane separating the target class from outliers. An alternative thresholding approach is to use the squared Euclidean distance as follows:

$$\theta = \max_{i=1,2,...,N} \text{dist}(x_i)^2$$  \hspace{1cm} (4.8)

A given test point $z \in \mathcal{X}$ is in the target class if the squared Euclidean distance from a point $z$ to the hyperplane is less than a threshold $\theta$. In other words, the test point $z$ will be in the target class if the following condition is satisfied:

$$s(z) = \frac{(w^*^T \phi(z; \mathcal{V}^*, B^*) - \rho^*)^2}{||w^*||_2^2} \leq \theta$$  \hspace{1cm} (4.9)

where $w^*$, $\rho^*$, $\mathcal{V}^*$, and $B^*$ are the normal vector, offset, network weights, and network biases of a trained model, respectively. Furthermore, we denote the anomaly score for a given test point $z$ by $s(z)$.

The SGD DLS-OCSVM training procedure is summarized in Algorithm 7.

In the following section, we systematically derive the mathematical equations for DLS-OCSVM using a simple artificial neural network with one hidden layer. This is followed by an analogous derivation adapted to a more complex model with two hidden layers. We conclude by generalizing these derivations to apply to arbitrarily deep neural networks with $L \in \mathbb{N}$ hidden layers.
Algorithm 7 DLS-OCSVM SGD Algorithm

**Input:** Training dataset $X = \{x_1, x_2, \ldots, x_N\} \subseteq \mathcal{X}$ with $x_i \in \mathbb{R}^p$, hyperparameter $\lambda$.

**Output:** A normal vector $w^*$, offset $\rho^*$, network weights $V^*$, and network biases $B^*$ of a trained model.

1: for each epoch do
2: Randomly sample a training observation $x \in \mathcal{X}$.
3: Calculate $\phi(x; V, B)$ by forward propagation through DLS-OCSVM model.
4: Calculate $L_s$ using Equation (4.4).
5: Update the network weights, biases, and normal vector using the Adam algorithm.
6: Solve for the hyperplane offset using Equation (4.5).
7: end for
8: return $w^*, \rho^*, V^*,$ and $B^*$

---

**Neural Network with One Hidden Layer**

For the mathematical derivations, we design a simple feed forward neural network with one hidden layer. This network has rectified linear unit (ReLU) activation functions in the hidden layer, denoted by $g_1(\cdot)$, and linear activation functions in the output layer, denoted by $g_2(\cdot)$.

Let $Z^{[1]}$ denote the weighted input of the first hidden layer. The output of the first hidden layer, denoted by $A^{[1]}$, is obtained after applying $g_1(\cdot)$ in an element-wise manner to the weighted input $Z^{[1]}$. For the output layer, the weighted input is denoted by $Z^{[2]}$, and the network output, defined as $\phi(\cdot; V, B) = A^{[2]}$, is obtained after $g_2(\cdot)$ is applied element-wise to $Z^{[2]}$.

During the forward propagation phase, an observation $x_*$ is initially processed at the input layer, then passed through the hidden layer, and finally is directed to the output layer. The mathematical functions detailing this progression are shown below.

$$Z^{[1]} = V^{[1]}x_* + b^{[1]} \quad (4.10)$$

$$A^{[1]} = g_1(Z^{[1]}) \quad (4.11)$$

$$Z^{[2]} = V^{[2]}A^{[1]} + b^{[2]} \quad (4.12)$$

$$\phi(x_*; V, B) = A^{[2]} = g_2(Z^{[2]}) \quad (4.13)$$
Through the application of the chain rule from differential calculus, backpropagation methodically propagates error signals backward through the network from the output layer to the input layer, thereby facilitating weight adjustments across the entire network. By initially defining the errors that are propagated backwards through each layer, a concise representation of the gradients for the neural network’s weights and biases can be obtained. The proofs for Proposition 4.0.1 and Proposition 4.0.2, respectively, can be found in Appendix D.

Define $\delta^{[2]}$ as the error propagated backwards through the output layer’s activation function $g_2$ as follows:

$$
\delta^{[2]} = \frac{1}{\nu} \left( w^T \phi(x^*; \mathcal{V}, \mathcal{B}) - \rho \right) w^T \circ g'_2(Z^{[2]})
$$

(4.14)

where $\circ$ denotes the Hadamard product.

Next, define $\delta^{[1]}$ as the error propagated backwards through the hidden layer’s activation function $g_1$ as follows:

$$
\delta^{[1]} = V^{[2]^T} \delta^{[2]} \circ g'_1(Z^{[1]})
$$

(4.15)

The gradients for the hyperplane normal vector and neural network weight matrices, respectively, are presented in Proposition 4.0.1.

**Proposition 4.0.1**

$$
\frac{\partial L_s}{\partial w} = \frac{1}{\nu} \left( w^T \phi(x^*; \mathcal{V}, \mathcal{B}) - \rho \right) \phi(x^*; \mathcal{V}, \mathcal{B}) + w
$$

(4.16)

$$
\frac{\partial L_s}{\partial V^{[2]}} = \delta^{[2]} A^{[1]^T} + \lambda V^{[2]}
$$

(4.17)

$$
\frac{\partial L_s}{\partial V^{[1]}} = \delta^{[1]} x^*_s + \lambda V^{[1]}
$$

(4.18)
Furthermore, the gradients for the two bias vectors are presented in Proposition 4.0.2.

**Proposition 4.0.2**

\[
\frac{\partial L_s}{\partial b[2]} = \delta^{[1]} \\
\frac{\partial L_s}{\partial b[1]} = \delta^{[2]}
\]

The SGD uses the following iterative updates for neural network layers \( \ell = \{1, 2\} \):

\[
V_{(t+1)}^{[\ell]} = V_{(t)}^{[\ell]} - \eta_1 \frac{\partial L_s}{\partial V_{(t)}^{[\ell]}} \\
b_{(t+1)}^{[\ell]} = b_{(t)}^{[\ell]} - \eta_2 \frac{\partial L_s}{\partial b_{(t)}^{[\ell]}} \\
w_{(t+1)} = w_{(t)} - \eta_3 \frac{\partial L_s}{\partial w_{(t)}} \\
\rho_{(t+1)} = \nu + w_{(t+1)}^T \phi \left( x_*; V_{(t+1)}, B_{(t+1)} \right)
\]

where \( \eta_1, \eta_2, \) and \( \eta_3 \) are defined as the learning rates for the network weights, network biases, and normal vector, respectively.

For a neural network with a single hidden layer, the SGD iterative update formulas states that the optimization problem (4.4) can be easily solved using SGD. First, the weights and bias terms are updated. Next, the normal vector is updated. Then the offset is updated using the results from Theorem 4.0.1.
In this section, we consider the mathematical derivations for an artificial neural network with two hidden layers. This network has rectified linear unit (ReLU) activation functions in the hidden layers, denoted by $g_i(\cdot)$, $i = 1, 2$, and linear activation functions in the output layer, denoted by $g_3(\cdot)$.

Let $Z^{[1]}$ denote the weighted input of the first hidden layer. The output of the first hidden layer, denoted by $A^{[1]}$, is obtained after applying $g_1(\cdot)$ in an element-wise manner to the weighted input $Z^{[1]}$. Similarly, $Z^{[2]}$ represents the weighted input of the second hidden layer. The output of the second hidden layer, denoted by $A^{[2]}$, is obtained after applying $g_2(\cdot)$ element-wise to the weighted input $Z^{[2]}$. For the output layer, the weighted input is denoted by $Z^{[3]}$, and the network output, defined as $\phi(\cdot; V, B) = A^{[3]}$, is obtained after $g_3(\cdot)$ is applied element-wise to $Z^{[3]}$.

During the forward propagation phase, an observation $x_*$ is initially processed at the input layer, then sequentially passed through each of the hidden layers, and finally is directed to the output layer. The mathematical functions detailing this progression are shown below.

\begin{align*}
Z^{[1]} &= V^{[1]}x_* + b^{[1]} \\
A^{[1]} &= g_1(Z^{[1]}) \\
Z^{[2]} &= V^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g_2(Z^{[2]}) \\
Z^{[3]} &= V^{[3]}A^{[2]} + b^{[3]} \\
\phi(x_*; V, B) &= A^{[3]} = g_3(Z^{[3]})
\end{align*}
Through the application of the chain rule from differential calculus, backpropagation methodically propagates error signals backward through the network from the output layer to the input layer, thereby facilitating weight adjustments across the entire network. By initially defining the errors that are propagated backwards through each layer, a concise representation of the gradients for the neural network’s weights and biases can be obtained. The proofs for Propositions 4.0.3 and 4.0.4, respectively, can be found in Appendix D.

Define $\delta^{[3]}$ as the error propagated backwards through the output layer’s activation function $g_3$ as follows:

$$\delta^{[3]} = \frac{1}{\nu} \left( w^T \phi(x^*; V, B) - \rho \right) w^T \circ g_3'(Z^{[3]})$$ (4.27)

where $\circ$ denotes the Hadamard product.

Next, define $\delta^{[2]}$ as the error propagated backwards through the second hidden layer’s activation function $g_2$ as follows:

$$\delta^{[2]} = V^{[3]} T \delta^{[3]} \circ g_2'(Z^{[2]})$$ (4.28)

Similarly, define $\delta^{[1]}$ as the error propagated backwards through the first hidden layer’s activation function $g_1$ as follows:

$$\delta^{[1]} = V^{[2]} T \delta^{[2]} \circ g_1'(Z^{[1]})$$ (4.29)
The gradients for the hyperplane normal vector and neural network weight matrices, respectively, are presented in Proposition 4.0.3.

**Proposition 4.0.3**

\[
\frac{\partial L_s}{\partial \mathbf{w}} = \frac{1}{\nu} (\mathbf{w}^T \phi (\mathbf{x}_*; \mathcal{V}, \mathcal{B}) - \rho) \phi (\mathbf{x}_*; \mathcal{V}, \mathcal{B})
\]

\[
\frac{\partial L_s}{\partial \mathbf{V}^{[3]}} = \delta^{[3]} A^{[2]T} + \lambda \mathbf{V}^{[3]}
\]

\[
\frac{\partial L_s}{\partial \mathbf{V}^{[2]}} = \delta^{[2]} A^{[1]T} + \lambda \mathbf{V}^{[2]}
\]

\[
\frac{\partial L_s}{\partial \mathbf{V}^{[1]}} = \delta^{[1]} \mathbf{x}_*^T + \lambda \mathbf{V}^{[1]}
\]

Furthermore, the gradients for the three bias terms are presented in Proposition 4.0.4.

**Proposition 4.0.4**

\[
\frac{\partial L_s}{\partial \mathbf{b}^{[3]}} = \delta^{[3]}
\]

\[
\frac{\partial L_s}{\partial \mathbf{b}^{[2]}} = \delta^{[2]}
\]

\[
\frac{\partial L_s}{\partial \mathbf{b}^{[1]}} = \delta^{[1]}
\]
The SGD uses the following iterative updates for neural network layers $\ell = \{1, 2, 3\}$:

$V_\ell(t+1) = V_\ell(t) - \eta_1 \frac{\partial L_s}{\partial V_\ell(t)}$

$b_\ell(t+1) = b_\ell(t) - \eta_2 \frac{\partial L_s}{\partial b_\ell(t)}$

$w(t+1) = w(t) - \eta_3 \frac{\partial L_s}{\partial w(t)}$

$\rho(t+1) = \nu + w_{(t+1)}^T \phi \left( x_* ; V_{(t+1)} , B_{(t+1)} \right)$

where $\eta_1, \eta_2,$ and $\eta_3$ are defined as the learning rates for the network weights, network biases, and normal vector, respectively.

_Neural Network with L Hidden Layers_

It is straightforward to generalize the model to incorporate deeper network architectures. Consider a neural network with $L$ hidden layers ($\ell = 1, 2, ..., L$) and one output layer ($\ell = L + 1$). The network architecture consists of a set of weight matrices $V = \{V^1, V^2, ..., V^{L+1}\}$ and bias vectors $B = \{b^1, b^2, ..., b^{L+1}\}$, corresponding to the weights $V^\ell$ and biases $b^\ell$ of layer $\ell \in \{1, 2, ..., L + 1\}$, respectively.

The generalized forward propagation equations for neural network layers $\ell = 1, 2, ..., L + 1$ can be expressed as follows:

$Z^\ell = V^\ell A^{\ell-1} + b^\ell$  \hspace{1cm} (4.37)

$A^\ell = g_\ell \left( Z^\ell \right)$  \hspace{1cm} (4.38)

where the network input is defined as $A^0 = x_*$. 

101
For output layer $\ell = L + 1$, the error propagated backwards through the output layer’s activation function $g_{L+1}$ can be expressed as follows:

$$\delta^{[L+1]} = \frac{1}{\nu} \left( w^T \phi(x^*_\ell; \mathcal{V}, \mathcal{B}) - \rho \right) w^T \circ g'_{L+1} \left( Z^{[L+1]} \right)$$  \hspace{1cm} (4.39)$$

where $\circ$ denotes the Hadamard product.

Then, for hidden layers $\ell = 1, 2, ..., L$, the errors propagated backwards through the hidden layers can be expressed as follows:

$$\delta^{[\ell]} = V^{[\ell+1]T} \delta^{[\ell+1]} \circ g'_{\ell} \left( Z^{[\ell]} \right)$$  \hspace{1cm} (4.40)$$

For neural network layers $\ell = 1, 2, ..., L + 1$, the gradients of the weight and bias parameters, respectively, are succinctly represented using the equations shown below.

$$\frac{\partial \mathcal{L}_s}{\partial V^{[\ell]}} = \delta^{[\ell]} A^{[\ell-1]T} + \lambda V^{[\ell]} \hspace{1cm} (4.41)$$

$$\frac{\partial \mathcal{L}_s}{\partial b^{[\ell]}} = \delta^{[\ell]} \hspace{1cm} (4.42)$$
DLS-OCSVM Classification Boundary

In this section, we illustrate how the DLS-OCSVM adapts its decision boundary through an iterative learning process, using a specially adapted “two moons” dataset for practical demonstration. This dataset is especially well-suited for demonstrating how DLS-OCSVM’s decision boundary can adapt to complex, non-linear patterns in the data, offering insights into its superior performance over classical models constrained by simpler geometric boundaries. Using a modified version of the popular “two moons” dataset, we enhance the complexity of the one-class classification task by designating both moons as the target class while introducing a set of outlier observations that are uniformly sampled from a circular region situated between the two moons.

We employ a series of illustrative diagrams to capture the step-by-step evolution of the DLS-OCSVM decision boundary, focusing on the moments of meaningful transformation throughout the training procedure. These visualizations highlight how the decision boundary iteratively adjusts to more accurately encapsulate the data points of the target class while excluding outliers.

The target class consists of 200 observations that are sampled from two interleaving half-circles with an added Gaussian noise that has standard deviation $\sigma = 0.04$. The outlier class is comprised of 50 uniformly sampled observations from a circle with radius $\frac{1}{5}$ that is centered at the mean of the target class. A visual representation of the data in the input space is shown in Figure 4.1.
Figure 4.1: Visualization of the simulated dataset comprised of the target class sampled from the Two Moons dataset and the outlier class uniformly sampled from a circle with radius $\frac{1}{5}$ in the input space.

The model is training exclusively on the data from the target class. In the preliminary stages of DLS-OCSVM model training, the classification boundary for the target class is characterized by an irregular shape with a large central void, resulting in a considerable number of target class observations being erroneously classified as outliers. This early stage of the boundary after 5 epochs is presented in Figure 4.2.
Figure 4.2: Visualization of the DLS-OCSVM decision boundary in the input space during model training after 5 epochs.

With additional model training, the classification boundary is gradually refined into a more spherical configuration. After the 10th epoch, the boundary’s irregular contours start to become more pronounced, and the central void begins to close. By 20 epochs, the shape further aligns with the target class’s dual half-circle structure, and the gap continues to close. Figure 4.3 illustrates the classification boundary after 10 and 20 epochs, respectively.
Figure 4.3: Visualization of the DLS-OCSVM decision boundary in the input space during model training after 10 epochs (left) and 20 epochs (right).

After 40 epochs, there’s a noticeable contraction in the boundary’s interior region, resulting in an improved data description that almost entirely captures the target class. By 75 epochs, this contraction continues, nearly isolating the outlier class completely and resulting in only a few misclassified instances. Figure 4.4 depicts the boundary at these stages.

Figure 4.4: Visualization of the DLS-OCSVM decision boundary in the input space during model training after 40 epochs (left) and 75 epochs (right).
After 200 epochs, the model has converged to a local minimum, resulting in a clear demarcation between the target class and outliers, as shown in Figure 4.5.

Figure 4.5: Visualization of the DLS-OCSVM decision boundary in the input space during model training after 200 epochs.
Simulation Study

Performance Evaluation

The performance of the proposed method is compared against other established methods using an evaluation framework based on the probability of a Type II error. Note that one can use the threshold $\theta$ outlined in the Methodology section to detect outliers. However, in scenarios where controlling the Type I error rate is a priority, it is recommended to use a threshold, $h$, tied to the probability of a Type I error.

Two particular components are of interest within this framework. The first quantifies the spatial distance between a new observation $z$ and the target class through a distance measure. The second is a threshold for discrimination, denoted by $h$, which serves as a point of comparison against the previously mentioned distance metric.

An observation that has not been previously observed will be assigned to the target class if the calculated distance measure is below a predefined threshold $h$. Under this setting, the assessment of misclassifications from the outlier class (Type II error) is not possible. As a result, control is limited to observations from the target class that have been mistakenly classified as outliers (Type I error). To conduct a comprehensive evaluation of the performance of DLS-OCSVM, both the probability of a Type I error, denoted by $\alpha$, and the probability of a Type II error, denoted by $\beta$, can be taken into account.
Consider the availability of a training set, consisting of \( p \)-dimensional vectors for \( i = 1, 2, \ldots, N \) assumed to originate from the target class. For each newly encountered vector \( z \), the distance statistic \( d \) is computed as follows:

\[
d = \frac{\left( w^* \phi(z; V^*, B^*) - \rho^* \right)^2}{||w^*||^2_2}
\]

(4.43)

where \( w^*, \rho^*, V^*, \) and \( B^* \) are the normal vector, offset, network weights, and network biases of a trained model, respectively. The new vector \( z \) is classified as an outlier if the condition \( d > h \) is satisfied, where the threshold \( h \) is chosen to achieve a specified Type I error rate \( \alpha \). Values of \( h \) are generated through the application of the bootstrap simulation technique summarized in Algorithm 8.

**Algorithm 8** DLS-OCSVM Bootstrap Algorithm for Threshold \( h \)

**Input:** Training dataset \( X = \{x_1, x_2, \ldots, x_N\} \subseteq \mathcal{X} \) with \( x_i \in \mathbb{R}^p \), \( B \) bootstrap samples, normal vector \( w^* \), offset \( \rho^* \), network weights \( V^* \), and network biases \( B^* \) of a trained model.

**Output:** A threshold \( h \).

1: \textbf{for} \( x_i \in \mathcal{X} \) \textbf{do}
2: \hspace{1em} \textbf{find} \( d_i = \frac{(w^* \phi(x_i; V^*, B^*) - \rho^*)^2}{||w^*||^2_2} \)
3: \textbf{end for}
4: \textbf{for} \( b \leftarrow 1, B \) \textbf{do}
5: \hspace{1em} (i) Draw a bootstrap sample of size \( N \) from the set of \( N \) \( d \) statistics
6: \hspace{1em} (ii) If \( \alpha \) is the desired Type I error, determine the \( 100 \times (1 - \alpha) \)th percentile value
7: \textbf{end for}
8: \textbf{return} \( h \)
**Performance Settings**

To assess the performance of DLS-OCSVM, we evaluate the probability of a Type II error under several types of scenarios in both low-dimensional settings and high-dimensional settings. First, we consider a data distribution shift in the mean vector using uncorrelated data that is generated from the multivariate normal distribution. Then, we compare the model performance of DLS-OCSVM using several different activation functions in the hidden layers to determine if any difficulties are encountered with activation functions other than ReLU.

The network architecture under consideration for DLS-OCSVM is an artificial neural network with 2 hidden layers that has bias incorporated in both hidden layers and the output layer. The hidden layers and output layer each have 15 neurons, respectively. Leaky ReLU activation functions with leakiness $\alpha = 0.1$ are used in the hidden layers, and linear activation functions are used in the output layer. DLS-OCSVM is trained using stochastic gradient descent (SGD) and Adam optimizer, proposed by Kingma and Ba [28], to optimize the network weights and biases as well as the hyperplane normal vector $w$, which we treat as a learnable parameter that is updated every epoch. The hyperplane offset is also updated every epoch using the closed-form solution from Equation (4.5).

All models are implemented using Python 3.10.12, PyTorch-2.0.1, and CUDA 11.8. The experiments are carried out on a PC with an NVIDIA GeForce RTX 3080 GPU and 128 GB of RAM.
In the first scenario, we evaluate the probability of a Type II error following a data distribution shift using uncorrelated multivariate normal data. We compare the model performance of DLS-OCSVM with respect to its kernel-based analogue, LS-OCSVM, as well as SVDD.

The target set is generated from a multivariate normal distribution with the mean vector set to zero $\mu_0 = 0$ and identity covariance matrix $\Sigma_0 = I_p$. We investigate both low-dimensional and high-dimensional scenarios. For the low-dimensional setting, we consider dimensions $p = 5$, $p = 10$ and $p = 20$, where $N = 100$ is the number of observations originating from the target class. For the high-dimensional setting, we consider $p = 100$ dimensions, where $N = 40$ is the number of observations originating from the target class.

After the model was trained on data originating solely from the target class, the evaluation of anomalous observations was considered by changing the zero mean vector $\mu_0 = 0$ from the target set’s underlying distribution to a vector having $\delta$ in its first component while the other vector components remained unchanged. If the data generating process has a distribution shift from $N(\mu_0, \Sigma_0)$ to $N(\mu_1, \Sigma_0)$, we are interested in the speed at which the algorithm can detect the change. Consequently, the performance following any step change in the mean can be modeled by changing a single component of the mean vector as follows: set the target set’s mean vector to $\mu_0 = 0$ and the covariance matrix to $\Sigma_0 = I_p$. Thereafter, by adding a shift $\delta$ to the first component of each vector we can explore all possible step changes in the mean vector.
Low-dimensional setting: $p = 5$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 5$ and $N = 100$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 4.6 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DLS-OCSVM</th>
<th>LS-OCSVM</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.87</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>0.6</td>
<td>0.85</td>
<td>0.91</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.90</td>
<td>0.93</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.89</td>
<td>0.92</td>
</tr>
<tr>
<td>0.9</td>
<td>0.78</td>
<td>0.87</td>
<td>0.91</td>
</tr>
<tr>
<td>1.0</td>
<td>0.74</td>
<td>0.79</td>
<td>0.84</td>
</tr>
<tr>
<td>1.5</td>
<td>0.58</td>
<td>0.65</td>
<td>0.73</td>
</tr>
<tr>
<td>2.0</td>
<td>0.38</td>
<td>0.49</td>
<td>0.58</td>
</tr>
<tr>
<td>2.5</td>
<td>0.22</td>
<td>0.32</td>
<td>0.41</td>
</tr>
<tr>
<td>3.0</td>
<td>0.20</td>
<td>0.18</td>
<td>0.25</td>
</tr>
<tr>
<td>3.5</td>
<td>0.01</td>
<td>0.08</td>
<td>0.13</td>
</tr>
<tr>
<td>4.0</td>
<td>0.00</td>
<td>0.03</td>
<td>0.05</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.01</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Figure 4.6: DLS-OCSVM simulation results for $p = 5$ with uncorrelated multivariate normal data.

The LS-OCSVM performs better than SVDD, while DLS-OCSVM achieves superior performance compared to both LS-OCSVM and SVDD.
Low-dimensional setting: $p = 10$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 10$ and $N = 100$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 4.7 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DLS-OCSVM</th>
<th>LS-OCSVM</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.87</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.85</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.91</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>0.9</td>
<td>0.77</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>1.0</td>
<td>0.74</td>
<td>0.90</td>
<td>0.93</td>
</tr>
<tr>
<td>1.5</td>
<td>0.57</td>
<td>0.84</td>
<td>0.88</td>
</tr>
<tr>
<td>2.0</td>
<td>0.38</td>
<td>0.75</td>
<td>0.81</td>
</tr>
<tr>
<td>2.5</td>
<td>0.22</td>
<td>0.63</td>
<td>0.70</td>
</tr>
<tr>
<td>3.0</td>
<td>0.10</td>
<td>0.47</td>
<td>0.56</td>
</tr>
<tr>
<td>3.5</td>
<td>0.04</td>
<td>0.32</td>
<td>0.40</td>
</tr>
<tr>
<td>4.0</td>
<td>0.01</td>
<td>0.17</td>
<td>0.25</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.09</td>
<td>0.14</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.04</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Figure 4.7: DLS-OCSVM simulation results for $p = 10$ with uncorrelated multivariate normal data.

The DLS-OCSVM achieves superior performance compared to LS-OCSVM and SVDD, with LS-OCSVM additionally outperforming SVDD.
Low-dimensional setting: $p = 20$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 20$ and $N = 100$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 4.8 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DLS-OCSVM</th>
<th>LS-OCSVM</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.93</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.6</td>
<td>0.86</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.85</td>
<td>0.92</td>
<td>0.93</td>
</tr>
<tr>
<td>0.8</td>
<td>0.83</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>0.9</td>
<td>0.80</td>
<td>0.91</td>
<td>0.92</td>
</tr>
<tr>
<td>1.0</td>
<td>0.78</td>
<td>0.91</td>
<td>0.92</td>
</tr>
<tr>
<td>1.5</td>
<td>0.63</td>
<td>0.86</td>
<td>0.88</td>
</tr>
<tr>
<td>2.0</td>
<td>0.46</td>
<td>0.79</td>
<td>0.81</td>
</tr>
<tr>
<td>2.5</td>
<td>0.28</td>
<td>0.69</td>
<td>0.70</td>
</tr>
<tr>
<td>3.0</td>
<td>0.16</td>
<td>0.55</td>
<td>0.58</td>
</tr>
<tr>
<td>3.5</td>
<td>0.08</td>
<td>0.40</td>
<td>0.43</td>
</tr>
<tr>
<td>4.0</td>
<td>0.03</td>
<td>0.27</td>
<td>0.29</td>
</tr>
<tr>
<td>4.5</td>
<td>0.01</td>
<td>0.15</td>
<td>0.17</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.07</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Figure 4.8: DLS-OCSVM simulation results for $p = 20$ with uncorrelated multivariate normal data.

DLS-OCSVM outperforms both LS-OCSVM and SVDD, with LS-OCSVM and SVDD having similar levels of performance.
High-dimensional setting: \( p = 100 \) and \( N = 40 \)

In this section we present simulation results for a distribution shift where \( p = 100 \) and \( N = 40 \). For this scenario, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 4.14 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>DLS-OCSVM</th>
<th>LS-OCSVM</th>
<th>SVDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.91</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.90</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.89</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.88</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.86</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.9</td>
<td>0.85</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>1.0</td>
<td>0.83</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>1.5</td>
<td>0.74</td>
<td>0.91</td>
<td>0.92</td>
</tr>
<tr>
<td>2.0</td>
<td>0.61</td>
<td>0.89</td>
<td>0.90</td>
</tr>
<tr>
<td>2.5</td>
<td>0.48</td>
<td>0.85</td>
<td>0.87</td>
</tr>
<tr>
<td>3.0</td>
<td>0.36</td>
<td>0.81</td>
<td>0.83</td>
</tr>
<tr>
<td>3.5</td>
<td>0.23</td>
<td>0.75</td>
<td>0.77</td>
</tr>
<tr>
<td>4.0</td>
<td>0.14</td>
<td>0.67</td>
<td>0.68</td>
</tr>
<tr>
<td>4.5</td>
<td>0.08</td>
<td>0.57</td>
<td>0.58</td>
</tr>
<tr>
<td>5.0</td>
<td>0.04</td>
<td>0.46</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Figure 4.9: DLS-OCSVM simulation results for \( p = 100 \) with uncorrelated multivariate normal data.

The performance of DLS-OCSVM is significantly better than LS-OCSVM and SVDD, with LS-OCSVM and SVD exhibiting similar performance outcomes.
In this section, we evaluate the performance of DLS-OCSVM with different activation functions. We run some simulations using several activation functions within the neural network to see if our proposal will encounter difficulties with activation functions other than ReLU. We generate from a multivariate normal distribution with the mean vector set to zero \( \mu_0 = 0 \) and identity covariance matrix \( \Sigma_0 = I_p \). We consider several low-dimensional settings with dimensions \( p = 5, p = 10, \) and \( p = 20 \), where \( N = 100 \) is the number of observations originating from the target class. We also consider a high-dimensional scenario with dimensions \( p = 100 \), where \( N = 40 \) is the number of observations from the target class. We compare the performances of DLS-OCSVM based on the probability of the Type II error using ReLU against logistic sigmoid and hyperbolic tangent activation functions.
In this section we present simulation results for a distribution shift where \( p = 5 \) and \( N = 100 \) for DLS-OCSVM using ReLU, logistic sigmoid, and hyperbolic tangent activation functions. The Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 4.15 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>ReLU</th>
<th>Sigmoid</th>
<th>Tanh</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.92</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.90</td>
<td>0.91</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td>0.5</td>
<td>0.87</td>
<td>0.86</td>
<td>0.87</td>
</tr>
<tr>
<td>0.6</td>
<td>0.85</td>
<td>0.84</td>
<td>0.85</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.81</td>
<td>0.83</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.79</td>
<td>0.79</td>
</tr>
<tr>
<td>0.9</td>
<td>0.78</td>
<td>0.76</td>
<td>0.77</td>
</tr>
<tr>
<td>1.0</td>
<td>0.74</td>
<td>0.74</td>
<td>0.74</td>
</tr>
<tr>
<td>1.5</td>
<td>0.58</td>
<td>0.55</td>
<td>0.56</td>
</tr>
<tr>
<td>2.0</td>
<td>0.38</td>
<td>0.34</td>
<td>0.37</td>
</tr>
<tr>
<td>2.5</td>
<td>0.22</td>
<td>0.19</td>
<td>0.20</td>
</tr>
<tr>
<td>3.0</td>
<td>0.10</td>
<td>0.08</td>
<td>0.09</td>
</tr>
<tr>
<td>3.5</td>
<td>0.04</td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>4.0</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Figure 4.10: DLS-OCSVM simulation results for \( p = 5 \) with alternative activation functions.

Sigmoid performs slightly better than ReLU and Tanh, while ReLU and Tanh exhibit comparable levels of performance.
In this section we present simulation results for a distribution shift where \( p = 10 \) and \( N = 100 \) for DLS-OCSVM using ReLU, logistic sigmoid, and hyperbolic tangent activation functions. The Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0 \} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 4.16 shown below.

**Figure 4.11: DLS-OCSVM simulation results for \( p = 10 \) with alternative activation functions.**

Performance across the three activation functions is evenly matched, suggesting that in this dimensional setting, selecting one activation function over another offers no distinct advantages.
Performance for $p = 20$ and $N = 100$

In this section we present simulation results for a distribution shift where $p = 20$ and $N = 100$ for DLS-OCSVM using ReLU, logistic sigmoid, and hyperbolic tangent activation functions. The Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 4.12 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>ReLU</th>
<th>Sigmoid</th>
<th>Tanh</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.93</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.90</td>
<td>0.89</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.88</td>
<td>0.87</td>
</tr>
<tr>
<td>0.6</td>
<td>0.86</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>0.7</td>
<td>0.85</td>
<td>0.84</td>
<td>0.84</td>
</tr>
<tr>
<td>0.8</td>
<td>0.83</td>
<td>0.81</td>
<td>0.81</td>
</tr>
<tr>
<td>0.9</td>
<td>0.80</td>
<td>0.80</td>
<td>0.79</td>
</tr>
<tr>
<td>1.0</td>
<td>0.78</td>
<td>0.76</td>
<td>0.77</td>
</tr>
<tr>
<td>1.5</td>
<td>0.63</td>
<td>0.59</td>
<td>0.60</td>
</tr>
<tr>
<td>2.0</td>
<td>0.46</td>
<td>0.41</td>
<td>0.43</td>
</tr>
<tr>
<td>2.5</td>
<td>0.28</td>
<td>0.25</td>
<td>0.26</td>
</tr>
<tr>
<td>3.0</td>
<td>0.16</td>
<td>0.12</td>
<td>0.14</td>
</tr>
<tr>
<td>3.5</td>
<td>0.08</td>
<td>0.05</td>
<td>0.06</td>
</tr>
<tr>
<td>4.0</td>
<td>0.03</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>4.5</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Figure 4.12: DLS-OCSVM simulation results for $p = 20$ with alternative activation functions.

Sigmoid and Tanh marginally outperform ReLU in the range of $1.5 \leq \delta \leq 3.5$, while both Sigmoid and Tanh exhibit similar levels of performance.
Performance for \( p = 100 \) and \( N = 40 \)

In this section we present simulation results for a distribution shift where \( p = 100 \) and \( N = 40 \) for DLS-OCSVM using ReLU, logistic sigmoid, and hyperbolic tangent activation functions. The Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 4.13 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>ReLU</th>
<th>Sigmoid</th>
<th>Tanh</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.92</td>
<td>0.92</td>
</tr>
<tr>
<td>0.4</td>
<td>0.91</td>
<td>0.91</td>
<td>0.91</td>
</tr>
<tr>
<td>0.5</td>
<td>0.90</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>0.6</td>
<td>0.89</td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td>0.7</td>
<td>0.88</td>
<td>0.88</td>
<td>0.88</td>
</tr>
<tr>
<td>0.8</td>
<td>0.86</td>
<td>0.86</td>
<td>0.86</td>
</tr>
<tr>
<td>0.9</td>
<td>0.85</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>1.0</td>
<td>0.83</td>
<td>0.85</td>
<td>0.86</td>
</tr>
<tr>
<td>1.5</td>
<td>0.74</td>
<td>0.76</td>
<td>0.78</td>
</tr>
<tr>
<td>2.0</td>
<td>0.61</td>
<td>0.64</td>
<td>0.68</td>
</tr>
<tr>
<td>2.5</td>
<td>0.48</td>
<td>0.52</td>
<td>0.59</td>
</tr>
<tr>
<td>3.0</td>
<td>0.36</td>
<td>0.39</td>
<td>0.47</td>
</tr>
<tr>
<td>3.5</td>
<td>0.23</td>
<td>0.27</td>
<td>0.36</td>
</tr>
<tr>
<td>4.0</td>
<td>0.17</td>
<td>0.17</td>
<td>0.27</td>
</tr>
<tr>
<td>4.5</td>
<td>0.08</td>
<td>0.11</td>
<td>0.19</td>
</tr>
<tr>
<td>5.0</td>
<td>0.04</td>
<td>0.06</td>
<td>0.13</td>
</tr>
</tbody>
</table>

Figure 4.13: DLS-OCSVM simulation results for \( p = 100 \) with alternative activation functions.

ReLU achieves superior performance compared to Sigmoid and Tanh in this high dimensional setting, with Sigmoid also exceeding Tanh in effectiveness.
Summary of Simulation Results

In this section, we evaluated the performance of DLS-OCSVM against competing methods by considering several types of scenarios in both low-dimensional settings and high-dimensional settings.

First, we considered simulations with uncorrelated multivariate normal data. In these simulations, it was observed that in low-dimensional settings LS-OCSVM achieves better performance than SVDD. However, as the dimensions are increased this difference in performance gradually diminishes. In the high-dimensional setting, these two methods perform comparably. In all scenarios under consideration, the DLS-OCSVM significantly outperforms both LS-OCSVM and SVDD.

In the subsequent set of simulations, we examined the performance of the DLS-OCSVM with different activation functions. In low-dimensional settings, the performance of the Sigmoid activation function suggests it may be used as an alternative to ReLU. However, in high-dimensional setting, ReLU is recommended due to its superior performance. Tanh is not recommended due to its overall lower performance.
We illustrate the real-world application of DLS-OCSVM, and compare its performance against LS-OCSVM and SVDD, using the ‘breast cancer dataset.’ The dataset was created by Dr. William H. Wolberg [58], a physician at the University of Wisconsin. The breast cancer dataset is publicly available and can be found in UCI Machine Learning Repository. There are two target classes in the dataset, labeled as 2 or 4. The dataset consists of nine numerical features measured on 683 patients: 444 diagnosed with benign cancer (target class 2) and 239 diagnosed with malignant cancer (target class 4).

The goal is to train a model with data exclusively from patients with benign cancer, and predict whether unseen patients have either benign or malignant cancer. We use the first 80 observations of the target class 2 data (benign cancer group) as our training set. This yields $p = 9$ variables with $N = 80$ observations. The distribution of this data is unknown and demonstrates the application of the proposed method on data that is not multivariate normal.

The test set is formed by using the last five observations of the target class 2 data (benign cancer group) and the first eight observations of the target class 4 data (malignant cancer group). This yields a total of 13 observations in the test set.

We applied DLS-OCSVM, LS-OCSVM, and SVDD to the 13 observations in the test set and compare their performance in detecting outliers (malignant cancer group). For DLS-OCSVM, we use the same neural network architecture outlined in the Performance Settings Section. For LS-OCSVM, we compute the $\alpha$’s using (2.24). In the case of SVDD, the Lagrange multipliers are obtained using a quadratic programming solver. The dual problem is optimized in this manner due to the nonexistence of a closed-form solution, a distinction from LS-OCSVM. The results for LS-OCSVM and SVDD are shown in Figures 4.14 and 4.15, respectively. Calculating the distance
statistic \( d \) using Equation (4.43) for DLS-OCSVM, the results are shown in Figure 4.16.

While all of the models were able to correctly classify the targets and outliers respectively, there are some notable differences. The first five observations, which belong to the target class 2 data (benign cancer group), are correctly classified by LS-OCSVM. However, there is some variation between these observations with respect to their distance to the boundary. Specifically, the fourth observation is much closer to the boundary than the other observations from the target class. Furthermore, while all of the outliers are properly classified, the second outlier observation is significantly closer to the boundary than the others.

![Figure 4.14: OCC using LS-OCSVM for the breast cancer dataset example.](image)

Figure 4.14: OCC using LS-OCSVM for the breast cancer dataset example.
For SVDD, the target class 2 data (benign cancer group) follows a similar pattern as LS-OCSVM. In the case of the first three observations, there exists an approximately uniform distance to the boundary, whereas the fourth observation stands out due to its considerably closer proximity to the boundary in relation to the other targets. However, the fifth observation from the target class exhibits a more substantial separation from the boundary than the first four data points, signifying a salient distinction. Furthermore, it is noteworthy that all outlier observations are positioned markedly closer to the boundary compared to LS-OCSVM, and a minimal degree of variation is evident in the outlier observations.

Figure 4.15: OCC using SVDD for the breast cancer dataset example.
Alternatively, DLS-OCSVM has very little variation in the first five target observations with respect to their distance to the boundary. In contrast to both LS-OCSVM and SVDD, the outliers demonstrate a notable increase in distance from the boundary. In other words, the DLS-OCSVM boundary separating the target class from outliers is characterized by a considerably greater degree of tightness in comparison to both LS-OCSVM and SVDD. This demonstrates the effectiveness of the proposed one-class classifier in a real-world application when dealing with data from an unknown distribution.

Figure 4.16: OCC using DLS-OCSVM for the breast cancer dataset example.
CHAPTER 5: DEEP SUPPORT MATRIX DATA DESCRIPTION

Introduction

The majority of one-class classification techniques are vector-based methods that are designed to work with tabular data. However, the native form of many types of data, such as grid structured data or images, are naturally represented as matrices or second-order tensors. For data that naturally assumes a matrix form, the process of vectorization is a commonly used technique that entails the transformation of matrix-based data into one-dimensional arrays. While this approach can play a pivotal role in expediting numerical operations and facilitating parallel processing, it inherently leads to structural information loss embedded within the original data structure. The process of converting a matrix into a one-dimensional array interferes with the spatial relationships among its constituent elements, eradicating the nuanced patterns and contextual dependencies within the native matrix representation. Furthermore, vectorization often involves dimensionality reduction and sequential processing, disregarding the inherent structure and sequential dependencies of the original data. Therefore, vectorization poses a significant challenge when working in domains where the preservation of spatial and sequential relationships plays an important role for accurate analysis and interpretation of the data.

The prominent one-class classification technique Support Vector Data Description (SVDD) introduced by Tax and Duin [53] requires vectorization when dealing with higher ordered objects, such as matrices. To circumvent this type of operation, Maboudou-Tchao developed Support Matrix Data Description (SMDD) [37] as a one-class classifier for matrices that does not require input data to be vectorized. For both SVDD and SMDD, kernel functions are used to map the training data from the original input space to a bounded spherical region in the feature space, effectively isolating outliers beyond this closed boundary. Using kernels, flexible and effective decision boundaries
can be attained to create a closed boundary and separate the target class from outliers. Since the native form of the target class typically cannot be adequately characterized by a spherical distribution, using kernel functions to map the training data into a higher dimensional feature space is one common approach used to solve one-class classification problems.

Despite the prevalent use of kernels, alternative methods exist for identifying feature mappings for the target class. Neural networks are one alternative that can be used to learn compact and rich feature representations of the target class. The issue is that conventional feed forward neural networks require vectorization since observations need to be represented as vectors to be used as input to the network. Alternatively, Gao et al. [16] developed the matrix neural network, which is a recently proposed neural network architecture that does not require vectorization of the input data whose native form are matrices. However, in the original formulation of matrix neural network, matrices were used as input to the network while the output of the network was still in vector form. The matrix neural network architecture was then extended by Do et al. [12] to accommodate a second-order tensor structure for both the input and output of the network, thereby eliminating any requirement for vectorization.
Methodology

We introduce a novel method that combines the rich feature mapping capabilities of matrix neural networks combined with Support Matrix Data Description (SMDD) to tackle one-class classification challenges involving second-order tensors. Deep Support Matrix Data Description (DSMDD) integrates matrix neural network and SMDD in an end-to-end framework for the one-class classification task without the need to vectorize second-order tensors. The goal of DSMDD is to jointly learn a matrix neural network’s weight parameters while minimizing the volume of a data enclosing hypersphere. The theoretical results of the DSMDD extend the results of Ruff et al. [48] to second-order tensors.

Consider a training dataset $X = \{X_1, X_2, ..., X_N\} \subseteq \mathcal{X}$ where $N$ is the number of matrices and $\mathcal{X} \subseteq \mathbb{R}^{n \times p}$ is an original input space for which we want a description. A description refers to a model characterized by a closed boundary in the form of a minimum volume hypersphere, or a “matrix sphere”, that distinguishes the target class from outliers. For this input space, we will define a corresponding output space $F \subseteq \mathbb{R}^{q \times d}$. Let $\phi(\cdot; \mathcal{U}, \mathcal{V})$ be a matrix neural network that maps $\mathcal{X} \mapsto F$ with $L \in \mathbb{N}$ layers. The network consists of $L - 1$ hidden layers and a final output layer, and the architecture is characterized by a set of weight matrices $\{\mathcal{U}, \mathcal{V}\} = \{U[1], V[1], ..., U[L], V[L]\}$ corresponding to the (row-mapping) weights $U[\ell]$ and (column-mapping) weights $V[\ell]$ of layer $\ell \in \{1, 2, ..., L\}$, respectively. In other words, $\phi(X; \mathcal{U}, \mathcal{V})$ denotes the feature representation of $X \in \mathcal{X}$ given by network $\phi$ with parameters $\mathcal{U}$ and $\mathcal{V}$.

The goal is to jointly learn the matrix neural network weight parameters $\mathcal{U}, \mathcal{V}$ while encapsulating the target class using a minimum volume hypersphere characterized by its center $C$, which is a matrix, and a radius $R > 0$ that contains all, or most, of the target class matrices. The formal mathematical representation of this model for finding the smallest hypersphere is is given by the
following constrained optimization problem:

\[
\min_{R, \xi, \mathcal{U}, \mathcal{V}} \quad R^2 + \frac{1}{\nu} \sum_{i=1}^{N} \xi_i + \frac{\lambda}{2} \sum_{\ell=1}^{L} (||\mathcal{U}^\ell||_F^2 + ||\mathcal{V}^\ell||_F^2)
\]

subject to \[ ||\phi(X_i; \mathcal{U}, \mathcal{V}) - C||_F^2 \leq R^2 + \xi_i, \quad i = 1, 2, \ldots, N \] (5.1)

and \[ \xi_i \geq 0, \quad i = 1, 2, \ldots, N \]

where \( C \) is the center of the matrix sphere, \( R \) is the radius of the matrix sphere, \( \xi_i \) are the slack variables, and the tuning parameter \( \nu \in (0, 1] \) is introduced to control the influence of the slack variables, consequently balancing boundary violations of the target class and the volume of the hypersphere. The last term in the objective introduces a weight decay regularization component applied to the network weights \( \mathcal{U} \) and \( \mathcal{V} \) with a tuning parameter \( \lambda > 0 \), where \( || \cdot ||_F \) is the Frobenius norm. The hypersphere center \( C \) is initialized using a small batch of the training data after an initial forward pass through the matrix neural network. It is held fixed and not treated as a learnable parameter during the training procedure.

The DSMDD constrained optimization problem (5.1) can equivalently be expressed as the following unconstrained optimization problem:

\[
\min_{R, \mathcal{U}, \mathcal{V}} \quad R^2 + \frac{1}{\nu N} \sum_{i=1}^{N} \max (0, ||\phi(X_i; \mathcal{U}, \mathcal{V}) - C||_F^2 - R^2) + \frac{\lambda}{2} \sum_{\ell=1}^{L} (||\mathcal{U}^\ell||_F^2 + ||\mathcal{V}^\ell||_F^2) \] (5.2)

Note that this optimization problem uses the hinge loss, which is not a smooth or differentiable function. Therefore, it is not possible to directly find the gradient. Furthermore, since we are using a matrix neural network with nonlinear activation functions for the feature mapping, we have an unconstrained optimization problem that is not convex. However, we can use subgradient optimization to find the subgradients.
We can define the loss function, denoted by $\mathcal{L}$, as follows:

$$
\mathcal{L}(R, U, V) = R^2 + \frac{1}{\nu N} \sum_{i=1}^{N} \max \left(0, ||\phi(X_i; U, V) - C||_F^2 - R^2\right) + \frac{\lambda}{2} \sum_{\ell=1}^{L} \left(||U[\ell]||_F^2 + ||V[\ell]||_F^2\right)
$$

(5.3)

To optimize this problem, we use a stochastic subgradient (SG) method. Define the DSMDD loss function for SG, denoted by $\mathcal{L}_s$, at a random point $X_*$ as follows:

$$
\mathcal{L}_s(R, U, V) = R^2 + \frac{1}{\nu} \max \left(0, ||\phi(X_*; U, V) - C||_F^2 - R^2\right) + \frac{\lambda}{2} \sum_{\ell=1}^{L} \left(||U[\ell]||_F^2 + ||V[\ell]||_F^2\right)
$$

(5.4)

A given test point $Z \in \mathcal{X}$ is in the target class if the Euclidean distance in the matrix space (i.e., the Frobenius norm) of the point $Z$ to the center of the hypersphere is less than the radius. In other words, the test point $Z$ will be in the target class if the following condition is satisfied:

$$
S(Z) = ||\phi(Z; U^*, V^*) - C||_F^2 \leq R^{*2}
$$

(5.5)

where $U^*$, $V^*$, and $R^{*2}$ are the matrix neural network weight parameters and radius squared of a trained model, respectively. Furthermore, we denote the anomaly score for a given test point $Z$ by $S(Z)$.

The SG DSMDD training procedure is summarized in Algorithm 9.
Algorithm 9 DSMDD SG Algorithm

**Input:** Training dataset $X = \{X_1, X_2, ..., X_N\} \subseteq \mathcal{X}$ with $X_i \in \mathbb{R}^{n \times p}$ and hyperparameters $\lambda, \nu$.

**Output:** Hypersphere radius $R^*$ and network weights $U^*, V^*$ of a trained model.

1: **Initialize:**
   - Hypersphere radius $R$, network weights $W$, and hypersphere center $C$ fixed as the average of a small batch of training data after an initial forward pass through the network $\phi(\cdot; U, V)$.
2: **for** each epoch **do**
3:   - Randomly sample a training observation $X \in \mathcal{X}$.
4:   - Calculate $\phi(X; U, V)$ by forward propagation through DSMDD model.
5:   - Calculate $L_s$ using Equation (5.4).
6:   - Update the network weights and hypersphere center using the Adam algorithm.
7:   - Update the radius $R$ as the $\nu$-quantile value of the target distances from the hypersphere center.
8: **end for**
9: **return** $R^*, U^*$, and $V^*$

In the following section, we systematically derive the mathematical equations for DSMDD using a simple matrix neural network with one hidden layer. This is followed by an analogous derivation adapted to a more complex model with two hidden layers. We conclude by generalizing these results to apply to arbitrarily deep matrix neural networks with $L \in \mathbb{N}$ hidden layers.

**Matrix Neural Network with One Hidden Layer**

For the mathematical derivations, we design a simple matrix neural network with a single hidden layer having rectified linear unit (ReLU) activation functions in the hidden layer, denoted by $g_1(\cdot)$, and linear activation functions in the output layer, denoted by $g_2(\cdot)$.

Let $Z^{[1]}$ denote the weighted input of the hidden layer. The output of the hidden layer, denoted by $A^{[1]}$, is obtained after applying $g_1(\cdot)$ in an element-wise manner to the weighted input $Z^{[1]}$. For the output layer, the weighted input is denoted by $Z^{[2]}$, and the network output, defined as $\phi(X; U, V) = A^{[2]}$, is obtained after $g_2(\cdot)$ is applied element-wise to $Z^{[2]}$. 
During the forward propagation phase, an observation \( X_* \) is initially processed at the input layer, then passed through the hidden layer, and finally is directed to the output layer. The mathematical functions detailing this progression are shown below.

\[
\begin{align*}
Z^{[1]} &= U^{[1]} X_* V^{[1]T} \\
A^{[1]} &= g_1(Z^{[1]}) \\
Z^{[2]} &= U^{[2]} A^{[1]} V^{[2]T} \\
\phi(X_*; U, V) &= A^{[2]} = g_2(Z^{[2]})
\end{align*}
\]  

Through the application of the chain rule from matrix differential calculus, backpropagation methodically propagates error signals backward through the network from the output layer to the input layer, thereby facilitating weight adjustments across the entire network.

Unlike conventional artificial neural networks, which have a single weight matrix in each layer, matrix neural networks have two weight matrices per layer. As a result, errors associated with both weight matrices in a given layer must be considered. By initially defining the errors that are propagated backwards through each layer, a concise representation of the subgradients for the matrix neural network’s weight matrices can be obtained. The proof for Proposition 5.0.1 can be found in Appendix E.

Define \( \delta^{[2]}_{u} \) as the error propagated backwards through the output layer’s activation function \( g_2 \) for the (row-mapping) weight matrix \( U^{[2]} \) as follows:

\[
\delta^{[2]}_{u} = 2 \nu ( \phi(X_*; U, V) - c) \circ g'_2(Z^{[2]}) V^{[2]}
\]

where \( \circ \) denotes the Hadamard product.
Similarly, let $\delta^{[2]}_v$ denote the error propagated backwards through the output layer’s activation function $g_2$ for the (column-mapping) weight matrix $V^{[2]}$ as follows:

$$
\delta^{[2]}_v = \frac{2}{\nu} \left( \phi (X_s; U, V) - c \right) \circ g'_2 (Z^{[2]})^T U^{[2]}
$$

(5.11)

Next, define $\delta^{[1]}_u$ as the error propagated backwards through the hidden layer’s activation function $g_1$ for the (row-mapping) weight matrix $U^{[1]}$ as follows:

$$
\delta^{[1]}_u = U^{[2]T} \delta^{[2]}_v \circ g'_1 (Z^{[1]}) V^{[1]}
$$

(5.12)

Similarly, let $\delta^{[1]}_v$ denote the error propagated backwards through the hidden layer’s activation function $g_1$ for the (column-mapping) weight matrix $V^{[1]}$ as follows:

$$
\delta^{[1]}_v = V^{[2]T} \delta^{[2]}_v \circ g'_1 (Z^{[1]})^T U^{[1]}
$$

(5.13)

The subgradients for the hypersphere radius and matrix neural network weight matrices, respectively, are presented in Proposition 5.0.1.
Proposition 5.0.1

\[
\partial R(L_s) = \begin{cases} 
2R, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
2R\left(1 - \frac{1}{\beta}\right), & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0
\end{cases}
\] (5.14)

\[
\partial U_{[2]}(L_s) = \begin{cases} 
\lambda U^{[2]}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_{u}^{[2]} A^{[1]T}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0
\end{cases}
\] (5.15)

\[
\partial V_{[2]}(L_s) = \begin{cases} 
\lambda V^{[2]}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_{v}^{[2]} A^{[1]}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0
\end{cases}
\] (5.16)

\[
\partial U_{[1]}(L_s) = \begin{cases} 
\lambda U^{[1]}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_{u}^{[1]} X^T, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0
\end{cases}
\] (5.17)

\[
\partial V_{[2]}(L_s) = \begin{cases} 
\lambda V^{[1]}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_{v}^{[1]} X, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0
\end{cases}
\] (5.18)

The SG uses the following iterative updates for matrix neural network layers \( \ell = \{1, 2\} \):

\[
U_{(t+1)}^{[\ell]} = U_{(t)}^{[\ell]} - \eta_1 \partial U_{(t)}^{[\ell]}(L_s)
\]

\[
V_{(t+1)}^{[\ell]} = V_{(t)}^{[\ell]} - \eta_1 \partial V_{(t)}^{[\ell]}(L_s)
\]

\[
R_{(t+1)} = R_{(t)} - \eta_2 \partial R(L_s)
\]

where \( \eta_1 \) and \( \eta_2 \) are defined as the learning rates for the matrix network weights and hypersphere radius, respectively.
Matrix Neural Network with Two Hidden Layers

In this section, we consider the mathematical derivations for a matrix neural network with two hidden layers. This network has rectified linear unit (ReLU) activation functions in the hidden layers, denoted by \( g_i(\cdot), i = 1, 2 \), and linear activation functions in the output layer, denoted by \( g_3(\cdot) \).

Let \( Z^{[1]} \) denote the weighted input of the first hidden layer. The output of the first hidden layer, denoted by \( A^{[1]} \), is obtained after applying \( g_1(\cdot) \) in an element-wise manner to the weighted input \( Z^{[1]} \). Similarly, \( Z^{[2]} \) represents the weighted input of the second hidden layer. The output of the second hidden layer, denoted by \( A^{[2]} \), is obtained after applying \( g_2(\cdot) \) element-wise to the weighted input \( Z^{[2]} \). For the output layer, the weighted input is denoted by \( Z^{[3]} \), and the network output, defined as \( \phi(\cdot; U, V) = A^{[3]} \), is obtained after \( g_3(\cdot) \) is applied element-wise to \( Z^{[3]} \).

During the forward propagation phase, an observation \( X_* \) is initially processed at the input layer, then passed through each of the hidden layers, and finally is directed to the output layer. The mathematical functions detailing this progression are shown below.

\[
Z^{[1]} = U^{[1]} X_* V^{[1]^T} \quad (5.19)
\]

\[
A^{[1]} = g_1 (Z^{[1]}) \quad (5.20)
\]

\[
Z^{[2]} = U^{[2]} A^{[1]} V^{[2]^T} \quad (5.21)
\]

\[
A^{[2]} = g_2 (Z^{[2]}) \quad (5.22)
\]

\[
Z^{[3]} = U^{[3]} A^{[2]} V^{[3]^T} \quad (5.23)
\]

\[
\phi(X_*; U, V) = A^{[3]} = g_3 (Z^{[3]}) \quad (5.24)
\]
Through the application of the chain rule from matrix differential calculus, backpropagation methodically propagates error signals backward through the network from the output layer to the input layer, thereby facilitating weight adjustments across the entire network.

Unlike conventional artificial neural networks, which have a single weight matrix in each layer, matrix neural networks have two weight matrices per layer. As a result, errors associated with both weight matrices in a given layer must be considered. By initially defining the errors that are propagated backwards through each layer, a concise representation of the subgradients for the matrix neural network’s weight matrices can be obtained. The proof for Proposition 5.0.3 can be found in Appendix E.

Define $\delta_u^{[3]}$ as the error propagated backwards through the output layer’s activation function $g_3$ for the (row-mapping) weight matrix $U^{[3]}$ as follows:

$$
\delta_u^{[3]} = \frac{2}{\nu} \left( \phi(X_s; U, V) - c \right) \circ g_3'(Z^{[3]}) \, V^{[3]} \tag{5.25}
$$

where $\circ$ denotes the Hadamard product.

Similarly, let $\delta_v^{[3]}$ denote the error propagated backwards through the output layer’s activation function $g_3$ for the (column-mapping) weight matrix $V^{[3]}$ as follows:

$$
\delta_v^{[3]} = \frac{2}{\nu} \left( \phi(X_s; U, V) - c \right) \circ g_3'(Z^{[3]})^T \, U^{[3]} \tag{5.26}
$$

Next, define $\delta_u^{[2]}$ as the error propagated backwards through the second hidden layer’s activation function $g_2$ for the (row-mapping) weight matrix $U^{[2]}$ as follows:

$$
\delta_u^{[2]} = U^{[3]T} \delta_u^{[3]} \circ g_2'(Z^{[2]}) \, V^{[2]} \tag{5.27}
$$
Similarly, let $\delta_v^{[2]}$ denote the error propagated backwards through the second hidden layer’s activation function $g_2$ for the (column-mapping) weight matrix $V^{[2]}$ as follows:

$$
\delta_v^{[2]} = V^{[3]T} \delta_v^{[3]} \circ g_2' \left( Z^{[2]} \right)^T U^{[2]}
$$

(5.28)

Next, define $\delta_u^{[1]}$ as the error propagated backwards through the first hidden layer’s action function $g_1$ for the (row-mapping) weight matrix $U^{[1]}$ as follows:

$$
\delta_u^{[1]} = U^{[2]T} \delta_u^{[2]} \circ g_1' \left( Z^{[1]} \right) V^{[1]}
$$

(5.29)

Similarly, let $\delta_v^{[1]}$ denote the error propagated backwards through the second hidden layer’s activation function $g_2$ for the (column-mapping) weight matrix $V^{[2]}$ as follows:

$$
\delta_v^{[1]} = V^{[2]T} \delta_v^{[2]} \circ g_1' \left( Z^{[1]} \right)^T U^{[1]}
$$

(5.30)

The subgradients for the hypersphere radius and matrix neural network weight matrices, respectively, are presented in Proposition 5.0.3.
Proposition 5.0.2

\[ \partial R (L_s) = \begin{cases} 
2R, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
2R \left( 1 - \frac{1}{\nu} \right), & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0 
\end{cases} \]  

(5.31)

\[ \partial U[3] (L_s) = \begin{cases} 
\lambda U[3], & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_u^{[3]} A^{[2]^T}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0 
\end{cases} \]  

(5.32)

\[ \partial V[3] (L_s) = \begin{cases} 
\lambda V[3], & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_v^{[3]} A^{[2]}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0 
\end{cases} \]  

(5.33)

\[ \partial U[2] (L_s) = \begin{cases} 
\lambda U[2], & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_u^{[2]} A^{[1]^T}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0 
\end{cases} \]  

(5.34)

\[ \partial V[2] (L_s) = \begin{cases} 
\lambda V[2], & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_v^{[2]} A^{[1]}, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0 
\end{cases} \]  

(5.35)

\[ \partial U[1] (L_s) = \begin{cases} 
\lambda U[1], & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_u^{[1]} X^T, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0 
\end{cases} \]  

(5.36)

\[ \partial V[1] (L_s) = \begin{cases} 
\lambda V[1], & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 \leq 0 \\
\delta_v^{[1]} X, & \text{if } ||\phi(X_i; W) - C||_F^2 - R^2 > 0 
\end{cases} \]  

(5.37)
The SG uses the following iterative updates for matrix neural network layers $\ell = \{1, 2, 3\}$:

$$U^{[\ell]}_{(t+1)} = U^{[\ell]}_{(t)} - \eta_1 \partial_{U^{[\ell]}_{(t)}} (\mathcal{L}_s)$$

$$V^{[\ell]}_{(t+1)} = V^{[\ell]}_{(t)} - \eta_1 \partial_{V^{[\ell]}_{(t)}} (\mathcal{L}_s)$$

$$R_{(t+1)} = R_{(t)} - \eta_2 \partial_R (\mathcal{L}_s)$$

where $\eta_1$ and $\eta_2$ are defined as the learning rates for the matrix network weights and hypersphere radius, respectively.

**Matrix Neural Network with $L$ Hidden Layers**

It is straightforward to generalize the model to incorporate deeper matrix neural network architectures. Consider a matrix neural network with $L \in \mathbb{N}$ hidden layers ($\ell = 1, 2, ..., L$) and one output layer ($\ell = L + 1$). The network architecture consists of a set of (row-mapping) weight matrices $\mathcal{U} = \{U^{[1]}, U^{[2]}, ..., U^{[L+1]}\}$ and (column-mapping) weight matrices $\mathcal{V} = \{V^{[1]}, V^{[2]}, ..., V^{[L+1]}\}$, corresponding to the (row-mapping) weights $U^{[\ell]}$ and (column-mapping) $V^{[\ell]}$ weights of layer $\ell \in \{1, 2, ..., L + 1\}$, respectively.

The generalized forward propagation equations for matrix neural network layers $\ell = 1, 2, ..., L + 1$ can be expressed as follows:

$$Z^{[\ell]} = U^{[\ell]} A^{[\ell-1]} V^{[\ell]T} \quad \text{(5.38)}$$

$$A^{[\ell]} = g_{\ell} (Z^{[\ell]}) \quad \text{(5.39)}$$

where the matrix neural network input is defined as $A^{[0]} = X_s$. 

139
For output layer $\ell = L + 1$, the error propagated backwards through the output layer’s activation function $g_{L+1}$ can be expressed as follows:

$$
\delta^{[L+1]}_u = \frac{2}{\nu} (\phi (X_\ast; \mathcal{U}, \mathcal{V}) - C) \circ g'_{L+1} (Z^{[L+1]}) V^{[L+1]} \\
\delta^{[L+1]}_v = \frac{2}{\nu} \left((\phi (X_\ast; \mathcal{U}, \mathcal{V}) - C) \circ g'_{L+1} (Z^{[L+1]}) \right)^T U^{[L+1]}
$$

(5.40)  

(5.41)

where $\circ$ denotes the Hadamard product.

Then, for hidden layers $\ell = 1, 2, ..., L$, the error equations can be expressed as follows:

$$
\delta^{[\ell]}_u = U^{[\ell+1]T} \delta^{[\ell+1]}_u \circ g'_\ell (Z^{[\ell]}) V^{[\ell]}  \\
\delta^{[\ell]}_v = V^{[\ell+1]T} \delta^{[\ell+1]}_v \circ g'_\ell (Z^{[\ell]})^T U^{[\ell]}
$$

(5.42)  

(5.43)

For matrix neural network layers $\ell = 1, 2, ..., L + 1$, the subgradients of the weight parameters are succinctly represented using the equations shown below.

$$
\partial U^{[\ell]} = \delta^{[\ell]}_u A^{[\ell-1]T}  \\
\partial V^{[\ell]} = \delta^{[\ell]}_v A^{[\ell-1]}
$$

(5.44)  

(5.45)
Alternative Objective Function

An alternative objective function can offer a simplified solution compared to the proposed method in the case where the training data is assumed to contain no outliers. In this approach, the optimization problem is simplified by excluding the radius term on the basis that all data representations are assumed to be contained within the boundary of the hypersphere. The goal is still to create a compact and closed boundary that encapsulates the target class within a hypersphere with center \( C \in \mathcal{F} \); however, now this is obtained by exclusively minimizing the squared Frobenius norm corresponding to the average distances from the target class representation matrices to to the hypersphere center in the native matrix space.

Consider a training dataset \( X = \{X_1, X_2, ..., X_N\} \subseteq \mathcal{X} \) where \( N \) is the number of matrices and \( \mathcal{X} \subseteq \mathbb{R}^{n \times p} \) is an original input space for which we want to have a description. For this input space, we will define a corresponding output space \( \mathcal{F} \subseteq \mathbb{R}^{q \times d} \). Let \( \phi(\cdot; U, V) \) be a matrix neural network that maps \( \mathcal{X} \mapsto \mathcal{F} \) with \( L \in \mathbb{N} \) layers. The network consists of \( L - 1 \) hidden layers and a final output layer, and the architecture is characterized by a set of weight matrices \( \{U, V\} = \{U^{[1]}, V^{[1]}, ..., U^{[L]}, V^{[L]}\} \) corresponding to the (row-mapping) weights \( U^{[\ell]} \) and (column-mapping) weights \( V^{[\ell]} \) of layer \( \ell \in \{1, 2, ..., L\} \), respectively. In other words, \( \phi(X; U, V) \) denotes the feature representation of \( X \in \mathcal{X} \) given by network \( \phi \) with parameters \( U \) and \( V \). We define the One-Class Deep Support Matrix Data Description (OC-DSMDD) objective as follows:

\[
\min_{U, V} \frac{1}{2N} \sum_{i=1}^{N} \|\phi(X_i; U, V) - C\|_F^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} (\|U^{[\ell]}\|_F^2 + \|V^{[\ell]}\|_F^2) \tag{5.46}
\]

where \( C \in \mathcal{F} \) is the center of the hypersphere, which is fixed as the average feature representations after an initial forward pass through the network. The last term in the objective is a regularization component for the network weights with tuning parameter \( \lambda > 0 \), where \( \|\cdot\|_F \) is the Frobenius norm.
norm.

Define the OC-DSMDD loss function for SG, denoted by $\mathcal{L}_{s_1}$, at a random point $x_*$ as follows:

$$
\mathcal{L}_{s_1} (U, V) = \frac{1}{2} ||\phi (X_*; U, V) - C||_F^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} (||U^{[\ell]}||_F^2 + ||V^{[\ell]}||_F^2)
$$

(5.47)

For the mathematical derivations, consider a simple matrix neural network with a single hidden layer as was previously outlined in the Methodology Section for DSMDD.

Define $\delta_{u_1}^{[2]}$ as the error propagated backwards through the output layer’s activation function $g_2$ for the (row-mapping) weight matrix $U^{[2]}$ as follows:

$$
\delta_{u_1}^{[2]} = (\phi(X_*; U, V) - C) \circ g'_2 (Z^{[2]}) V^{[2]}
$$

(5.48)

where $\circ$ denotes the Hadamard product.

Similarly, let $\delta_{v_1}^{[2]}$ denote the error propagated backwards through the output layer’s activation function $g_2$ for the (column-mapping) weight matrix $V^{[2]}$ as follows:

$$
\delta_{v_1}^{[2]} = \left( (\phi(X_*; U, V) - C) \circ g'_2 (Z^{[2]}) \right)^T U^{[2]}
$$

(5.49)

Next, define $\delta_{u_1}^{[1]}$ as the error propagated backwards through the hidden layer’s activation function $g_1$ for the (row-mapping) weight matrix $U^{[1]}$ as follows:

$$
\delta_{u_1}^{[1]} = U^{[2]^T} \delta_{u_1}^{[2]} \circ g'_1 (Z^{[1]}) V^{[1]}
$$

(5.50)

Similarly, let $\delta_{v_1}^{[1]}$ denote the error propagated backwards through the hidden layer’s activation function $g_1$ for the (column-mapping) weight matrix $V^{[1]}$ as follows:
function $g_1$ for the (column-mapping) weight matrix $V^{[1]}$ as follows:

$$\delta_{v_1}^{[1]} = V^{[2]T} \delta_{v_1}^{[2]} \circ g'_1 (Z^{[1]})^T U^{[1]}$$  (5.51)

The subgradients for the matrix neural network weight matrices are presented in Proposition 5.0.3, and the proof can be found in Appendix F.

**Proposition 5.0.3**

$$\partial_{U^{[2]}} (\mathcal{L}_{s_1}) = \begin{cases} 
\lambda U^{[2]}, & \text{if } ||\phi(X_i; W) - C||^2_F \leq 0 \\
\delta_{u_1}^{[2]} A^{[1]T}, & \text{if } ||\phi(X_i; W) - C||^2_F > 0
\end{cases}$$  (5.52)

$$\partial_{V^{[2]}} (\mathcal{L}_{s_1}) = \begin{cases} 
\lambda V^{[2]}, & \text{if } ||\phi(X_i; W) - C||^2_F \leq 0 \\
\delta_{v_1}^{[2]} A^{[1]}, & \text{if } ||\phi(X_i; W) - C||^2_F > 0
\end{cases}$$  (5.53)

$$\partial_{U^{[1]}} (\mathcal{L}_{s_1}) = \begin{cases} 
\lambda U^{[1]}, & \text{if } ||\phi(X_i; W) - C||^2_F \leq 0 \\
\delta_{u_1}^{[1]} X^*_T, & \text{if } ||\phi(X_i; W) - C||^2_F > 0
\end{cases}$$  (5.54)

$$\partial_{V^{[1]}} (\mathcal{L}_{s_1}) = \begin{cases} 
\lambda V^{[1]}, & \text{if } ||\phi(X_i; W) - C||^2_F \leq 0 \\
\delta_{v_1}^{[1]} X, & \text{if } ||\phi(X_i; W) - C||^2_F > 0
\end{cases}$$  (5.55)

The SG uses the following iterative updates for matrix neural network layers $\ell = \{1, 2\}$:

$$U^{[\ell]}_{(t+1)} = U^{[\ell]}_{(t)} - \eta \partial_{U^{[\ell]}} (\mathcal{L}_{s_1})$$

$$V^{[\ell]}_{(t+1)} = V^{[\ell]}_{(t)} - \eta \partial_{V^{[\ell]}} (\mathcal{L}_{s_1})$$
where $\eta$ is defined as the learning rate for the network weights.

The exclusion of the radius term in this simplified methodology necessitates a new approach for outlier detection. As an alternative to the radius term, a threshold $h$ obtained using bootstrap simulation can be employed for the detection of outliers. A given test point $Z \in \mathcal{X}$ is in the target class if the squared Frobenius norm of the point $Z$ to the center of the hypersphere is less than a threshold $h$. In other words, the test point $Z$ will be in the target class if the following condition is satisfied:

$$S_1(Z) = \|\phi(Z; U^*, V^*) - C\|_F^2 \leq h \quad (5.56)$$

where $U^*$, $V^*$ are the network parameters of a trained model, and $S_1(Z)$ denotes the anomaly score for a given test point $Z$.

The SG OC-DSMDD training procedure is summarized in Algorithm 10.

---

**Algorithm 10 OC-DSMDD SG Algorithm**

*Input*: Training dataset $D = \{X_1, X_2, \ldots, X_N\} \subseteq \mathcal{X}$ with $X_i \in \mathbb{R}^{n \times p}$ and hyperparameter $\lambda$.

*Output*: Network weights $U^*$ and $V^*$ of a trained model.

1: **Initialize:**
   - Network weights $U$, $V$, and hypersphere center $C$ as the average of a small batch of training data after an initial forward pass through the network $\phi(\cdot; U, V)$.

2: **for** each epoch **do**

3: Randomly sample a training observation $X \in \mathcal{X}$.

4: Calculate $\phi(X; U, V)$ by forward propagation through OC-DSMDD model.

5: Calculate $L_s$ using Equation (5.47).

6: Update the network weights using the Adam algorithm.

7: **end for**

8: **return** $U^*$, $V^*$

---
Simulation Study

Performance Evaluation

We evaluate the performance of the proposed methods with other existing methods using an evaluation scheme based on the probability of a Type II error. It should be noted that within the DSMDD framework, the radius $R^2$ may be utilized for outlier detection. However, for users prioritizing the minimization of Type II error rates, the adoption of a threshold $h$ calibrated against the probability of a Type I error presents a more suitable alternative.

In this context, we are interested in two distinct components. The first is a distance measure that quantifies the spatial separation between a new observation $Z$ to the target class. The second is a discernment threshold, denoted by $h$, intended to serve as a comparative benchmark against the aforementioned distance metric.

A previously unseen observation will be classified as a target if the distance measure is smaller than a threshold $h$. In this framework, the absence of observations originating from the outlier class during the training procedure prevents the assessment of the Type II error rate. Therefore, only the target class observations that are erroneously categorized as outliers (Type I error) can be controlled. In order to evaluate the performance of DSMDD, we can use the probability of a Type I error, denoted by $\alpha$, and the probability of a Type II error, denoted by $\beta$.

Consider the availability of a training set, consisting of $N$ matrices $X \in \mathbb{R}^{n \times p}$ assumed to originate from the target class. For each newly encountered matrix $Z$, the distance statistic $d$ is computed as

$$d = ||\phi(Z; U^*, V^*) - C||_F^2$$  \hspace{1cm} (5.57)

where $U^*$ and $V^*$ are the network weights of a trained model. The new matrix $Z$ is classified
as an outlier if the condition \( d > h \) is satisfied, where the threshold \( h \) is chosen to achieve a specified Type I error rate \( \alpha \). Values of \( h \) are obtained by using the bootstrap simulation technique summarized in Algorithm 11.

**Algorithm 11 DSMDD Bootstrap Algorithm for Threshold \( h \)**

**Input:** Training dataset \( X = \{X_1, X_2, \ldots, X_N\} \subseteq \mathcal{X} \) with \( X_i \in \mathbb{R}^{n \times p} \), \( B \) bootstrap samples, and network weights \( U^*, V^* \) of a trained model.

**Output:** A threshold \( h \).

1. for \( X_i \in \mathcal{X} \) do
2. \hspace{1em} find \( d_i = ||\phi(X_i; U^*, V^*) - C||^2_F \)
3. end for
4. for \( b \leftarrow 1, B \) do
5. \hspace{1em} (i) Draw a bootstrap sample of size \( N \) from the set of \( N \) \( d \) statistics.
6. \hspace{1em} (ii) If \( \alpha \) is the desired Type I error, determine the \( 100 \times (1 - \alpha)^{th} \) percentile value.
7. end for
8. Calculate the threshold \( h \) by taking an average of \( B \ 100 \times (1 - \alpha)^{th} \) percentile values.
9. return \( h \)
Performance Settings

To assess the performance of DSMDD, we evaluate the probability of a Type II error under two different types of scenarios of distribution shift. In the first scenario, we highlight the applicability of this method to effectively handle both square and non-square matrices by considering a data distribution shift in the mean matrix using data that is generated from the matrix-variate normal distribution. The second scenario focuses on a specialized application tailored to square matrices. For these simulations, we consider a data distribution shift in the covariance matrix using data that is generated from the multivariate normal distribution when there are fewer observations than variables.

The network architecture under consideration for DSMDD is a matrix neural network with 2 hidden layers and an output layer. In the first scenario that deals with both square and non-square matrices, we use hidden layers of size $15 \times 15$ and an output layer of size $10 \times 10$ for dimensions $(3, 5)$ and $(5, 5)$. For dimensions $(10, 20)$ and $(20, 10)$, we use hidden layers of size $25 \times 25$ and an output layer of size $20 \times 20$. In the second scenario that deals with square covariance matrices, we use hidden layers of size $15 \times 15$ and an output layer of size $10 \times 10$ for the $p = 5$ dimension setting. For $p = 10$, we use hidden layers of size $25 \times 25$ and an output layer of size $20 \times 20$. Leaky ReLU activation functions with leakiness $\alpha = 0.1$ are used in the hidden layers, and linear activation functions are used in the output layer for all scenarios considered.

The hypersphere center $C$ is fixed as the average feature representation of a small batch of training data after performing an initial forward pass through the matrix neural network, and is not treated as a learnable parameter. The hypersphere radius is initialized by randomly choosing a value in the range $[0.1, 1]$. DSMDD is trained using stochastic subgradient (SG) methods and Adam optimizer, proposed by Kingma and Ba [28], to optimize the network weight matrices $\mathcal{U}$ and $\mathcal{V}$.
The radius can be updated using two different procedures. The first option is to treat the radius as a learnable parameter and update its value every epoch using the subgradient update. An alternative strategy is to find the value of the radius by taking the $1 - \nu$ percentile of the squared Frobenius norms from the feature representations of the training set to the center of the hypersphere. Using this second approach, an alternating minimization updating strategy can be employed where the network weights are updated for $k \in \mathbb{N}$ epochs while the radius is held fixed. Then, after every $k$–th epoch, the radius is updated using the network weight parameters $U$ and $V$ of the latest update. This alternating strategy then repeats and is carried out until convergence to a local minima. For the following simulations, we use this second approach to train the DSMDD model.

All models were implemented using Python 3.10.12, PyTorch-2.0.1, and CUDA 11.8. The experiments are carried out on a PC with an NVIDIA GeForce RTX 3080 GPU and 128 GB of RAM.
To assess the performance of DSMDD, we evaluate the probability of a Type II error following a data distribution shift in the mean matrix using matrix-variate normal data.

The target class is generated from a matrix-variate normal distribution $\mathcal{X} \sim \mathcal{N}_{n \times p} (\mathbf{M}, \Sigma \otimes \Psi)$ with mean matrix $\mathbf{M} \in \mathbb{R}^{n \times p}$ and covariance matrix $\Sigma \otimes \Psi$, where $\Sigma \in \mathbb{R}^{n \times n}$ and $\Psi \in \mathbb{R}^{p \times p}$. The mean matrix is set to zero $\mathbf{M} = 0_{n \times p}$ and identity covariance matrix $\Sigma_0 \otimes \Psi_0 = I_n \otimes I_p$.

We investigate four different dimension settings. The dimensions we consider are $(3 \times 5), (5 \times 5), (10 \times 20)$, and $(20 \times 10)$ where $(n \times p)$ are the dimensions of the matrices and $N = 50$ is the number of matrices originating from the target class.

After the model was trained on data originating solely from the target class, the evaluation of anomalous observations was considered by changing the zero mean matrix $\mathbf{M}_0 = 0_{n \times p}$ from the target set’s underlying distribution to a matrix $\mathbf{M}_1$ having $\delta$ in its first row and column component while the other elements remain unchanged.

$$
\mathbf{M}_1 = \begin{pmatrix}
\delta & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 0
\end{pmatrix}_{n \times p}
$$

If the data generating process has a distribution shift from $\mathcal{N} (\mathbf{M}_0, \Sigma_0 \otimes \Psi_0)$ to $\mathcal{N} (\mathbf{M}_1, \Sigma_0 \otimes \Psi_0)$, we are interested in the speed at which the algorithm can detect the change. Consequently, the performance following any step change in the mean can be modeled by changing a single component of the mean matrix as follows: set the target set’s mean matrix to $\mathbf{M}_0 = 0$ and the covariance matrix to $\Sigma_0 \otimes \Psi_0 = I_n \otimes I_p$. Thereafter, by adding a shift $\delta$ to the first row and column component of each mean matrix we can explore all possible step changes in the mean matrix.
Simulation setting: mean shift $n = 3, \ p = 5$

In this section we present simulation results for a distribution shift where $n = 3, p = 5$, and $N = 50$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.1 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.91</td>
<td>0.91</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.89</td>
<td>0.90</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.87</td>
<td>0.89</td>
<td>0.94</td>
</tr>
<tr>
<td>0.6</td>
<td>0.85</td>
<td>0.87</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.83</td>
<td>0.85</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.80</td>
<td>0.83</td>
<td>0.93</td>
</tr>
<tr>
<td>0.9</td>
<td>0.78</td>
<td>0.81</td>
<td>0.93</td>
</tr>
<tr>
<td>1.0</td>
<td>0.75</td>
<td>0.79</td>
<td>0.93</td>
</tr>
<tr>
<td>1.5</td>
<td>0.58</td>
<td>0.64</td>
<td>0.88</td>
</tr>
<tr>
<td>2.0</td>
<td>0.39</td>
<td>0.47</td>
<td>0.81</td>
</tr>
<tr>
<td>2.5</td>
<td>0.22</td>
<td>0.29</td>
<td>0.69</td>
</tr>
<tr>
<td>3.0</td>
<td>0.10</td>
<td>0.16</td>
<td>0.56</td>
</tr>
<tr>
<td>3.5</td>
<td>0.04</td>
<td>0.07</td>
<td>0.41</td>
</tr>
<tr>
<td>4.0</td>
<td>0.01</td>
<td>0.03</td>
<td>0.25</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.01</td>
<td>0.15</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Figure 5.1: DSMDD and OC-DSMDD simulation results for dimensions $(n \times p) = (3, 5)$ with matrix-variate normal data.

The DSMDD exhibits superior performance compared to OC-DSMDD in the range of $0.5 \leq \delta \leq 4.0$. Moreover, both DSMDD and OC-DSMDD show better performance than SMDD.
Simulation setting: mean shift $n = 5, \ p = 5$

In this section we present simulation results for a distribution shift where $n = 5, p = 5$, and $N = 50$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.2 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.90</td>
<td>0.90</td>
<td>0.94</td>
</tr>
<tr>
<td>0.5</td>
<td>0.88</td>
<td>0.89</td>
<td>0.94</td>
</tr>
<tr>
<td>0.6</td>
<td>0.86</td>
<td>0.87</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.84</td>
<td>0.85</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.81</td>
<td>0.83</td>
<td>0.93</td>
</tr>
<tr>
<td>0.9</td>
<td>0.79</td>
<td>0.80</td>
<td>0.93</td>
</tr>
<tr>
<td>1.0</td>
<td>0.76</td>
<td>0.78</td>
<td>0.92</td>
</tr>
<tr>
<td>1.5</td>
<td>0.59</td>
<td>0.64</td>
<td>0.90</td>
</tr>
<tr>
<td>2.0</td>
<td>0.40</td>
<td>0.45</td>
<td>0.84</td>
</tr>
<tr>
<td>2.5</td>
<td>0.24</td>
<td>0.28</td>
<td>0.76</td>
</tr>
<tr>
<td>3.0</td>
<td>0.11</td>
<td>0.15</td>
<td>0.66</td>
</tr>
<tr>
<td>3.5</td>
<td>0.05</td>
<td>0.06</td>
<td>0.51</td>
</tr>
<tr>
<td>4.0</td>
<td>0.01</td>
<td>0.02</td>
<td>0.37</td>
</tr>
<tr>
<td>4.5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.24</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Figure 5.2: DSMDD and OC-DSMDD simulation results for dimensions $(n \times p) = (5, 5)$ with matrix-variate normal data.

The DSMDD achieves marginally better performance than OC-SMDD in the range of $0.8 \leq \delta \leq 3.0$, while both DSMDD and OC-DSMDD demonstrate greater performance than SMDD.
Simulation setting: mean shift \( n = 10, \ p = 20 \)

In this section we present simulation results for a distribution shift where \( n = 10, \ p = 20, \) and \( N = 50 \). For this scenario, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05 \). A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.3 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.95</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.93</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.93</td>
<td>0.91</td>
<td>0.95</td>
</tr>
<tr>
<td>0.7</td>
<td>0.92</td>
<td>0.90</td>
<td>0.95</td>
</tr>
<tr>
<td>0.8</td>
<td>0.91</td>
<td>0.88</td>
<td>0.95</td>
</tr>
<tr>
<td>0.9</td>
<td>0.90</td>
<td>0.87</td>
<td>0.94</td>
</tr>
<tr>
<td>1.0</td>
<td>0.88</td>
<td>0.86</td>
<td>0.94</td>
</tr>
<tr>
<td>1.5</td>
<td>0.81</td>
<td>0.77</td>
<td>0.94</td>
</tr>
<tr>
<td>2.0</td>
<td>0.70</td>
<td>0.67</td>
<td>0.92</td>
</tr>
<tr>
<td>2.5</td>
<td>0.55</td>
<td>0.55</td>
<td>0.91</td>
</tr>
<tr>
<td>3.0</td>
<td>0.41</td>
<td>0.42</td>
<td>0.89</td>
</tr>
<tr>
<td>3.5</td>
<td>0.26</td>
<td>0.30</td>
<td>0.85</td>
</tr>
<tr>
<td>4.0</td>
<td>0.16</td>
<td>0.19</td>
<td>0.81</td>
</tr>
<tr>
<td>4.5</td>
<td>0.08</td>
<td>0.11</td>
<td>0.75</td>
</tr>
<tr>
<td>5.0</td>
<td>0.04</td>
<td>0.05</td>
<td>0.69</td>
</tr>
</tbody>
</table>

Figure 5.3: DSMDD and OC-DSMDD simulation results for dimensions \((n \times p) = (10, 20)\) with matrix-variate normal data.

The performance of OC-DSMDD marginally exceeds that of DSMDD in the range of \( 0.6 \leq \delta \leq 2.0 \). However, DSMDD marginally outperforms OC-DSMDD in the range of \( 3.5 \leq \delta \leq 4.5 \). Moreover, the performance of both DSMDD and OC-DSMDD is notably better than SMDD.
Simulation setting: mean shift $n = 20$, $p = 10$

In this section we present simulation results for a distribution shift where $n = 20$, $p = 10$, and $N = 50$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.4 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.95</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.94</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td>0.4</td>
<td>0.91</td>
<td>0.91</td>
<td>0.95</td>
</tr>
<tr>
<td>0.5</td>
<td>0.90</td>
<td>0.90</td>
<td>0.95</td>
</tr>
<tr>
<td>0.6</td>
<td>0.88</td>
<td>0.89</td>
<td>0.94</td>
</tr>
<tr>
<td>0.7</td>
<td>0.86</td>
<td>0.88</td>
<td>0.94</td>
</tr>
<tr>
<td>0.8</td>
<td>0.85</td>
<td>0.87</td>
<td>0.94</td>
</tr>
<tr>
<td>0.9</td>
<td>0.82</td>
<td>0.85</td>
<td>0.94</td>
</tr>
<tr>
<td>1.0</td>
<td>0.80</td>
<td>0.83</td>
<td>0.94</td>
</tr>
<tr>
<td>1.5</td>
<td>0.65</td>
<td>0.74</td>
<td>0.93</td>
</tr>
<tr>
<td>2.0</td>
<td>0.47</td>
<td>0.61</td>
<td>0.92</td>
</tr>
<tr>
<td>2.5</td>
<td>0.31</td>
<td>0.46</td>
<td>0.90</td>
</tr>
<tr>
<td>3.0</td>
<td>0.17</td>
<td>0.33</td>
<td>0.88</td>
</tr>
<tr>
<td>3.5</td>
<td>0.08</td>
<td>0.20</td>
<td>0.84</td>
</tr>
<tr>
<td>4.0</td>
<td>0.03</td>
<td>0.11</td>
<td>0.80</td>
</tr>
<tr>
<td>4.5</td>
<td>0.01</td>
<td>0.05</td>
<td>0.74</td>
</tr>
<tr>
<td>5.0</td>
<td>0.00</td>
<td>0.02</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Figure 5.4: DSMDD and OC-DSMDD simulation results for dimensions $(n \times p) = (20, 10)$ with matrix-variate normal data.

The performance of DSMDD exceeds both OC-DSMDD and SMDD, and OC-DSMDD exhibits better performance compared to SMDD.
To assess the performance of DSMDD in a specialized application for square matrices, we evaluate the probability of a Type II error following a data distribution shift in the covariance matrix using data that is generated from the multivariate normal distribution. Specifically, we consider a one-class classification problem in the context of covariance matrices where the sample data has fewer observations than variables. Given the singular nature of these matrices, classical statistical methodologies are not applicable, thereby emphasizing certain advantageous properties of DSMDD that make it particularly well-suited to handle one-class classification problems for rank deficient matrices.

To construct the target class, we first generate \( N \) subgroups, where the size of each subgroup is denoted by \( m \), from a \( p \)-variate multivariate normal distribution with the mean vector set to zero \( \mu_0 = 0 \) and identity covariance matrix \( \Sigma_0 = I_p \). For each subgroup, the empirical covariance matrix is computed resulting in a total of \( N \) covariance matrices that collectively represent the target class.

Suppose we have a training set \( X_i, i = 1, 2, ..., N \), which are \((p \times p)\) covariance matrices that are rank deficient and not positive definite assumed to originate from the target class. We investigate two different dimension settings, introducing a variety of changes to the scale of the underlying distribution under each setting. First, we consider dimensions \( p = 5 \) with a subgroup size of \( m = 4 \), resulting in \( N = 50 \) covariance matrices with dimensions \((5 \times 5)\). In the subsequent scenario, we consider \( p = 10 \) dimensions and a subgroup size of \( m = 7 \), yielding \( N = 50 \) covariance matrices with dimensions \((10 \times 10)\). For each dimension setting, we consider three different scenarios of distribution shift in scale.
In the first scenario ("variance shift"), we changed the covariance matrix from $\Sigma_0 = I_{p \times p}$ to a matrix $\Sigma_1$ having $1 + \delta$ in the $(1, 1)$ position with the other elements unchanged, and left the mean vector at $0$.

$$\Sigma_1 = \begin{pmatrix}
1 + \delta & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{pmatrix}_{p \times p}$$

In the second scenario ("correlation shift"), we changed the covariance matrix from $\Sigma_0 = I_{p \times p}$ to a matrix $\Sigma_2$ having a correlation $\delta$ in the $(1, 2)$ and $(2, 1)$ positions with the other elements unchanged, and left the mean vector at $0$.

$$\Sigma_2 = \begin{pmatrix}
1 & \delta & 0 & \cdots & 0 \\
\delta & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{pmatrix}_{p \times p}$$

In the final scenario ("simultaneous variance and correlation shifts"), the covariance matrix $\Sigma_0 = I_{p \times p}$ was changed to a matrix $\Sigma_3$ by putting $\delta$ in the $(1, 2)$ and $(2, 1)$ positions, as well as $1 + \delta$ in the $(1, 1)$ position while the other elements remained unchanged, and left the mean vector at $0$.

$$\Sigma_3 = \begin{pmatrix}
1 + \delta & \delta & 0 & \cdots & 0 \\
\delta & 1 & 0 & \cdots & 0 \\
0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 1
\end{pmatrix}_{p \times p}$$
Simulation setting: variance shift \( p = 5, \ m = 4 \)

In this section we present simulation results for a distribution shift in scale where \( p = 5, \ m = 4, \) and \( N = 50. \) For this scenario, the Type II error is computed for \( \delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\} \), and we generate 15,000 independent and identically distributed observations at each value of \( \delta \) where the probability of a Type I error is set to \( \alpha = 0.05. \) A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.5 shown below.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.93</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.91</td>
<td>0.91</td>
<td>0.92</td>
</tr>
<tr>
<td>0.3</td>
<td>0.89</td>
<td>0.89</td>
<td>0.91</td>
</tr>
<tr>
<td>0.4</td>
<td>0.87</td>
<td>0.87</td>
<td>0.90</td>
</tr>
<tr>
<td>0.5</td>
<td>0.86</td>
<td>0.84</td>
<td>0.89</td>
</tr>
<tr>
<td>0.6</td>
<td>0.83</td>
<td>0.82</td>
<td>0.87</td>
</tr>
<tr>
<td>0.7</td>
<td>0.81</td>
<td>0.80</td>
<td>0.86</td>
</tr>
<tr>
<td>0.8</td>
<td>0.79</td>
<td>0.77</td>
<td>0.85</td>
</tr>
<tr>
<td>0.9</td>
<td>0.76</td>
<td>0.75</td>
<td>0.83</td>
</tr>
<tr>
<td>1.0</td>
<td>0.74</td>
<td>0.73</td>
<td>0.81</td>
</tr>
<tr>
<td>1.5</td>
<td>0.65</td>
<td>0.61</td>
<td>0.73</td>
</tr>
<tr>
<td>2.0</td>
<td>0.55</td>
<td>0.51</td>
<td>0.65</td>
</tr>
<tr>
<td>2.5</td>
<td>0.48</td>
<td>0.44</td>
<td>0.56</td>
</tr>
<tr>
<td>3.0</td>
<td>0.41</td>
<td>0.38</td>
<td>0.51</td>
</tr>
<tr>
<td>3.5</td>
<td>0.36</td>
<td>0.33</td>
<td>0.46</td>
</tr>
<tr>
<td>4.0</td>
<td>0.33</td>
<td>0.29</td>
<td>0.40</td>
</tr>
<tr>
<td>4.5</td>
<td>0.28</td>
<td>0.25</td>
<td>0.36</td>
</tr>
<tr>
<td>5.0</td>
<td>0.24</td>
<td>0.23</td>
<td>0.32</td>
</tr>
</tbody>
</table>

Figure 5.5: DSMDD and OC-DSMDD variance shift simulation results for \( p = 5, \ m = 4. \)

The DSMDD attains a level of performance that surpasses OC-DSMDD in the range \( 1.5 \leq \delta \leq 4.5, \) with OC-DSMDD additionally offering better performance compared to SMDD.
Simulation setting: correlation shift $p = 5, \ m = 4$

In this section we present simulation results for a distribution shift in scale where $p = 5, \ m = 4$, and $N = 50$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.6 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.3</td>
<td>0.93</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>0.4</td>
<td>0.91</td>
<td>0.90</td>
<td>0.94</td>
</tr>
<tr>
<td>0.5</td>
<td>0.90</td>
<td>0.89</td>
<td>0.93</td>
</tr>
<tr>
<td>0.6</td>
<td>0.89</td>
<td>0.88</td>
<td>0.92</td>
</tr>
<tr>
<td>0.7</td>
<td>0.87</td>
<td>0.85</td>
<td>0.92</td>
</tr>
<tr>
<td>0.8</td>
<td>0.86</td>
<td>0.84</td>
<td>0.90</td>
</tr>
<tr>
<td>0.9</td>
<td>0.85</td>
<td>0.83</td>
<td>0.89</td>
</tr>
<tr>
<td>1.0</td>
<td>0.83</td>
<td>0.81</td>
<td>0.88</td>
</tr>
<tr>
<td>1.5</td>
<td>0.71</td>
<td>0.68</td>
<td>0.79</td>
</tr>
<tr>
<td>2.0</td>
<td>0.57</td>
<td>0.55</td>
<td>0.67</td>
</tr>
<tr>
<td>2.5</td>
<td>0.43</td>
<td>0.43</td>
<td>0.54</td>
</tr>
<tr>
<td>3.0</td>
<td>0.33</td>
<td>0.33</td>
<td>0.42</td>
</tr>
<tr>
<td>3.5</td>
<td>0.24</td>
<td>0.25</td>
<td>0.32</td>
</tr>
<tr>
<td>4.0</td>
<td>0.19</td>
<td>0.19</td>
<td>0.25</td>
</tr>
<tr>
<td>4.5</td>
<td>0.14</td>
<td>0.15</td>
<td>0.20</td>
</tr>
<tr>
<td>5.0</td>
<td>0.11</td>
<td>0.11</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Figure 5.6: DSMDD and OC-DSMDD correlation shift simulation results for $p = 5, \ m = 4$.

The DSMDD and OC-DSMDD exhibit comparable performance. However, both DSMDD and OC-DSMDD significantly outperform SMDD.
Simulation setting: simultaneous variance and correlation Shifts $p = 5$, $m = 4$

In this section we present simulation results for a distribution shift in scale where $p = 5$, $m = 4$, and $N = 50$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.7 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.93</td>
<td>0.92</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.90</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.87</td>
<td>0.88</td>
<td>0.92</td>
</tr>
<tr>
<td>0.4</td>
<td>0.85</td>
<td>0.85</td>
<td>0.90</td>
</tr>
<tr>
<td>0.5</td>
<td>0.81</td>
<td>0.82</td>
<td>0.89</td>
</tr>
<tr>
<td>0.6</td>
<td>0.77</td>
<td>0.80</td>
<td>0.86</td>
</tr>
<tr>
<td>0.7</td>
<td>0.74</td>
<td>0.77</td>
<td>0.85</td>
</tr>
<tr>
<td>0.8</td>
<td>0.71</td>
<td>0.74</td>
<td>0.82</td>
</tr>
<tr>
<td>0.9</td>
<td>0.68</td>
<td>0.72</td>
<td>0.80</td>
</tr>
<tr>
<td>1.0</td>
<td>0.65</td>
<td>0.68</td>
<td>0.77</td>
</tr>
<tr>
<td>1.5</td>
<td>0.51</td>
<td>0.56</td>
<td>0.66</td>
</tr>
<tr>
<td>2.0</td>
<td>0.41</td>
<td>0.45</td>
<td>0.56</td>
</tr>
<tr>
<td>2.5</td>
<td>0.33</td>
<td>0.35</td>
<td>0.46</td>
</tr>
<tr>
<td>3.0</td>
<td>0.26</td>
<td>0.28</td>
<td>0.36</td>
</tr>
<tr>
<td>3.5</td>
<td>0.21</td>
<td>0.23</td>
<td>0.29</td>
</tr>
<tr>
<td>4.0</td>
<td>0.17</td>
<td>0.19</td>
<td>0.23</td>
</tr>
<tr>
<td>4.5</td>
<td>0.14</td>
<td>0.15</td>
<td>0.18</td>
</tr>
<tr>
<td>5.0</td>
<td>0.12</td>
<td>0.12</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Figure 5.7: DSMDD and OC-DSMDD variance and correlation shift simulation results for $p = 5$, $m = 4$.

The performance of DSMDD marginally exceeds that of OC-DSMDD, while both DSMDD and OC-DSMDD notably outperform SMDD.
Simulation setting: variance shift $p = 10$, $m = 7$

In this section we present simulation results for a distribution shift in scale where $p = 10$, $m = 7$, and $N = 50$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.8 shown below.

<table>
<thead>
<tr>
<th>Type II Error Rate</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.93</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.92</td>
<td>0.92</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.90</td>
<td>0.90</td>
<td>0.92</td>
</tr>
<tr>
<td>0.4</td>
<td>0.88</td>
<td>0.88</td>
<td>0.91</td>
</tr>
<tr>
<td>0.5</td>
<td>0.86</td>
<td>0.86</td>
<td>0.90</td>
</tr>
<tr>
<td>0.6</td>
<td>0.82</td>
<td>0.84</td>
<td>0.88</td>
</tr>
<tr>
<td>0.7</td>
<td>0.80</td>
<td>0.82</td>
<td>0.87</td>
</tr>
<tr>
<td>0.8</td>
<td>0.77</td>
<td>0.78</td>
<td>0.85</td>
</tr>
<tr>
<td>0.9</td>
<td>0.75</td>
<td>0.76</td>
<td>0.83</td>
</tr>
<tr>
<td>1.0</td>
<td>0.72</td>
<td>0.74</td>
<td>0.82</td>
</tr>
<tr>
<td>1.5</td>
<td>0.59</td>
<td>0.62</td>
<td>0.72</td>
</tr>
<tr>
<td>2.0</td>
<td>0.48</td>
<td>0.50</td>
<td>0.62</td>
</tr>
<tr>
<td>2.5</td>
<td>0.38</td>
<td>0.40</td>
<td>0.53</td>
</tr>
<tr>
<td>3.0</td>
<td>0.30</td>
<td>0.32</td>
<td>0.44</td>
</tr>
<tr>
<td>3.5</td>
<td>0.24</td>
<td>0.26</td>
<td>0.37</td>
</tr>
<tr>
<td>4.0</td>
<td>0.20</td>
<td>0.21</td>
<td>0.31</td>
</tr>
<tr>
<td>4.5</td>
<td>0.15</td>
<td>0.17</td>
<td>0.26</td>
</tr>
<tr>
<td>5.0</td>
<td>0.13</td>
<td>0.15</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Figure 5.8: DSMDD and OC-DSMDD variance shift simulation results for $p = 10$, $m = 7$.

The DSMDD and OC-DSMDD exhibit comparable performance. However, SMDD achieves notably lower performance compared to both DSMDD and SMDD.
Simulation setting: correlation shift $p = 10, \; m = 7$

In this section we present simulation results for a distribution shift in scale where $p = 10, \; m = 7,$ and $N = 50.$ For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, \ldots, 0.9, 1.0, 1.5, 2.0, \ldots, 5.0\},$ and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05.$ A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.9 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.94</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.91</td>
<td>0.93</td>
</tr>
<tr>
<td>0.4</td>
<td>0.91</td>
<td>0.90</td>
<td>0.92</td>
</tr>
<tr>
<td>0.5</td>
<td>0.89</td>
<td>0.89</td>
<td>0.91</td>
</tr>
<tr>
<td>0.6</td>
<td>0.88</td>
<td>0.88</td>
<td>0.90</td>
</tr>
<tr>
<td>0.7</td>
<td>0.86</td>
<td>0.87</td>
<td>0.89</td>
</tr>
<tr>
<td>0.8</td>
<td>0.85</td>
<td>0.85</td>
<td>0.88</td>
</tr>
<tr>
<td>0.9</td>
<td>0.73</td>
<td>0.71</td>
<td>0.77</td>
</tr>
<tr>
<td>1.0</td>
<td>0.58</td>
<td>0.54</td>
<td>0.62</td>
</tr>
<tr>
<td>1.5</td>
<td>0.41</td>
<td>0.38</td>
<td>0.47</td>
</tr>
<tr>
<td>2.0</td>
<td>0.27</td>
<td>0.26</td>
<td>0.33</td>
</tr>
<tr>
<td>2.5</td>
<td>0.17</td>
<td>0.18</td>
<td>0.23</td>
</tr>
<tr>
<td>3.0</td>
<td>0.12</td>
<td>0.11</td>
<td>0.15</td>
</tr>
<tr>
<td>3.5</td>
<td>0.07</td>
<td>0.07</td>
<td>0.10</td>
</tr>
<tr>
<td>4.0</td>
<td>0.05</td>
<td>0.05</td>
<td>0.07</td>
</tr>
<tr>
<td>4.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.9: DSMDD and OC-DSMDD correlation shift simulation results for $p = 10, \; m = 7.$

The OC-DSMDD and DSMDD achieve comparable performance, however the OC-DSMDD marginally outperforms DSMDD in the range $1.5 \leq \delta \leq 2.5.$ Moreover, both OC-DSMDD and DSMDD outperforms SMDD.
**Simulation setting: simultaneous variance and correlation shifts** $p = 10, \ m = 7$

In this section we present simulation results for a distribution shift in scale where $p = 10, \ m = 7$, and $N = 50$. For this scenario, the Type II error is computed for $\delta \in \{0.0, 0.1, 0.2, ..., 0.9, 1.0, 1.5, 2.0, ..., 5.0\}$, and we generate 15,000 independent and identically distributed observations at each value of $\delta$ where the probability of a Type I error is set to $\alpha = 0.05$. A table of the Type II error rates alongside a corresponding plot that visualizes these rates is provided in Figure 5.10 shown below.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>DSMDD</th>
<th>OC-DSMDD</th>
<th>SMDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.95</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>0.1</td>
<td>0.94</td>
<td>0.93</td>
<td>0.94</td>
</tr>
<tr>
<td>0.2</td>
<td>0.93</td>
<td>0.92</td>
<td>0.93</td>
</tr>
<tr>
<td>0.3</td>
<td>0.92</td>
<td>0.90</td>
<td>0.91</td>
</tr>
<tr>
<td>0.4</td>
<td>0.90</td>
<td>0.87</td>
<td>0.90</td>
</tr>
<tr>
<td>0.5</td>
<td>0.86</td>
<td>0.85</td>
<td>0.88</td>
</tr>
<tr>
<td>0.6</td>
<td>0.84</td>
<td>0.82</td>
<td>0.85</td>
</tr>
<tr>
<td>0.7</td>
<td>0.81</td>
<td>0.78</td>
<td>0.83</td>
</tr>
<tr>
<td>0.8</td>
<td>0.78</td>
<td>0.75</td>
<td>0.80</td>
</tr>
<tr>
<td>0.9</td>
<td>0.75</td>
<td>0.71</td>
<td>0.77</td>
</tr>
<tr>
<td>1.0</td>
<td>0.72</td>
<td>0.68</td>
<td>0.75</td>
</tr>
<tr>
<td>1.5</td>
<td>0.57</td>
<td>0.51</td>
<td>0.60</td>
</tr>
<tr>
<td>2.0</td>
<td>0.43</td>
<td>0.37</td>
<td>0.46</td>
</tr>
<tr>
<td>2.5</td>
<td>0.32</td>
<td>0.27</td>
<td>0.34</td>
</tr>
<tr>
<td>3.0</td>
<td>0.23</td>
<td>0.20</td>
<td>0.24</td>
</tr>
<tr>
<td>3.5</td>
<td>0.17</td>
<td>0.14</td>
<td>0.17</td>
</tr>
<tr>
<td>4.0</td>
<td>0.13</td>
<td>0.10</td>
<td>0.12</td>
</tr>
<tr>
<td>4.5</td>
<td>0.09</td>
<td>0.07</td>
<td>0.09</td>
</tr>
<tr>
<td>5.0</td>
<td>0.07</td>
<td>0.05</td>
<td>0.06</td>
</tr>
</tbody>
</table>

Figure 5.10: DSMDD and OC-DSMDD variance and correlation shift simulation results for $p = 10, \ m = 7$.

The DSMDD and SMDD have comparable performance results. However, OC-DSMDD exhibits better performance in the range $0.6 \leq \delta \leq 5.0$. 

161
Summary of Simulation Results

In this section, we evaluated the performance of DSMDD and OC-DSMDD against SMDD across various types of scenarios involving square and non-square matrices.

In the first set of simulations, we considered several scenarios where a distribution shift in location was introduced using matrix-variate normal data for both square and non-square matrices. In all cases, both DSMDD and OC-DSMDD attained notably superior performance compared to SMDD. In the \((n \times p) = (20, 10)\) scenario, the performance of OC-DSMDD was slightly worse than DSMDD. However, in all other scenarios, these two methods had very similar performance results. Therefore, under the settings considered, the differences in performance between DSMDD and OC-DSMDD may fluctuate depending on the dimensional settings.

As a special application to square matrices, the subsequent set of simulations focused on a distribution shift in scale using covariance matrices generated from multivariate normal data when there were fewer observations than variables. In the \(p = 5\) setting, DSMDD and OC-DSMDD have similar performance results and both outperform SMDD by a considerable margin. The notable difference in performance between the kernel-based SMDD model compared to the deep learning models was also encountered in the first scenario related to variance shift in the \(p = 10\) setting. However, in the subsequent two scenarios, these differences in model performance were less pronounced. Consequently, under the settings considered, in lower dimensional settings the deep learning methods outperform the kernel-based method. However, this advantage may lessen in higher dimensions, influenced by the particular type of shift in scale that is encountered.
Theoretical Properties

A collection of propositions presented by Ruff et al. [48] highlighted several theoretical properties of DSVDD. The main focus of these properties relates to averting the phenomenon of hypersphere collapse, which occurs when the model learns degenerate or uninformative solutions. To adapt these vector-based properties of DSVDD to DSMDD, we derive similar matrix-based properties for DSMDD tailored for matrix space. In the following proofs, we demonstrate how using an improperly formulated matrix neural network that includes bias, or by treating the hypersphere center as a learnable parameter, may lead to a (matrix) hypersphere collapse. For the following propositions, let \( J(R, U, V) \) denote the objective function (5.2) for DSMDD.

**Proposition 5.0.4 (All-zero-weights solution)** Define \( \{U_0, V_0\} \) as the set comprising exclusively zero-valued network weights, such that for every set of weight matrices \( \{U^{[\ell]}, V^{[\ell]}\} \in \{U_0, V_0\} \) of layer \( \ell \in \{1, 2, ..., L\} \), respectively, it holds that \( U^{[\ell]}_0 = 0 \) and \( V^{[\ell]}_0 = 0 \). Under this configuration, the network’s functional mapping becomes invariant, yielding the same output irrespective of the variations in the input. Restated, this implies that for any \( X, \bar{X} \in \mathcal{X} \), the matrix neural network feature representations can be defined as \( \phi(X; U_0, V_0) = \phi(\bar{X}; U_0, V_0) =: C_0 \in \mathcal{F} \). Consequently, in the event that the condition \( C = C_0 \) is satisfied, then the optimal solution for DSMDD is given by \( \{U^*, V^*\} = \{U_0, V_0\} \) and \( R^* = 0 \).

**Proof.** Across all scenarios characterized by \( (R, U, V) \), we can assume the inequality \( J(R, U, V) \geq 0 \) is consistently satisfied. Considering that the output from the network \( \phi(X; U_0, V_0) \), where all network weights are zero, is invariant for every input \( X \in \mathcal{X} \), then all errors in the empirical summation of the objective are zero resulting in \( J(R^*, U^*, V^*) = 0 \). This is due to the zero-valued network parameters leading to a zero mapping for any input, where the hypersphere center is defined as \( C = \phi(X; U_0, V_0) \). Consequently, the optimal solution is \( (R^*, U^*, V^*) = (0, U_0, V_0) \).
Proposition 5.0.5 (Bias terms) Define $C \in F$ as the centroid of a predefined (matrix) hypersphere. Within the matrix neural network $\phi(\cdot; U, V) : \mathcal{X} \mapsto F$, if a hidden layer incorporates a bias term, then for every $X \in \mathcal{X}$ there necessarily arises an optimal solution set $(R^*, U^*, V^*)$ for the objective $L_s(U, V, R)$, where $R^*$ is equal to zero and the network output is a constant mapping $\phi(X; U^*, V^*) = C^*$.

Proof. Consider a matrix neural network with $L \in \mathbb{N}$ layers. The network consists of $L-1$ hidden layers and a final output layer, and the architecture is characterized by a set of weight matrices $\{U, V\} = \{U[1], V[1], \ldots, U[L], V[L]\}$ and bias matrices $B = \{B[1], B[2], \ldots, B[L]\}$, corresponding to the weights $U[\ell], V[\ell]$ and biases $B[\ell]$ of layer $\ell = \{1, 2, ..., L\}$, respectively. For any $X \in \mathcal{X}$, the resulting network output of layer $\ell$ can be expressed as follows:

$$A[\ell] = g_\ell \left(Z[\ell] = g_{\ell-1} \left(U[\ell] A[\ell-1] V[\ell]^{\top} + B[\ell]\right)\right)$$

where $A[\ell-1]$ is the output of the previous $\ell - 1$ hidden layer, and is obtained after applying $g_{\ell-1}(\cdot)$ in an element-wise manner to the weighted input $Z[\ell-1]$, and $g_\ell(\cdot)$ is the activation function of layer $\ell$ that is applied element-wise to the weighted input $Z[\ell]$. If $U[\ell] = 0$ and $V[\ell] = 0$, then $A[\ell] = g_\ell(B[\ell])$. In other words, every input $X \in \mathcal{X}$ leads to a constant output from layer $\ell$.

Let the hypersphere center $C$ be a function of the bias $B[\ell]$ as well as the ensuing parameters $(U[\ell+1], V[\ell+1], \ldots, U[L], V[L])$ such that it is in the image of the matrix neural network. Ignoring any contribution to the loss from the weight regularization component, an optimal solution is obtained by setting $R^* = 0$ when the network parameters are chosen in this manner.
Proposition 5.0.6 \((\nu\text{-property})\) The hyperparameter \(\nu \in (0, 1]\) in the DSMDD objective (5.2) is

i. an upper bound on the fraction of training observations classified as outliers.

ii. a lower bound on the fraction of training observations falling either outside or directly on the optimal boundary.

Proof. Ad (i). Let the squared Frobenius distance (in matrix space) from each network representation to the hypersphere center be defined as

\[ d_i = \|\phi(X_i; \mathcal{U}, \mathcal{V}) - C\|^2_F \text{ for } i = 1, 2, \ldots, N. \]

Without loss of generality assume \(d_1 \geq d_2 \geq \ldots \geq d_N\) such that \(d_N\) corresponds to the distance of the farthest training observation from the hypersphere center. Then, the number of outliers can be expressed as the cardinality of the set of observations with distance larger than the squared radius. In other words, the number of outliers is given by

\[ N_{\text{outlier}} = |\{i : d_i > R^2\}|. \]

Therefore, we can express the DSMDD objective (in radius \(R\)) as follows:

\[ J(R) = R^2 - \frac{N_{\text{outlier}}}{\nu N} R^2 = \left(1 - \frac{N_{\text{outlier}}}{\nu N}\right) R^2. \]

As DSMDD aims to find the smallest radius \(R\), it follows that the expression \(1 - \frac{N_{\text{outlier}}}{\nu N}\) maintains a positive value, thus leading to the implication \(\nu N\) should be at least as large as \(N_{\text{outlier}}\). Therefore, based on this criterion, it implies that the upper limit of outliers to be disregarded should not exceed \(\nu N\).

Ad (ii). Throughout the training process, the radius \(R\) is progressively minimized as long as the condition \(N_{\text{outlier}} \leq \nu N\) holds, which inherently results in a gradual increase in \(N_{\text{outlier}}\). Should the condition \(R^2 > d_N\) be satisfied, \(N_{\text{outlier}}\) attains its minimal value of 0, indicating that the boundary encompasses the entire training set. Alternatively, if \(N_{\text{outlier}}\) assumes its maximum value of \(\nu N\) in accordance with condition (i), it results in the smallest possible \(R^*^2\) which is determined
to be $d_{i_*}$, where $i_* = N - N_{\text{outlier}}$. In other words, $d_{i_*}$ is defined as the $(N - N_{\text{outlier}})$–th distance ranking in terms of farthest proximity from the decision boundary. Consider the set of training observations that either lie on the boundary of the hypersphere or beyond it. This set can be represented as $\{X_i | d_i \geq R^*^2\}$. Therefore, the following inequality arises: $|\{X_i | d_i \geq R^*^2\}| = |\{X_i | d_i > R^*^2\} \cup \{X_i | d_i = R^*^2\}| \geq N_{\text{outlier}} + 1 \geq \nu N$. It follows that no fewer than $\nu N$ samples are subjected to either exclusion or are situated precisely along the optimal classification boundary.
Digital Security Application: Facial Recognition

We evaluate the performance of DSMDD with an application to “The ORL Database of Faces” ORL dataset from AT&T Laboratories Cambridge. The dataset contains 10 facial images from 40 distinct people. The images are upright, frontal faces of 40 subjects against a dark homogeneous background.

The subjects’ images are taken with varying facial features, skin tone, and gender. The images were captured at different times and have different variations, including expressions and decorations. Each original image has size $92 \times 112$, and we scale each grayscale pixel matrix to $[0, 1]$. An example of sample images of one individual in the dataset is shown in Figure 5.11.

![Figure 5.11: A sample of images of one individual in the ORL dataset.](image)

To evaluate the performance of the one-class classifiers for matrices on the ORL dataset, we start by selecting the first eight images of a random person to form the training set. The test set consists of the last two images of that person as well as the first eight images from another random person. This yields a test set of ten matrices with dimensions $\mathbb{R}^{92 \times 112}$. The threshold is found using the bootstrap simulation procedure outlined in Algorithm 11.

The goal is to train a model with images exclusively from one person in the dataset, and predict whether unseen images include the subject the model was trained on or is a different person.
We applied DSMDD and SMDD to the 8 images in the training set, and compare their performance in detecting images of people not previously seen by the model. For DSMDD, we use the same matrix neural network architecture outlined in the Performance Settings Section. For SMDD, the kernel distance statistic $d$ is calculated using Equation (2.80), and the threshold is obtained using bootstrap simulation with $B = 5000$ independent bootstrap samples. The results for SMDD are shown in Figure 5.12. Calculating the distance statistic $d$ using Equation (5.57) for DSMDD, the results are shown in Figure 5.13.

The first two observations, which are images of the subject in the training set, are incorrectly classified as outliers by SMDD. However, there is a notable distance between the two target observations and the remaining eight observations which are correctly classified as outliers. Additionally, there is some variation between the outlier observations. The first three outliers are approximately the same distance from the boundary. The following three observations have little variation and are situated closer to the boundary. The last two observations have a distance from the boundary that is somewhere in between.

![Figure 5.12: OCC using SMDD for the ORL database of faces example.](image)
Alternatively, DSMDD is able to correctly differentiate the targets from outliers. The first two observations in the test set are correctly classified as targets, and the remaining eight observations are classified as outliers. Moreover, we see minor variation for the first three outlier observations in terms of their distance to the boundary. The remaining five outlier observations have negligible variation with respect to their proximity to the boundary.

Figure 5.13: OCC using DSMDD for the ORL database of faces example.

In summary, DSMDD is able to correctly discriminate between the targets and outliers using the ORL database of faces, whereas SMDD cannot. This demonstrates the effectiveness of the proposed one-class classifier in a real-world application when dealing with image data from an unknown distribution.
Industrial Application: Aluminum Smelting

In this section, we demonstrate a special application of the DSMDD in detecting changes in the covariance matrix for an aluminum smelting process. The dataset under consideration is from an aluminum smelter as documented by Hawkins and Maboudou-Tchao [20].

The focus of this problem relates to checking the quality of the smelter feed, and measuring the concentration of alumina ($\text{Al}_2\text{O}_3$) alongside four contaminants, namely silica ($\text{SiO}_2$), ferric oxide ($\text{Fe}_2\text{O}_3$), magnesium oxide ($\text{MgO}$), and calcium oxide ($\text{CaO}$). In total, there are $p = 5$ variables in the dataset that are critical to smelter processes. Ensuring stability in alumina and calcium oxide levels is desirable, while variations in silica, ferric oxide, and magnesium oxide are related to the input materials. The correlated nature of the concentration levels suggests a multivariate analysis is warranted.

The goal is to train a model using data exclusively from covariance matrices reflecting stable smelter operating conditions, and to detect if there’s been a change in the characteristics of the chemical process for previously unseen data. There are a total of 189 observations in the dataset. We use the first 24 observations as the training set, and the next 48 observations as the test set. To construct the covariance matrices, both the training set and test set are further segmented into smaller subgroups, each consisting of $m = 2$ observations. For example, the first covariance matrix in the training set is constructed using observations 1 and 2, the second covariance matrix is constructed using observations 3 and 4, etc. This results in a total of 12 covariance matrices in the training set and 24 covariance matrices in the test set.

In this problem, we’re tasked with identifying potential quality issues in aluminum smelting data by detecting anomalies using empirical covariance matrices. However, the singular nature of these matrices precludes the use of classical statistical methods, underscoring the DSMDD’s particular
suitability for this type of one-class classification problem where the number of observations is less than the number of variables.

The rationale behind only using a portion of the available data for the test set, rather than all 165 remaining observations, is rooted in the procedural need to suspend system operations following the detection of an anomaly, as manifested through changes in the covariance matrix. This pause is essential for a thorough investigation into the nature of the detected anomaly, thereby enabling corrective measures to be implemented before the process is reinstated.

We applied DSMDD and SMDD to the 24 covariance matrices in the test set and compare their performance in detecting a change in the structure of the covariance matrix. For DSMDD, we use a matrix neural network with 2 hidden layers. In both hidden layers, we use $(15 \times 15)$ weights, and in the output layer we use $(10 \times 10)$ weights. Leaky ReLU activation functions with leakiness $\alpha = 0.1$ are used in the hidden layers, and linear activation functions are used in the output layer. For SMDD, we use a power-Euclidean Gaussian kernel and the Lagrange multipliers are computed using a quadratic programming solver. For both methods, the threshold is obtained using bootstrap simulation with $B = 5000$ independent bootstrap samples.

To effectively detect shifts in the covariance matrix, classical statistical methods often rely on assumptions related to the underlying data distribution. Typically, these approaches require the data to be normally distributed. In contrast, DSMDD is a nonparametric technique that does not require any assumptions to be made related to the distribution of the data, such as normality. To demonstrate this point, we examine two different approaches for this aluminum smelting case study. In the first approach, we apply a pair of transformations to the dataset to shift the non-normal data towards a state that achieves approximate marginal normality. The second approach leaves the dataset unaltered, without making any transformations or assumptions related to normality.
Approach 1: Transforming Data to Meet Normality Conditions

Two preliminary transformations were suggested by Hawkins and Maboudou-Tchao [20] in an effort to achieve approximate marginal normality. First, a logarithmic transformation is performed on $\text{SiO}_2$, followed by an inverse transformation on $\text{CaO}$. Then, the model is trained exclusively on the target class data comprised of the 12 covariance matrices in the training set. We calculate the kernel distance statistic using Equation (2.80) for SMDD, and the results are shown in Figure 5.14. For DSMDD, we compute the distance statistic using Equation (5.57), and the results are shown in Figure 5.15.

While both models have correctly indicated that an outlier occurred at observation 32, there are some notable differences. For SMDD, we observe significant variation from the 13th to the 31st observation in terms of their proximity to the boundary. Specifically, the 23rd observation is much closer to the boundary than the majority of the other observations.

![Figure 5.14: OCC using SMDD for the aluminum smelting dataset example.](image)
Alternatively, DSMDD exhibits minimal variation in the observations leading up to the detected outlier in terms of distance to the boundary. With only two minor irregularities at the 21st and 29th observations, the model demonstrates highly uniform and stable predictions with very little deviation. Furthermore, the outlier’s distance from the boundary is significantly larger in DSMDD than in SMDD when considering the scale of distance, indicating a much tighter boundary distinction for DSMDD compared to SMDD.

Figure 5.15: OCC using DSMDD for the aluminum smelting dataset example.

For this particular dataset, both models identified covariance matrix 32 as an outlier, demonstrating the proposed method’s effectiveness in a real-world industrial application when the data is approximately normally distributed.
Approach 2: Direct Nonparametric Analysis, Unaltered by Transformations

In this section, no data transformations are applied, thereby facilitating a comparison of model performances in environments lacking normally distributed data. As with the previous approach, the model is trained exclusively on the target class data, comprised of the 12 covariance matrices in the training set. We calculate the kernel distance statistic using Equation (2.80) for SMDD, and the results are shown in Figure 5.16. For DSMDD, we compute the distance statistic using Equation (5.57), and the results are shown in Figure 5.17. In this setting, we observe significant differences between the two methods.

While the SMDD model correctly detects the outlier, it lies precariously close the boundary. Additionally, two new observations are incorrectly classified as outliers, specifically observations 18 and 19.

![Graph showing distance statistics for OCC using SMDD](image)

Figure 5.16: OCC using SMDD for the aluminum smelting dataset example.
The DSMDD accurately detects the outlier without misclassifying any targets, a contrast to SMDD. Moreover, DSMDD offers a superior overall data description compared to SMDD, especially when considering the spatial separation between the outlier and the boundary.

In summary, both methods demonstrated strong performance with normally distributed data. However, SMDD encountered difficulties with accurate classification of the target class when dealing with non-normal data. In contrast, DSMDD stood out with its exceptional performance, evidenced by the effective classification boundaries regardless of the underlying data distribution.

Figure 5.17: OCC using DSMDD for the aluminum smelting dataset example.
CHAPTER 6: CONCLUSION AND FUTURE WORK

Boundary-based methods for the one-class classification task are flexible non-parametric techniques that delineate optimal classification boundaries between the data class of interest and outliers without requiring any underlying assumptions related to the origin of the anomalous data. Support vector methods fall into this category, and constitute some of the most well-known one-class classification methods that can be employed to provide useful data descriptions of the target class.

In this dissertation, three novel support vector methods were proposed that fused deep learning with the underlying mechanics of various boundary-based one-class classification techniques. The first two proposed methods were developed from a least squares perspective that enabled the derivation of analytic results that were not possible with existing methods. These methods, which are designed for tabular data, include Deep Least Squares Support Vector Data Description (DLS-SVDD) and Deep Least Squares One-Class Support Vector Machine (DLS-OCSVM) which, unlike their conventional counterparts DSVDD and OC-NN, do not need to rely on subgradient methods for optimization that can result in instability during model training. Additionally, these methods use neural networks for feature mapping instead of kernel functions, so there is no need for quadratic programming solvers which are highly inefficient when working with large volumes of data. They are also more efficient to use than other least squares kernel-based support vector methods which can be solved in closed-form, since these methods do not require the entire training set to be considered during prediction after the model has been trained, unlike in LS-SVDD and LS-OCSVM, respectively.

The final method proposed in this dissertation was designed for one-class classification problems when working with second-order tensors. It is well known that the process of vectorization can
lead to structural information loss when spatial or contextual relationships exist within the data structure. The development of the Deep Support Matrix Data Description (DSMDD) was inspired by the idea of using a neural network architecture that does not rely on data vectorization, thereby keeping the data in its native form as a second-order tensor, with a boundary-based one-class classifier. In this novel approach, matrix neural networks are fused with Support Matrix Data Description (SMDD) for the one-class classification task. The mathematical foundation of the proposed model is fully derived beginning with the simplest of network architectures to arbitrarily deep network architectures. Furthermore, the integration of SMDD into deep networks requires the matrix neural network and the training methodology adhere to certain properties, which are theoretically proven.

One potential area of future research is to generalize the DSMDD for tensor data of arbitrary size. Another interesting direction of research would be to include multiple objectives into the optimization problems of DLS-SVDD and DLS-OCSVM such that one can consider minimizing the existing loss functions while also optimizing additional criteria to assist the model in learning better data descriptions with tabular data that may lack spatial or temporal characteristics.
APPENDIX A: ONE-CLASS CLASSIFICATION EVALUATION FRAMEWORK
Consider the following hypothesis test.

\[ H_0 : \text{Object is a target} \]
\[ H_1 : \text{Object is an outlier} \]

The four possible decisions associated with this hypothesis test are summarized in the table shown below.

<table>
<thead>
<tr>
<th>Predictions</th>
<th>Object from target class</th>
<th>Object from outlier class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classified as a target object</td>
<td>True Negative (Correct Decision)</td>
<td>False Negative (Type II Error)</td>
</tr>
<tr>
<td>Classified as an outlier object</td>
<td>False Positive (Type I Error)</td>
<td>True Positive (Correct Decision)</td>
</tr>
</tbody>
</table>
In the one-class classification framework, no observations originating from the outlier class are available during model training. This type of classification problem is encountered in various real-world applications where the characterization of outliers is notably challenging, which may be due to various factors such as financial constraints or practical limitations. Under this setting, conventional binary classification evaluation metrics, which rely on a balanced view of both positive and negative class examples, cannot be directly applied. Specifically, the false negative rate (FNR) – the proportion of outlier examples incorrectly classified as belonging to the target class – cannot be measured due to the absence of outliers in the training set.

Given this constraint, the focus shifts to managing the false positive rate (FPR) – the proportion of target class instances that are incorrectly classified as outliers. The control over false positives is critical in various types of one-class classification applications, such as fraud detection or disease outbreak monitoring, where the cost of erroneously labeling a target as an outlier can be high.

To evaluate the performance of one-class classifiers in this constrained setting, we use the probability of Type I error, denoted by $\alpha$, and the probability of Type II error, denoted by $\beta$. In the context of one-class classification, the Type I error rate refers to the probability of incorrectly labeling a target class instance as an outlier. Conversely, the Type II error rate represents the probability of incorrectly labeling an outlier class instance as a target.

However, as previously mentioned, due to the absence of outlier class observations during model training, the Type II error rate $\beta$ cannot be directly assessed in one-class classification scenarios. Instead, the emphasis is placed on controlling the Type I error rate $\alpha$, thereby managing the rate of false alarms. The challenge then lies in finding a threshold that balances sensitivity and specificity, without any knowledge pertaining to the outlier data.
APPENDIX B: DLS-SVDD GRADIENT DERIVATIONS
Let

\[ J_1 = R^2 + \frac{1}{2\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^2 ||W^{[\ell]}||_F^2 \]

**Proof of Theorem 3.0.1**

\[
\frac{\partial J_1}{\partial R^2} = \frac{\partial}{\partial R^2} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^2 ||W^{[\ell]}||_F^2 \right] \\
= \frac{\partial}{\partial R^2} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right)^2 \right] \\
= 1 + \frac{2}{2\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial R^2} \left[ ||\phi(x_\ast; W, B) - c||^2 - R^2 \right] \\
= 1 - \frac{1}{\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right) \\
\overset{\text{set}}{=} 0 \\
\Rightarrow 1 = \frac{1}{\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right) \\
\Rightarrow 1 = \frac{1}{\nu} \left( ||\phi(x_\ast; W, B) - c||^2 \right) - \frac{R^2}{\nu} \\
\Rightarrow \frac{R^2}{\nu} = \frac{1}{\nu} ||\phi(x_\ast; W, B) - c||^2 - 1 \\
\Rightarrow R^2 = ||\phi(x_\ast; W, B) - c||^2 - \nu

**Proof of Proposition 3.0.1**

\[
\frac{\partial J_1}{\partial c} = \frac{\partial}{\partial c} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^2 ||W^{[\ell]}||_F^2 \right] \\
= \frac{\partial}{\partial c} \left[ 1 \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right)^2 \right] \\
= \frac{2}{2\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial c} \left[ ||\phi(x_\ast; W, B) - c||^2 - R^2 \right] \\
= \frac{1}{\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial c} \left[ (\phi(x_\ast; W, B) - c)^T (\phi(x_\ast; W, B) - c) \right] \\
= \frac{-2}{\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right) (\phi(x_\ast; W, B) - c) \\
= \frac{2}{\nu} \left( ||\phi(x_\ast; W, B) - c||^2 - R^2 \right) (c - \phi(x_\ast; W, B))

182
\[ \frac{\partial J_1}{\partial W^{[2]}} = \frac{\partial}{\partial W^{[2]}} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(\mathbf{x}; W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{i=1}^2 ||W^{[i]}||^2 \right] \]

\[ = \frac{\partial}{\partial W^{[2]}} \left[ \frac{1}{2\nu} \left( ||\phi(\mathbf{x}; W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \left( ||W^{[1]}||^2 + ||W^{[2]}||^2 \right) \right] \]

\[ = 2 \frac{\partial}{\partial W^{[2]}} \left( ||\phi(\mathbf{x}; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[2]}} \left[ ||\phi(\mathbf{x}; W, B) - c||^2 - R^2 \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[2]}} \left[ ||W^{[2]}||^2 \right] \]

\[ = 2 \frac{1}{\nu} \left( ||\phi(\mathbf{x}; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[2]}} \left[ \left( \phi(\mathbf{x}; W, B) - c \right)^T \left( \phi(\mathbf{x}; W, B) - c \right) \right] + \left( \frac{2\lambda}{2} \right) W^{[2]} \]

\[ = \frac{2}{\nu} \left( ||\phi(\mathbf{x}; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[2]}} \left[ g_2 \left( W^{[2]} A^{[1]} + b^{[2]} \right) \right] + \lambda W^{[2]} \]

\[ = \delta^{[2]} A^{[1]^T} + \lambda W^{[2]} \]
Proof of Proposition 3.0.2

\[
\frac{\partial J_1}{\partial b[i]} = \frac{1}{\partial b[i]} \left[ R^2 + \frac{1}{2\nu} \left( |\phi(x_\ast; W, B) - c|^2 - R^2 \right)^2 \right] + \frac{\lambda}{2} \sum_{i=1}^{2} |W_i|^2 F_i^2 \]

\[
= \frac{1}{\partial b[i]} \left[ \frac{1}{2\nu} \left( |\phi(x_\ast; W, B) - c|^2 - R^2 \right)^2 \right]
\]

\[
= \frac{2}{\partial b[i]} \left( |\phi(x_\ast; W, B) - c|^2 - R^2 \right) \frac{\partial}{\partial b[i]} \left[ |\phi(x_\ast; W, B) - c|^2 - R^2 \right]
\]

\[
= \frac{1}{\nu} \left( |\phi(x_\ast; W, B) - c|^2 - R^2 \right) \frac{\partial}{\partial b[i]} \left[ (\phi(x_\ast; W, B) - c)^T (\phi(x_\ast; W, B) - c) \right]
\]

\[
= \frac{2}{\nu} \left( |\phi(x_\ast; W, B) - c|^2 - R^2 \right) (\phi(x_\ast; W, B) - c) \frac{\partial}{\partial b[i]} \left[ g_2 \left( W_2 A_1 + b_2 \right) \right]
\]

\[
= \frac{2}{\nu} \left( |\phi(x_\ast; W, B) - c|^2 - R^2 \right) (\phi(x_\ast; W, B) - c) \circ g_2 \left( Z \right)
\]

\[
= \delta^2
\]
Let

$$J_2 = R^2 + \frac{1}{2\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{3} ||W^{[\ell]}||^{2}_{R}$$

Proof of Proposition 3.0.3

$$\frac{\partial J_2}{\partial c} = \frac{\partial}{\partial c} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{3} ||W^{[\ell]}||^{2}_{R} \right]$$

$$= \frac{\partial}{\partial c} \left[ \frac{1}{2\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right)^2 \right]$$

$$= \frac{2}{2\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial c} \left( \phi(x_*, W, B) - c \right)$$

$$= \frac{2}{\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right) \left( c - \phi(x_*, W, B) \right)$$

$$\frac{\partial J_2}{\partial W^{[3]}} = \frac{\partial}{\partial W^{[3]}} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{3} ||W^{[\ell]}||^{2}_{R} \right]$$

$$= \frac{\partial}{\partial W^{[3]}} \left[ \frac{1}{2\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right)^2 \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[3]}} \left[ \left( ||W^{[3]}||^2_{R} \right)^2 \right]$$

$$= \frac{2}{2\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[3]}} \left[ ||\phi(x_*, W, B) - c||^2 - R^2 \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[3]}} \left[ \left( ||W^{[3]}||^2_{R} \right)^2 \right]$$

$$= \frac{2}{\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right) \left( c - \phi(x_*, W, B) \right)$$

$$= \frac{2}{\nu} \left( ||\phi(x_*, W, B) - c||^2 - R^2 \right) \left( c - \phi(x_*, W, B) \right)$$

$$= \delta^{[3]} A^{[2]T} + \lambda W^{[3]}$$
\[
\frac{\partial J_2}{\partial W^{[2]}} = \frac{\partial}{\partial W^{[2]}} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{3} ||W^{[\ell]}||_F^2 \right]
\]

\[
= \frac{\partial}{\partial W^{[2]}} \left[ \frac{1}{2\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \left( ||W^{[1]}||_F^2 + ||W^{[2]}||_F^2 + ||W^{[3]}||_F^2 \right) \right]
\]

\[
= \frac{2}{2\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[2]}} \left[ ||\phi(x_\star; W, B) - c||^2 - R^2 \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[2]}} \left[ ||W^{[2]}||_F^2 \right]
\]

\[
= \frac{1}{\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[2]}} \left[ ||\phi(x_\star; W, B) - c||^2 \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[2]}} \left[ \text{tr} \left( W^{[2]} W^{[2]T} \right) \right]
\]

\[
= \frac{1}{\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[2]}} \left[ (\phi(x_\star; W, B) - c)^T (\phi(x_\star; W, B) - c) \right] + \left( \frac{2\lambda}{2} \right) W^{[2]}
\]

\[
= \frac{2}{\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[2]}} \left[ (\phi(x_\star; W, B) - c) g_3 \left( Z^{[3]} \right) \right] + \lambda W^{[2]}
\]

\[
= \frac{2}{\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial W^{[2]}} \left[ (\phi(x_\star; W, B) - c) \circ g'_3 \left( Z^{[3]} \right) \right] W^{[3]} A^{[2]} + \frac{\partial}{\partial W^{[2]}} \left[ W^{[3]} A^{[2]} + b^{[3]} \right] + \lambda W^{[2]}
\]

\[
= W^{[3]} T \delta^{[3]} g_3 \left( Z^{[2]} \right) + \lambda W^{[2]}
\]

\[
= W^{[3]} T \delta^{[3]} \circ g'_3 \left( Z^{[2]} \right) W^{[2]} A^{[1]} + \lambda W^{[2]}
\]

\[
= \delta^{[2]} A^{[1]} + \lambda W^{[2]}
\]
\[
\frac{\partial J_2}{\partial W^{[1]}} = \frac{\partial}{\partial W^{[1]}} \left[ R^2 + \frac{1}{2\nu} \left( \|\phi(x^*; W, B) - c\|^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{3} \|W^{[\ell]}\|^2 \right] \\
= \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2\nu} \left( \|\phi(x^*; W, B) - c\|^2 - R^2 \right)^2 + \frac{\lambda}{2} \left( \|W^{[3]}\|^2 + \|W^{[2]}\|^2 + \|W^{[1]}\|^2 \right) \right] \\
= \frac{2}{\nu} \left( \|\phi(x^*; W, B) - c\|^2 - R^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \|\phi(x^*; W, B) - c\|^2 - R^2 \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[1]}} \left[ \|W^{[1]}\|^2 \right] \\
= \frac{1}{\nu} \left( \|\phi(x^*; W, B) - c\|^2 - R^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \|\phi(x^*; W, B) - c\|^2 \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[1]}} \left[ \text{tr} \left( W^{[1]} W^{[1]T} \right) \right] \\
= \frac{1}{\nu} \left( \|\phi(x^*; W, B) - c\|^2 - R^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \left( \phi(x^*; W, B) - c \right)^T \left( \phi(x^*; W, B) - c \right) \right] + \left( \frac{2\lambda}{2} \right) W^{[1]} \\
= \frac{2}{\nu} \left( \|\phi(x^*; W, B) - c\|^2 - R^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \phi(x^*; W, B) - c \right] \frac{\partial}{\partial W^{[1]}} \left[ g_3 \left( Z^{[3]} \right) \right] + \lambda W^{[1]} \\
= \frac{2}{\nu} \left( \|\phi(x^*; W, B) - c\|^2 - R^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \phi(x^*; W, B) - c \right] \frac{\partial}{\partial W^{[1]}} \left[ g_3 \left( Z^{[3]} \right) \right] W^{[3]} A^{[2]} + b^{[3]} + \lambda W^{[1]} \\
= W^{[3]} T \delta^{[3]} \frac{\partial}{\partial W^{[1]}} \left[ g_2 \left( Z^{[2]} \right) \right] + \lambda W^{[1]} \\
= W^{[3]} T \delta^{[3]} \circ \hat{g}_2 \left( Z^{[2]} \right) \frac{\partial}{\partial W^{[1]}} \left[ W^{[2]} A^{[1]} + b^{[2]} \right] + \lambda W^{[1]} \\
= W^{[2]} T \delta^{[2]} \frac{\partial}{\partial W^{[1]}} \left[ g_1 \left( Z^{[1]} \right) \right] + \lambda W^{[1]} \\
= W^{[2]} T \delta^{[2]} \circ \hat{g}_1 \left( Z^{[1]} \right) \frac{\partial}{\partial W^{[1]}} \left[ W^{[1]} x + b^{[1]} \right] + \lambda W^{[1]} \\
= \delta^{[1]} x^T + \lambda W^{[1]}
\]
Proof of Proposition 3.0.4

\[
\frac{\partial J_2}{\partial \mathbf{b}^{[3]}} = \frac{\partial}{\partial \mathbf{b}^{[3]}} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{l=1}^{3} ||\mathbf{W}^{[l]}||^2_F \right] \\
= \frac{\partial}{\partial \mathbf{b}^{[3]}} \left[ \frac{1}{2\nu} \left( ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 - R^2 \right)^2 \right] \\
= \frac{2}{2\nu} \left( ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 - R^2 \right) \frac{\partial}{\partial \mathbf{b}^{[3]}} \left[ ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 - R^2 \right] \\
= \frac{1}{\nu} \left( ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 - R^2 \right) \frac{\partial}{\partial \mathbf{b}^{[3]}} \left[ ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 ight] \\
= \frac{1}{\nu} \left( ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 - R^2 \right) \frac{\partial}{\partial \mathbf{b}^{[3]}} \left[ (\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c})^T (\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}) \right] \\
= \frac{2}{\nu} \left( ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 - R^2 \right) (\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}) \frac{\partial}{\partial \mathbf{b}^{[3]}} \left[ g_3 \left( \mathbf{Z}^{[3]} \right) \right] \\
= \frac{2}{\nu} \left( ||\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}||^2 - R^2 \right) (\phi(\mathbf{x}_i; \mathbf{W}, \mathbf{B}) - \mathbf{c}) \circ g'_3 \left( \mathbf{Z}^{[3]} \right) \\
= \mathbf{W}^{[3]} \frac{\partial}{\partial \mathbf{b}^{[3]}} \left[ g_2 \left( \mathbf{Z}^{[2]} \right) \right] \\
= \mathbf{W}^{[3]} \circ g'_2 \left( \mathbf{Z}^{[2]} \right) \frac{\partial}{\partial \mathbf{b}^{[2]}} \left[ \mathbf{W}^{[2]} \mathbf{A}^{[1]} + \mathbf{b}^{[2]} \right] \\
= \delta^{[2]} 
\]
\[
\frac{\partial J_2}{\partial b^{[1]}} = \frac{\partial}{\partial b^{[1]}} \left[ R^2 + \frac{1}{2\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right)^2 + \frac{\lambda}{2} \sum_{i=1}^3 ||W^{[i]}||_F^2 \right] \\
= \frac{\partial}{\partial b^{[1]}} \left[ \frac{1}{2\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right)^2 \right] \\
= \frac{2}{2\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial b^{[1]}} \left[ ||\phi(x_\star; W, B) - c||^2 - R^2 \right] \\
= \frac{1}{\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial b^{[1]}} \left[ ||\phi(x_\star; W, B) - c||^2 \right] \\
= \frac{1}{\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) \frac{\partial}{\partial b^{[1]}} \left[ (\phi(x_\star; W, B) - c)^T (\phi(x_\star; W, B) - c) \right] \\
= \frac{2}{\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) (\phi(x_\star; W, B) - c) \frac{\partial}{\partial b^{[1]}} \left[ g_3 (Z^{[3]}) \right] \\
= \frac{2}{\nu} \left( ||\phi(x_\star; W, B) - c||^2 - R^2 \right) (\phi(x_\star; W, B) - c) \frac{\partial}{\partial b^{[1]}} \left[ W^{[3]}A^{[2]} + b^{[3]} \right] \\
= W^{[3]^T} g^{[3]} \frac{\partial}{\partial b^{[1]}} \left[ g_2 (Z^{[2]}) \right] \\
= W^{[3]^T} g^{[3]} \circ g_2 \left( Z^{[2]} \right) \frac{\partial}{\partial b^{[1]}} \left[ W^{[2]}A^{[1]} + b^{[2]} \right] \\
= W^{[2]^T} g^{[2]} \frac{\partial}{\partial b^{[1]}} \left[ g_1 (Z^{[1]}) \right] \\
= W^{[2]^T} g^{[2]} \circ g_1 \left( Z^{[1]} \right) \frac{\partial}{\partial b^{[1]}} \left[ W^{[1]}x_\star + b^{[1]} \right] \\
= \delta^{[1]}
\]
APPENDIX C: DLS-OCSVDD GRADIENT DERIVATIONS
Let

\[ J_3 = \frac{1}{2} \left( \| \phi(x; W, B) - c \|_2^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{2} \| W^{[\ell]} \|_F^2 \]

**Proof of Proposition 3.0.5**

\[
\frac{\partial J_3}{\partial c} = \frac{\partial}{\partial c} \left[ \frac{1}{2} \left( \| \phi(x; W, B) - c \|_2^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{2} \| W^{[\ell]} \|_F^2 \right] \\
= \frac{\partial}{\partial c} \left[ \frac{1}{2} \left( \| \phi(x; W, B) - c \|_2^2 \right)^2 \right] \\
= \frac{2}{2} \| \phi(x; W, B) - c \|_2^2 \frac{\partial}{\partial c} \left[ (\phi(x; W, B) - c) \left( \phi(x; W, B) - c \right)^T (\phi(x; W, B) - c) \right] \\
= -2\| \phi(x; W, B) - c \|_2^2 (\phi(x; W, B) - c) \\
= 2\| \phi(x; W, B) - c \|_2^2 (c - \phi(x; W, B))
\]

\[
\frac{\partial J_3}{\partial W^{[1]}} = \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2} \left( \| \phi(x; W, B) - c \|_2^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{2} \| W^{[\ell]} \|_F^2 \right] \\
= \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2} \left( \| \phi(x; W, B) - c \|_2^2 \right)^2 \right] + \frac{\lambda}{2} \frac{\partial}{\partial W^{[1]}} \left[ \sum_{\ell=1}^{2} \| W^{[\ell]} \|_F^2 \right] \\
= \frac{2}{2} \left( \| \phi(x; W, B) - c \|_2^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \| \phi(x; W, B) - c \|_2^2 \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[1]}} \left[ \| W^{[2]} \|_F^2 \right] \\
= \left( \| \phi(x; W, B) - c \|_2^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ (\phi(x; W, B) - c) \left( \phi(x; W, B) - c \right)^T (\phi(x; W, B) - c) \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial W^{[1]}} \left[ tr(W^{[2]}W^{[2]T}) \right] \\
= 2 \left( \| \phi(x; W, B) - c \|_2^2 \right) (\phi(x; W, B) - c) \circ g_2 \left( W^{[2]}A^{[1]} + b^{[2]} \right) + \left( \frac{\lambda}{2} \right) W^{[2]} \\
= 2 \left( \| \phi(x; W, B) - c \|_2^2 \right) (\phi(x; W, B) - c) \circ g_2 \left( Z^{[2]} A^{[1]T} + \lambda W^{[2]} \right) \\
= 2 \left( \| \phi(x; W, B) - c \|_2^2 \right) (\phi(x; W, B) - c) \circ g_2 \left( Z^{[2]} A^{[1]T} + \lambda W^{[2]} \right) \\
= \delta_1^{[2]} A^{[1]T} + \lambda W^{[2]}
\]
\[
\frac{\partial J_3}{\partial W^{[1]}} = \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{2} ||W^{[\ell]}||_F^2 \right] \\
= \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right)^2 + \frac{\lambda}{2} \left( ||W^{[1]}||_F^2 + ||W^{[2]}||_F^2 \right) \right] \\
= \frac{2}{2} \left( ||\phi(x; W, B) - c||^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right) + \frac{\lambda}{2} \frac{\partial}{\partial W^{[1]}} \left[ ||W^{[1]}||_F^2 \right] \right] \\
= \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right) + \frac{\lambda}{2} \frac{\partial}{\partial W^{[1]}} \left[ tr(W^{[1]} W^{[1]T}) \right] \right] \\
= W^{[2]T} \left( \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right) \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right) + \frac{\lambda}{2} \frac{\partial}{\partial W^{[1]}} \left[ tr(W^{[1]} W^{[1]T}) \right] \right] \right) \\
= W^{[2]T} \frac{\partial}{\partial W^{[1]}} \left[ \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right) + \frac{\lambda}{2} \frac{\partial}{\partial W^{[1]}} \left[ tr(W^{[1]} W^{[1]T}) \right] \right) \\
= \delta^{[1]} x^T + \lambda W^{[1]} \\

Proof of Proposition 3.0.6 \[
\frac{\partial J_3}{\partial b^{[2]}} = \frac{\partial}{\partial b^{[2]}} \left[ \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{2} ||W^{[\ell]}||_F^2 \right] \\
= \frac{\partial}{\partial b^{[2]}} \left[ \frac{1}{2} \left( ||\phi(x; W, B) - c||^2 \right)^2 \right] \\
= \frac{2}{2} \left( ||\phi(x; W, B) - c||^2 \right) \frac{\partial}{\partial b^{[2]}} \left[ ||\phi(x; W, B) - c||^2 \right] \\
= \left( ||\phi(x; W, B) - c||^2 \right) \frac{\partial}{\partial b^{[2]}} \left[ (\phi(x; W, B) - c)^T (\phi(x; W, B) - c) \right] \\
= 2 \left( ||\phi(x; W, B) - c||^2 \right) (\phi(x; W, B) - c) \frac{\partial}{\partial b^{[2]}} \left[ g_2 \left( W^{[2]} A^{[1]} + b^{[2]} \right) \right] \\
= 2 \left( ||\phi(x; W, B) - c||^2 \right) (\phi(x; W, B) - c) \frac{\partial}{\partial b^{[2]}} \left[ Z^{[2]} \right] \\
= \delta^{[2]}_1 
\]
\[
\frac{\partial J_1}{\partial b^{[1]}} = \frac{\partial}{\partial b^{[1]}} \left[ \frac{1}{2} \left( \left\| \phi(x^*; W, B) - c \right\|^2 \right)^2 + \frac{\lambda}{2} \sum_{t=1}^{2} \left\| W^{[t]} \right\|^2 \right]
\]
\[
= \frac{\partial}{\partial b^{[1]}} \left[ \frac{1}{2} \left( \left\| \phi(x^*; W, B) - c \right\|^2 \right)^2 \right]
\]
\[
= \left( \frac{2}{2} \right) \left( \left\| \phi(x^*; W, B) - c \right\|^2 \right) \frac{\partial}{\partial b^{[1]}} \left[ \left\| \phi(x^*; W, B) - c \right\|^2 \right]
\]
\[
= \left( \left\| \phi(x^*; W, B) - c \right\|^2 \right) \frac{\partial}{\partial b^{[1]}} \left[ \left( \phi(x^*; W, B) - c \right)^T \left( \phi(x^*; W, B) - c \right) \right]
\]
\[
= W^{[2]}^T \left( \left\| \phi(x^*; W, B) - c \right\|^2 \right) \left( \phi(x^*; W, B) - c \right)^T \left( \phi(x^*; W, B) - c \right) \frac{\partial}{\partial b^{[1]}} \left[ g_1(W^{[1]}x^* + b^{[1]}) \right]
\]
\[
= W^{[2]}^T \delta_1^{[2]} \circ g_1^\prime \left( Z^{[1]} \right) \frac{\partial}{\partial b^{[1]}} \left[ W^{[1]}x^* + b^{[1]} \right]
\]
\[
= \delta_1^{[1]}
\]
APPENDIX D: DLS-OCSVM GRADIENT DERIVATIONS
Let

\[ J_1 = \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \sum_{i=1}^2 ||V[i]||_F^2 \]

Proof of Theorem 4.0.1

\[
\frac{\partial J_1}{\partial \rho} = \frac{\partial}{\partial \rho} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \sum_{i=1}^2 ||V[i]||_F^2 \right] = -1 + \frac{1}{\nu} \left( \rho - w^T \phi(x_*, V, B) \right) = 0
\]

\[
\Rightarrow 1 = \frac{1}{\nu} \left( \rho - w^T \phi(x_*, V, B) \right)
\]

\[
\Rightarrow 1 = \frac{\rho}{\nu} - \frac{w^T \phi(x_*, V, B)}{\nu}
\]

\[
\Rightarrow \frac{\rho}{\nu} = 1 + \frac{w^T \phi(x_*, V, B)}{\nu}
\]

\[
\Rightarrow \rho = \nu + w^T \phi(x_*, V, B)
\]

Proof of Proposition 4.0.1

\[
\frac{\partial J_1}{\partial w} = \frac{\partial}{\partial w} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \sum_{i=1}^2 ||V[i]||_F^2 \right] = \frac{\partial}{\partial w} \left[ \frac{1}{2} w^T w + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 \right] = \frac{1}{2} w + \frac{2}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{\partial}{\partial w} \left[ \rho - w^T \phi(x_*, V, B) \right] = w - \frac{1}{\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \phi(x_*, V, B) = \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) \phi(x_*, V, B) + w
\]
\[
\frac{\partial J_1}{\partial \mathbf{V}[2]} = \frac{\partial}{\partial \mathbf{V}[2]} \left[ \frac{1}{2} \mathbf{w}^T \mathbf{w} - \rho + \frac{1}{2\nu} \left( \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^2 ||\mathbf{V}^{[\ell]}||_F^2 \right]
\]

\[
= \frac{\partial}{\partial \mathbf{V}[2]} \left[ \frac{1}{2\nu} \left( \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right)^2 + \frac{\lambda}{2} \left( ||\mathbf{V}^{[1]}||_F^2 + ||\mathbf{V}^{[2]}||_F^2 \right) \right]
\]

\[
= \frac{1}{2\nu} \left( \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right) \frac{\partial}{\partial \mathbf{V}[2]} \left[ \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right] + \frac{\lambda}{2} \frac{\partial}{\partial \mathbf{V}[2]} \left[ ||\mathbf{V}^{[2]}||_F^2 \right]
\]

\[
= \frac{1}{\nu} \left( \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right) \mathbf{w}^T \frac{\partial}{\partial \mathbf{V}[2]} \left[ \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right] + \frac{\lambda}{2} \frac{\partial}{\partial \mathbf{V}[2]} \left[ tr\left( \mathbf{V}^{[2]} \mathbf{V}^{[2]^T} \right) \right]
\]

\[
= \frac{1}{\nu} \left( \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) - \rho \right) \mathbf{w}^T \frac{\partial}{\partial \mathbf{V}[2]} \left[ \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right] + \frac{\lambda}{2} \frac{\partial}{\partial \mathbf{V}[2]} \left[ \mathbf{V}^{[2]} \mathbf{A}^{[1]} + \mathbf{b}^{[2]} \right] + \lambda \mathbf{V}^{[2]}
\]

\[
= \delta^{[2]} \mathbf{A}^{[1]^T} + \lambda \mathbf{V}^{[2]}
\]

\[
\frac{\partial J_1}{\partial \mathbf{V}[1]} = \frac{\partial}{\partial \mathbf{V}[1]} \left[ \frac{1}{2} \mathbf{w}^T \mathbf{w} - \rho + \frac{1}{2\nu} \left( \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^2 ||\mathbf{V}^{[\ell]}||_F^2 \right]
\]

\[
= \frac{\partial}{\partial \mathbf{V}[1]} \left[ \frac{1}{2\nu} \left( \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right)^2 + \frac{\lambda}{2} \left( ||\mathbf{V}^{[1]}||_F^2 + ||\mathbf{V}^{[2]}||_F^2 \right) \right]
\]

\[
= \frac{1}{2\nu} \left( \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right) \frac{\partial}{\partial \mathbf{V}[1]} \left[ \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right] + \frac{\lambda}{2} \frac{\partial}{\partial \mathbf{V}[1]} \left[ ||\mathbf{V}^{[1]}||_F^2 \right]
\]

\[
= \frac{1}{\nu} \left( \rho - \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right) \frac{\partial}{\partial \mathbf{V}[1]} \left[ \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) \right] + \frac{\lambda}{2} \frac{\partial}{\partial \mathbf{V}[1]} \left[ tr\left( \mathbf{V}^{[1]} \mathbf{V}^{[1]^T} \right) \right]
\]

\[
= \mathbf{V}^{[2]^T} \left( \frac{1}{\nu} \left( \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) - \rho \right) \mathbf{w}^T \phi(\mathbf{x}_*; \mathbf{V}, \mathbf{B}) + \frac{\lambda}{2} \right) \mathbf{V}^{[1]} + \lambda \mathbf{V}^{[1]}
\]

\[
= \delta^{[1]} \mathbf{x}_*^T + \lambda \mathbf{V}^{[1]}
\]
Proof of Proposition 4.0.2

\[
\frac{\partial J}{\partial b^{[2]}} = \frac{\partial}{\partial b^{[2]}} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \sum_{i=1}^2 \|V_i\|^2 \right]
= \frac{\partial}{\partial b^{[2]}} \left[ \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 \right]
= \frac{2}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{\partial}{\partial b^{[2]}} \left[ \rho - w^T \phi(x_*, V, B) \right]
= -\frac{1}{\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{\partial}{\partial b^{[2]}} \left[ w^T g_2 \left( V^{[2]} A^{[1]} + b^{[2]} \right) \right]
= -\frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T g'_2 \left( V^{[2]} A^{[1]} + b^{[2]} \right) \frac{\partial}{\partial b^{[2]}} \left[ g_2 \left( V^{[2]} A^{[1]} + b^{[2]} \right) \right]
= -\frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T g'_2 \left( Z^{[2]} \right)
= \delta^{[2]}
\]

\[
\frac{\partial J}{\partial b^{[1]}} = \frac{\partial}{\partial b^{[1]}} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \sum_{i=1}^2 \|V_i\|^2 \right]
= \frac{\partial}{\partial b^{[1]}} \left[ \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 \right]
= \frac{2}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{\partial}{\partial b^{[1]}} \left[ \rho - w^T \phi(x_*, V, B) \right]
= \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T g'_2 \left( V^{[2]} A^{[1]} + b^{[2]} \right) \frac{\partial}{\partial b^{[1]}} \left[ g_1 \left( V^{[1]} x_* + b^{[1]} \right) \right]
= \frac{V^{[2]} w^T}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T g'_2 \left( Z^{[2]} \right) \frac{\partial}{\partial b^{[1]}} \left[ g_1 \left( V^{[1]} x_* + b^{[1]} \right) \right]
= \frac{V^{[2]} w^T}{\nu} \delta^{[2]} \circ g'_1 \left( V^{[1]} x_* + b^{[1]} \right) \frac{\partial}{\partial b^{[1]}} \left[ V^{[1]} x_* + b^{[1]} \right]
= \frac{V^{[2]} w^T}{\nu} \delta^{[2]} \circ g'_1 \left( Z^{[1]} \right)
= \delta^{[1]}
\]
Let

\[ J_2 = \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^{3} ||V^{[\ell]}||_F^2 \]

Proof of Proposition 4.0.3

\[ \frac{\partial J_2}{\partial w} = \frac{\partial}{\partial w} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 \right] + \frac{\lambda}{2} \sum_{\ell=1}^{3} \frac{3}{2} ||V^{[\ell]}||_F^2 \]

\[ = \frac{\partial}{\partial w} \left[ \frac{1}{2} w^T w + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 \right] \]

\[ = \frac{2}{\nu} w + \frac{2}{\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{2}{\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{1}{\nu} \phi(x_*, V, B) \]

\[ = \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) \phi(x_*, V, B) + w \]

\[ \frac{\partial J_2}{\partial V^{[3]}} = \frac{\partial}{\partial V^{[3]}} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 \right] + \frac{\lambda}{2} \sum_{\ell=1}^{3} \frac{3}{2} ||V^{[\ell]}||_F^2 \]

\[ = \frac{\partial}{\partial V^{[3]}} \left[ \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 \right] + \frac{\lambda}{2} \frac{3}{2} \left( ||V^{[1]}||_F^2 ||V^{[2]}||_F^2 + ||V^{[3]}||_F^2 \right) \]

\[ = \frac{\partial}{\partial V^{[3]}} \left[ \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 \right] + \frac{\lambda}{2} \frac{3}{2} \left( ||V^{[1]}||_F^2 ||V^{[2]}||_F^2 + ||V^{[3]}||_F^2 \right) \]

\[ = \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \frac{\partial}{\partial V^{[3]}} \left( Z^{[3]} \right) + \frac{2\lambda}{2} V^{[3]} \]

\[ = \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \frac{\partial}{\partial V^{[3]}} \left( g_3 \left( V^{[3]} A^{[2]} + b^{[3]} \right) \right) + \frac{2\lambda}{2} V^{[3]} \]

\[ = \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \frac{\partial}{\partial V^{[3]}} \left( Z^{[3]} A^{[2]} + b^{[3]} \right) + \lambda V^{[3]} \]

\[ = \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \frac{\partial}{\partial V^{[3]}} \left( Z^{[3]} A^{[2]} + b^{[3]} + \lambda V^{[3]} \right) \]

\[ = \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \frac{\partial}{\partial V^{[3]}} \left( Z^{[3]} A^{[2]} + \lambda V^{[3]} \right) \]

\[ = g_3^{[3]} A^{[2]T} + \lambda V^{[3]} \]
\[
\frac{\partial J_2}{\partial V[2]} = \frac{\partial}{\partial V[2]} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^3 ||V[\ell]||_F^2 \right]
\]
\[
= \frac{\partial}{\partial V[2]} \left[ \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \left( ||V[1]||_F^2 + ||V[2]||_F^2 + ||V[3]||_F^2 \right) \right]
\]
\[
= \frac{2}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{\partial}{\partial V[2]} \left[ \rho - w^T \phi(x_*, V, B) \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial V[2]} \left[ ||V[2]||_F^2 \right]
\]
\[
= \frac{1}{\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{\partial}{\partial V[2]} \left[ w^T \phi(x_*, V, B) \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial V[2]} \left[ \text{tr} \left( V[2]^T V[2] \right) \right]
\]
\[
= \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \frac{\partial}{\partial V[2]} \left[ g_3 \left( Z[3] \right) \right] + \left( \frac{2\lambda}{2} \right) V[2]
\]
\[
= \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \circ g_3 \left( Z[3] \right) \frac{\partial}{\partial V[2]} \left[ V[3]^T A[2] + b[3] \right] + \lambda V[2]
\]
\[
\]
\[
\]

\[
\frac{\partial J_2}{\partial V[1]} = \frac{\partial}{\partial V[1]} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \sum_{\ell=1}^3 ||V[\ell]||_F^2 \right]
\]
\[
= \frac{\partial}{\partial V[1]} \left[ \frac{1}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right)^2 + \frac{\lambda}{2} \left( ||V[1]||_F^2 + ||V[2]||_F^2 + ||V[3]||_F^2 \right) \right]
\]
\[
= \frac{2}{2\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{\partial}{\partial V[1]} \left[ \rho - w^T \phi(x_*, V, B) \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial V[1]} \left[ ||V[1]||_F^2 \right]
\]
\[
= \frac{1}{\nu} \left( \rho - w^T \phi(x_*, V, B) \right) \frac{\partial}{\partial V[1]} \left[ w^T \phi(x_*, V, B) \right] + \left( \frac{\lambda}{2} \right) \frac{\partial}{\partial V[1]} \left[ \text{tr} \left( V[1]^T V[1] \right) \right]
\]
\[
= \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \frac{\partial}{\partial V[1]} \left[ g_2 \left( Z[2] \right) \right] \frac{\partial}{\partial Z[2]} \left[ g_1 \left( Z[1] \right) \right] + \left( \frac{2\lambda}{2} \right) V[1]
\]
\[
= \frac{1}{\nu} \left( w^T \phi(x_*, V, B) - \rho \right) w^T \circ g_2 \left( Z[2] \right) \frac{\partial}{\partial V[1]} \left[ V[3]^T A[2] + b[3] \right] + \lambda V[1]
\]
\[
\]
\[
\]
\[
\]
\[
= \delta[1] x_*^T + \lambda V[1]
\]
Proof of Proposition 4.0.4

\[
\frac{\partial J_2}{\partial b[i]} = \frac{\partial}{\partial b[i]} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} (\rho - w^T \phi(x^*; V, B))^2 + \frac{\lambda}{2} \sum_{i=1}^{3} ||V[i]||_F^2 \right]
\]

\[
= \frac{\partial}{\partial b[i]} \left[ \frac{1}{2\nu} (\rho - w^T \phi(x^*; V, B))^2 \right]
\]

\[
= \frac{2}{2\nu} \left( \rho - w^T \phi(x^*; V, B) \right) \frac{\partial}{\partial b[i]} \left[ \rho - w^T \phi(x^*; V, B) \right]
\]

\[
= -\frac{1}{\nu} \left( \rho - w^T \phi(x^*; V, B) \right) \frac{\partial}{\partial b[i]} \left[ w^T g_3 (V[3] A[2] + b[3]) \right]
\]

\[
= \frac{1}{\nu} \left( w^T \phi(x^*; V, B) - \rho \right) w^T \circ g_3 \left( V[3] A[2] + b[3] \right) \frac{\partial}{\partial b[i]} \left[ V[3] A[2] + b[3] \right]
\]

\[
= \frac{1}{\nu} \left( w^T \phi(x^*; V, B) - \rho \right) w^T \circ g_3 \left( Z[2] \right)
\]

\[
= \delta[3]
\]
\[
\frac{\partial J_2}{\partial b^{[1]}} = \frac{\partial}{\partial b^{[1]}} \left[ \frac{1}{2} w^T w - \rho + \frac{1}{2\nu} \left( \rho - w^T \phi(x^{\ast}; V, B) \right)^2 + \frac{\lambda}{2} \sum_{t=1}^{3} \| V^{(t)} \|_F^2 \right]
\]

\[
= \frac{\partial}{\partial b^{[1]}} \left[ \frac{1}{2\nu} \left( \rho - w^T \phi(x^{\ast}; V, B) \right)^2 \right]
\]

\[
= \frac{2}{2\nu} \left( \rho - w^T \phi(x^{\ast}; V, B) \right) \frac{\partial}{\partial b^{[1]}} \left[ \rho - w^T \phi(x^{\ast}; V, B) \right]
\]

\[
= -\frac{1}{\nu} \left( \rho - w^T \phi(x^{\ast}; V, B) \right) \frac{\partial}{\partial b^{[1]}} \left[ w^T g_3 \left( V^{[3]} A^{[2]} + b^{[3]} \right) \right]
\]

\[
= \frac{1}{\nu} \left( w^T \phi(x^{\ast}; V, B) - \rho \right) w^T \circ g_3 \left( Z^{[3]} \right) \frac{\partial}{\partial b^{[1]}} \left[ V^{[1]} A^{[2]} + b^{[3]} \right]
\]

\[
= V^{[3]} T \delta^{[3]} \frac{\partial}{\partial b^{[1]}} \left[ g_2 \left( Z^{[2]} \right) \right]
\]

\[
= V^{[3]} T \delta^{[3]} \circ g_2^\prime \left( Z^{[2]} \right) \frac{\partial}{\partial b^{[1]}} \left[ V^{[2]} A^{[1]} + b^{[2]} \right]
\]

\[
= \frac{\partial}{\partial b^{[1]}} \left[ g_1 \left( Z^{[1]} \right) \right]
\]

\[
= V^{[2]} T \delta^{[2]} \circ g_1^\prime \left( Z^{[1]} \right) \frac{\partial}{\partial b^{[1]}} \left[ V^{[1]} x^{\ast} + b^{[1]} \right]
\]

\[
= \delta^{[1]}
\]
APPENDIX E: DSMDD SUBGRADIENT DERIVATIONS
Let

\[ J_1 = R^2 + \frac{1}{\nu} \left( \| \phi(X_\ast; W) - c \|^2 - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{2} \left( \| U^{[\ell]} \|^2_F + \| V^{[\ell]} \|^2_F \right) \]

Proof of Proposition 5.0.1

\[ \partial_R (J_1) = \partial_R \left[ R^2 + \frac{1}{\nu} \left( \| \phi(X_\ast; W) - c \|^2 - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{2} \left( \| U^{[\ell]} \|^2_F + \| V^{[\ell]} \|^2_F \right) \right] \]

\[ = \partial_R \left[ R^2 - \frac{1}{\nu} R^2 \right] \]

\[ = 2R - \left( \frac{2R}{\nu} \right) \]

\[ = 2R \left( 1 - \frac{1}{\nu} \right) \]

\[ \partial_{U^{[2]}} (J_1) = \partial_{U^{[2]}} \left[ R^2 + \frac{1}{\nu} \left( \| \phi(X_\ast; W) - c \|^2 - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{2} \left( \| U^{[\ell]} \|^2_F + \| V^{[\ell]} \|^2_F \right) \right] \]

\[ = \partial_{U^{[2]}} \left[ \frac{1}{\nu} \left( \| \phi(X_\ast; W) - c \|^2 - R^2 \right) + \left( \frac{\lambda}{2} \right) \left( \| U^{[1]} \|^2_F + \| V^{[1]} \|^2_F \right) \sum_{\ell=1}^{2} \left( \| U^{[\ell]} \|^2_F + \| V^{[\ell]} \|^2_F \right) \right] \]

\[ = \partial_{U^{[2]}} \left[ \frac{1}{\nu} \| A^{[2]} - C \|^2_F \right] + \left( \frac{\lambda}{2} \right) \partial_{U^{[2]}} \left[ \| U^{[2]} \|^2_F \right] \]

\[ = \left( \frac{2}{\nu} \right) \left( A^{[2]} - C \right) \partial_{U^{[2]}} \left[ g_2' \left( Z^{[2]} \right) - C \right] + \left( \frac{\lambda}{2} \right) \partial_{U^{[2]}} \left[ \| U^{[2]} \|^2_F \right] \]

\[ = \left( \frac{2}{\nu} \right) \left( A^{[2]} - C \right) \circ g_2' \left( Z^{[2]} \right) \partial_{U^{[2]}} \left[ U^{[2]} A^{[1]} V^{[2]^T} \right] + \left( \frac{2\lambda}{2} \right) U^{[2]} \]

\[ = \left[ \left( \frac{2}{\nu} \right) \left( A^{[2]} - C \right) \circ g_2' \left( Z^{[2]} \right) V^{[2]} \right] A^{[1]^T} + \lambda U^{[2]} \]

\[ = \delta_a^{[2]} A^{[1]^T} + \lambda U^{[2]} \]
\[
\partial_{\nu_2} (J_1) = \partial_{\nu_2} \left[ R^2 + \frac{1}{\nu} \left( ||\phi(X_\nu;W) - C||_F^2 - R^2 \right) + \frac{\lambda}{2} \sum_{i=1}^2 \left( ||U_i||_F^2 + ||V_i||_F^2 \right) \right] \\
= \partial_{\nu_2} \left[ \frac{1}{\nu} \left( ||\phi(X_\nu;W) - C||_F^2 - R^2 \right) + \frac{\lambda}{2} \left( ||U_1||_F^2 + ||V_1||_F^2 + ||U_2||_F^2 + ||V_2||_F^2 \right) \right] \\
= \partial_{\nu_2} \left[ \frac{1}{\nu} ||A_2 - C||_F^2 \right] + \left( \frac{\lambda}{2} \right) \partial_{\nu_2} \left[ ||V_i||_F^2 \right] \\
= \left( \frac{2}{\nu} \right) \left( A_2 - C \right)^T \partial_{\nu_2} \left[ g_2^\prime (Z_2) - C \right] + \left( \frac{\lambda}{2} \right) \partial_{\nu_2} \left[ tr \left( V_2^2 \right) \right] \\
= \left( \frac{2}{\nu} \right) \left( \left( A_2 - C \right) \circ g_2^\prime (Z_2) \right)^T \partial_{\nu_2} \left[ U_2^T A_1^T V_2^2 \right] + \left( \frac{2\lambda}{2} \right) \nu V_2 \\
= \left[ \left( \frac{2}{\nu} \right) \left( \left( A_2 - C \right) \circ g_2^\prime (Z_2) \right)^T U_2^2 \right] A_1^T + \lambda V_2 \\
= \delta_{\nu_2} A_1^T + \lambda V_2
\]

\[
\partial_{U_1} (J_1) = \partial_{U_1} \left[ R^2 + \frac{1}{\nu} \left( ||\phi(X_\nu;W) - C||_F^2 - R^2 \right) + \frac{\lambda}{2} \sum_{i=1}^2 \left( ||U_i||_F^2 + ||V_i||_F^2 \right) \right] \\
= \partial_{U_1} \left[ \frac{1}{\nu} \left( ||\phi(X_\nu;W) - C||_F^2 - R^2 \right) + \frac{\lambda}{2} \left( ||U_1||_F^2 + ||V_1||_F^2 + ||U_2||_F^2 + ||V_2||_F^2 \right) \right] \\
= \partial_{U_1} \left[ \frac{1}{\nu} ||A_2 - C||_F^2 \right] + \left( \frac{\lambda}{2} \right) \partial_{U_1} \left[ ||V_i||_F^2 \right] \\
= \left( \frac{2}{\nu} \right) \left( A_2 - C \right) \partial_{U_1} \left[ g_2 (Z_2) - C \right] + \left( \frac{\lambda}{2} \right) \partial_{U_1} \left[ tr \left( U_1 U_1^T \right) \right] \\
= \left[ \left( \frac{2}{\nu} \right) \left( A_2 - C \right) \circ g_2 (Z_2) \right] \partial_{U_1} \left[ U_2^2 A_1^T V_2^2 \right] + \left( \frac{2\lambda}{2} \right) U_1 \\
= \left[ \left( \frac{2}{\nu} \right) \left( A_2 - C \right) \circ g_2 (Z_2) \right] \partial_{U_1} \left[ U_2^2 g_1 (U_1^T X V_2^2) \right] V_2^2 \lambda U_1 \\
= U_2^2 \left[ \left( \frac{2}{\nu} \right) \left( A_2 - C \right) \circ g_2 (Z_2) \right] V_2^2 \partial_{U_1} \left[ g_1 (U_1^T X V_2^2) \right] + \lambda U_1 \\
= U_2^2 \left[ \left( \frac{2}{\nu} \right) \left( A_2 - C \right) \circ g_2 (Z_2) \right] V_2^2 \circ g_1 (Z_1) \partial_{U_1} \left[ U_1^T X V_2^2 \right] + \lambda U_1 \\
= \left[ U_2^2 \right] \left[ \left( \frac{2}{\nu} \right) \left( A_2 - C \right) \circ g_2 (Z_2) \right] V_2^2 \circ g_1 (Z_1) V_1^T X^T + \lambda U_1 \\
= \left[ U_2^2 \right] \delta_{U_1} \circ g_1 (Z_1) V_1^T X^T + \lambda U_1 \\
= \delta_{U_1} X^T + \lambda U_1
\]
\[ \partial_{\nu[1]} (J_1) = \partial_{\nu[1]} \left[ R^2 + \frac{1}{\nu} \left( \| \phi (X, \nu \omega) - C \|_F^2 - R^2 \right) + \frac{\lambda}{2} \sum_{i=1}^{2} \left( \| U^{[i]} \|_F^2 + \| V^{[i]} \|_F^2 \right) \right] \\
= \partial_{\nu[1]} \left[ \frac{1}{\nu} \left( \| \phi (X, \nu \omega) - C \|_F^2 - R^2 \right) + \frac{\lambda}{2} \left( \| U^{[1]} \|_F^2 + \| V^{[1]} \|_F^2 + \| U^{[2]} \|_F^2 + \| V^{[2]} \|_F^2 \right) \right] \\
= \partial_{\nu[1]} \left[ \frac{1}{\nu} \| A^{[2]} - C \|_F^2 \right] + \left( \frac{\lambda}{2} \right) \partial_{\nu[1]} \left[ \| V^{[1]} \|_F^2 \right] \\
= \left( \frac{2}{\nu} \right) \left( A^{[2]} - C \right)^T \partial_{\nu[1]} \left[ g_2 (Z^{[2]}) - C \right] + \left( \frac{\lambda}{2} \right) \partial_{\nu[1]} \left[ \text{tr} \left( V^{[1]} V^{[1]^T} \right) \right] \\
= \left( \frac{2}{\nu} \right) \left( \left( A^{[2]} - C \right) \circ g_2' (Z^{[2]}) \right)^T \partial_{\nu[1]} \left[ U^{[2]} A^{[1]} V^{[2]^T} \right] + \left( \frac{\lambda}{2} \right) V^{[1]} \\
= \left( \frac{2}{\nu} \right) \left( \left( A^{[2]} - C \right) \circ g_2' (Z^{[2]}) \right)^T \partial_{\nu[1]} \left[ U^{[2]} g_1 \left( U^{[1]} X V^{[1]^T} \right) V^{[2]^T} \right] + \lambda V^{[1]} \\
= \left( \frac{2}{\nu} \right) V^{[1]^T} \left( \left( A^{[2]} - C \right) \circ g_2' (Z^{[2]}) \right)^T U^{[2]} \partial_{\nu[1]} \left[ g_1 \left( U^{[1]} X V^{[1]^T} \right) \right] + \lambda V^{[1]} \\
= \left( \frac{2}{\nu} \right) V^{[1]^T} \left( \left( A^{[2]} - C \right) \circ g_2' (Z^{[2]}) \right)^T U^{[2]} \circ g_1' (Z^{[1]^T}) \partial_{\nu[1]} \left[ U^{[1]} X V^{[1]^T} \right] + \lambda V^{[1]} \\
= \left( \frac{2}{\nu} \right) V^{[1]^T} \left( \left( A^{[2]} - C \right) \circ g_2' (Z^{[2]}) \right)^T U^{[2]} \circ g_1' (Z^{[1]^T}) U^{[1]} X + \lambda V^{[1]} \\
= V^{[1]^T} \left[ \frac{2}{\nu} \right] \left( \left( A^{[2]} - C \right) \circ g_2' (Z^{[2]}) \right)^T U^{[2]} \circ g_1' (Z^{[1]^T}) U^{[1]} X + \lambda V^{[1]} \\
= V^{[1]^T} \delta_{\nu[1]} (Z^{[1]^T}) U^{[1]} X + \lambda V^{[1]} \\
= \delta_{\nu[1]} (X) + \lambda V^{[1]} \]
Let

\[ J_2 = R^2 + \frac{1}{\nu} \left( \left\| \phi(X_\ast; W) - c \right\|^2 - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{3} \left( \| U^{[\ell]} \|_{F}^2 + \| V^{[\ell]} \|_{F}^2 \right) \]

Proof of Proposition 5.0.2

\[ \partial_R (J_2) = \partial_R \left[ R^2 + \frac{1}{\nu} \left( \left\| \phi(X_\ast; W) - C \right\|^2 - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{3} \left( \| U^{[\ell]} \|_{F}^2 + \| V^{[\ell]} \|_{F}^2 \right) \right] \]

\[ = \partial_R \left[ R^2 - \frac{1}{\nu} R^2 \right] \]

\[ = 2R - \left( \frac{2R}{\nu} \right) \]

\[ = 2R \left( 1 - \frac{1}{\nu} \right) \]

\[ \partial_{U^{[3]}} (J_2) = \partial_{U^{[3]}} \left[ R^2 + \frac{1}{\nu} \left( \left\| \phi(X_\ast; W) - C \right\|^2 - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{3} \left( \| U^{[\ell]} \|_{F}^2 + \| V^{[\ell]} \|_{F}^2 \right) \right] \]

\[ = \partial_{U^{[3]}} \left[ \frac{1}{\nu} \left( \left\| \phi(X_\ast; W) - C \right\|^2 - R^2 \right) + \left( \frac{\lambda}{2} \right) \left( \| U^{[1]} \|_{F}^2 + \| V^{[1]} \|_{F}^2 + \ldots + \| U^{[3]} \|_{F}^2 + \| V^{[3]} \|_{F}^2 \right) \right] \]

\[ = \partial_{U^{[3]}} \left[ \frac{1}{\nu} \| A^{[3]} - C \|_{F}^2 \right] + \left( \frac{\lambda}{2} \right) \partial_{U^{[3]}} \left[ \| U^{[3]} \|_{F}^2 \right] \]

\[ = \left( \frac{2}{\nu} \right) \left( A^{[3]} - C \right) \partial_{U^{[3]}} \left[ g_3 \left( Z^{[3]} \right) - C \right] + \left( \frac{\lambda}{2} \right) \partial_{U^{[3]}} \left[ tr \left( U^{[3]} A^{[3]T} \right) \right] \]

\[ = \left( \frac{2}{\nu} \right) \left( A^{[3]} - C \right) \circ g_3' \left( Z^{[3]} \right) \partial_{U^{[3]}} \left[ U^{[3]} A^{[3]T} \right] + \left( \frac{2\lambda}{2} \right) U^{[3]} \]

\[ = \left( \frac{2}{\nu} \right) \left( A^{[3]} - C \right) \circ g_3' \left( Z^{[3]} \right) V^{[3]} A^{[3]T} + \lambda U^{[3]} \]

\[ = \delta^{[3]} A^{[3]T} + \lambda U^{[3]} \]
\[ \partial_{\nu[3]}(J_2) = \partial_{\nu[3]} \left[ R^2 + \frac{1}{\nu} \left( ||\phi(X_\nu; W) - C||^2_F - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{3} \left( ||U[\ell]||^2_F + ||V[\ell]||^2_F \right) \right] \\
= \partial_{\nu[3]} \left[ \frac{1}{\nu} \left( ||\phi(X_\nu; W) - C||^2_F - R^2 \right) + \left( \frac{\lambda}{2} \right) \left( ||U[1]||^2_F + ||V[1]||^2_F + \ldots + ||U[3]||^2_F + ||V[3]||^2_F \right) \right] \\
= \partial_{\nu[3]} \left[ \frac{1}{\nu} ||A[3] - C||^2_F + \left( \frac{\lambda}{2} \right) \partial_{\nu[3]} \left[ ||V[3]||^2_F \right] \right] \\
= \left( \frac{2}{\nu} \right) \left( A[3] - C \right)^T \partial_{\nu[3]} \left[ g_3(Z[3]) - C \right] + \left( \frac{\lambda}{2} \right) \partial_{\nu[3]} \left[ tr \left( V[3]^T V[3] \right) \right] \\
= \left( \frac{2}{\nu} \right) \left( A[3] - C \right) \circ g_3(Z[3]) \partial_{\nu[3]} \left[ U[3]^T A[3]^T V[3]^T \right] + \left( \frac{2\lambda}{2} \right) \left( V[3] \right) } \\

\[ \partial_{U[2]}(J_2) = \partial_{U[2]} \left[ R^2 + \frac{1}{\nu} \left( ||\phi(X_\nu; W) - C||^2_F - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{3} \left( ||U[\ell]||^2_F + ||V[\ell]||^2_F \right) \right] \\
= \partial_{U[2]} \left[ \frac{1}{\nu} \left( ||\phi(X_\nu; W) - C||^2_F - R^2 \right) + \left( \frac{\lambda}{2} \right) \left( ||U[1]||^2_F + ||V[1]||^2_F + \ldots + ||U[3]||^2_F + ||V[3]||^2_F \right) \right] \\
= \partial_{U[2]} \left[ \frac{1}{\nu} ||A[3] - C||^2_F + \left( \frac{\lambda}{2} \right) \partial_{U[2]} \left[ ||U[2]||^2_F \right] \right] \\
= \left( \frac{2}{\nu} \right) \left( A[3] - C \right) \partial_{U[2]} \left[ g_3(Z[3]) - C \right] + \left( \frac{\lambda}{2} \right) \partial_{U[2]} \left[ tr \left( U[2]^T U[2]^T \right) \right] \\
= \left( \frac{2}{\nu} \right) \left( A[3] - C \right) \circ g_3(Z[3]) \partial_{U[2]} \left[ U[3]^T A[3]^T V[3]^T \right] + \left( \frac{2\lambda}{2} \right) U[2] \\
= \left( \frac{2}{\nu} \right) \left( A[3] - C \right) \circ g_3(Z[3]) \partial_{U[2]} \left[ U[3]^T g_2(Z[2]) V[3]^T \right] + \lambda U[2] \\
= \left( \frac{2}{\nu} \right) \left( A[3] - C \right) \circ g_3(Z[3]) \partial_{U[2]} \left[ U[3]^T g_2(Z[2]) V[3]^T \right] + \lambda U[2] \\
\[
\partial_{\mathcal{V}[2]}(J_2) = \partial_{\mathcal{V}[2]} \left[ R^2 + \frac{1}{\nu} \left( \|\phi(X_\nu;\mathcal{W}) - \mathcal{C} \|_F^2 - R^2 \right) + \frac{\lambda}{2} \sum_{\ell=1}^{3} \left( \|U^{[\ell]}\|_F^2 + \|V^{[\ell]}\|_F^2 \right) \right]
\]
\[
= \partial_{\mathcal{V}[2]} \left[ \frac{1}{\nu} \left( \|\phi(X_\nu;\mathcal{W}) - \mathcal{C} \|_F^2 - R^2 \right) + \left( \frac{\lambda}{2} \right) \left( \|U^{[\nu]}\|_F^2 + \|V^{[\nu]}\|_F^2 + \ldots + \|U^{[3]}\|_F^2 + \|V^{[3]}\|_F^2 \right) \right]
\]
\[
= \partial_{\mathcal{V}[2]} \left[ \frac{1}{\nu} \|A^{[\nu]} - \mathcal{C}\|_F^2 \right] + \left( \frac{\lambda}{2} \right) \partial_{\mathcal{V}[2]} \left[ \|V^{[2]}\|_F^2 \right]
\]
\[
= \left( \frac{2}{\nu} \right) \left( A^{[\nu]} - \mathcal{C} \right)^T \partial_{\mathcal{V}[2]} \left[ g_3 \left( Z^{[\nu]} \right) - \mathcal{C} \right] + \left( \frac{\lambda}{2} \right) \partial_{\mathcal{V}[2]} \left[ \text{tr} \left( V^{[2]} V^{[2]^T} \right) \right]
\]
\[
= \left( \frac{2}{\nu} \right) \left( \left( A^{[\nu]} - \mathcal{C} \right) \circ g'_3 \left( Z^{[\nu]} \right) \right)^T \partial_{\mathcal{V}[2]} \left[ U^{[3]} A^{[2]} V^{[3]^T} \right] + \left( \frac{2\lambda}{2} \right) V^{[2]}
\]
\[
= \left( \frac{2}{\nu} \right) \left( \left( A^{[\nu]} - \mathcal{C} \right) \circ g'_3 \left( Z^{[\nu]} \right) \right)^T \partial_{\mathcal{V}[2]} \left[ U^{[3]} g_2 \left( Z^{[2]} \right) V^{[3]^T} \right] + \lambda V^{[2]}
\]
\[
= \left( \frac{2}{\nu} \right) V^{[3]^T} \left( \left( A^{[\nu]} - \mathcal{C} \right) \circ g'_3 \left( Z^{[\nu]} \right) \right)^T \partial_{\mathcal{V}[2]} \left[ g_2 \left( U^{[2]} A^{[1]} V^{[2]^T} \right) \right] + \lambda V^{[2]}
\]
\[
= V^{[3]^T} \left[ \left( \frac{2}{\nu} \right) \left( \left( A^{[\nu]} - \mathcal{C} \right) \circ g'_3 \left( Z^{[\nu]} \right) \right)^T \partial_{\mathcal{V}[2]} \left[ U^{[2]} A^{[1]} V^{[2]^T} \right] \right] + \lambda V^{[2]}
\]
\[
= \delta^{[2]}_\nu A^{[1]} + \lambda V^{[2]}
\]
\[\partial_{U^{[1]}}(J_2) = \partial_{U^{[1]}} \left[ \frac{1}{2} \| \phi (X, \omega) - C \|^2_F - R^2 \right] + \frac{1}{2} \sum_{i=1}^{3} \left( \| U^{[i]} \|^2_F + \| V^{[i]} \|^2_F \right) \]

\[= \partial_{U^{[1]}} \left[ \frac{1}{2} \| \phi (X, \omega) - C \|^2_F - R^2 \right] + \frac{1}{2} \sum_{i=1}^{3} \left( \| U^{[i]} \|^2_F + \| V^{[i]} \|^2_F \right) \]

\[= \partial_{U^{[1]}} \left[ \frac{1}{2} \| A^{[3]} - C \|^2_F \right] + \left( \frac{\lambda}{2} \right) \partial_{U^{[1]}} \left[ \| U^{[1]} \|^2_F \right] \]

\[= \frac{2}{\nu} \left( \frac{1}{2} \right) \left( A^{[3]} - C \right) \partial_{U^{[1]}} \left[ g_3 (Z^{[3]} - C) \right] + \left( \frac{\lambda}{2} \right) \partial_{U^{[1]}} \left[ \text{tr} \left( U^{[1]} U^{[1]^T} \right) \right] \]

\[= \frac{2}{\nu} \left( \frac{1}{2} \right) \left( A^{[3]} - C \right) \circ g_3 (Z^{[3]}) \partial_{U^{[1]}} \left[ U^{[3]} A^{[3]} V^{[3]^T} \right] + \left( \frac{2\lambda}{2} \right) U^{[1]} \]

\[= \frac{2}{\nu} \left( \frac{1}{2} \right) \left( A^{[3]} - C \right) \circ g_3 (Z^{[3]}) \partial_{U^{[1]}} \left[ U^{[3]} g_2 (Z^{[2]}) V^{[3]^T} \right] + \lambda U^{[1]} \]

\[= \left( \frac{2}{\nu} \right) \left( \frac{1}{2} \right) \left( A^{[3]} - C \right) \circ g_3 (Z^{[3]}) \partial_{U^{[1]}} \left[ U^{[3]} g_2 (U^{[2]} A^{[1]} V^{[2]^T}) V^{[3]^T} \right] + \lambda U^{[1]} \]

\[= U^{[3]^T} \left( \frac{2}{\nu} \right) \left( \frac{1}{2} \right) \left( A^{[3]} - C \right) \circ g_3 (Z^{[3]}) \partial_{U^{[1]}} \left[ U^{[3]} g_2 (U^{[2]} A^{[1]} V^{[2]^T}) \right] + \lambda U^{[1]} \]

\[= U^{[3]^T} \partial_{U^{[1]}} \circ g_2 (Z^{[2]}) \partial_{U^{[1]}} \left[ U^{[2]} A^{[1]} V^{[2]^T} \right] + \lambda U^{[1]} \]

\[= U^{[2]^T} U^{[3]^T} \partial_{U^{[1]}} \circ g_2 (Z^{[2]}) \partial_{U^{[1]}} \left[ U^{[3]} g_1 (Z^{[1]}) \right] + \lambda U^{[1]} \]

\[= U^{[2]^T} \left[ U^{[3]^T} \partial_{U^{[1]}} \circ g_2 (Z^{[2]}) V^{[2]^T} \right] \circ g_1 (Z^{[1]}) \partial_{U^{[1]}} \left[ U^{[1]} X, V^{[1]^T} \right] + \lambda U^{[1]} \]

\[= U^{[2]^T} \partial_{U^{[1]}} \circ g_2 (Z^{[2]}) V^{[2]^T} \circ g_1 (Z^{[1]}) V^{[1]^T} \partial_{U^{[1]}} \left[ U^{[1]} X, V^{[1]^T} \right] + \lambda U^{[1]} \]

\[= \delta_{U^{[1]}} \left[ U^{[1]} X^T + \lambda U^{[1]} \right] \]

\[= \delta_{U^{[1]}} \left[ X^T + \lambda U^{[1]} \right] \]
\[ \partial_{V'[1]}(J_2) = \partial_{V'[1]} \left[ R^2 + \frac{1}{\nu} (\| \phi (X_*; W) - C \|^2_{\mathbb{F}} - R^2) + \frac{1}{2} \sum_{\ell=1}^{3} \left( \| U^{[\ell]} \|^2_{\mathbb{F}} + \| V^{[\ell]} \|^2_{\mathbb{F}} \right) \right] \\
= \partial_{V'[1]} \left[ \frac{1}{\nu} (\| \phi (X_*; W) - C \|^2_{\mathbb{F}} - R^2) + \left( \frac{\lambda}{2} \right) \left( \| U^{[1]} \|^2_{\mathbb{F}} + \| V^{[1]} \|^2_{\mathbb{F}} + \ldots + \| U^{[3]} \|^2_{\mathbb{F}} + \| V^{[3]} \|^2_{\mathbb{F}} \right) \right] \\
= \partial_{V'[1]} \left[ \frac{1}{\nu} \| A^{[3]} - C \|^2_{\mathbb{F}} + \left( \frac{\lambda}{2} \right) \| V^{[1]} \|^2_{\mathbb{F}} \right] \\
= \left( \frac{2}{\nu} \right) \left( A^{[3]} - C \right)^T \partial_{V'[1]} \left[ g_3 \left( Z^{[3]} \right) - C \right] + \left( \frac{\lambda}{2} \right) \partial_{V'[1]} \left[ tr \left( V^{[1]} V^{[1]T} \right) \right] \\
= \left( \frac{2}{\nu} \right) \left( \left( A^{[3]} - C \right) \circ g_3 \left( Z^{[3]} \right) \right)^T \partial_{V'[1]} \left[ U^{[3]} A^{[2]} V^{[2]T} \right] + \left( \frac{\lambda}{2} \right) V^{[1]} \\
= \left( \frac{2}{\nu} \right) \left( \left( A^{[3]} - C \right) \circ g_3 \left( Z^{[3]} \right) \right)^T \partial_{V'[1]} \left[ U^{[3]} g_2 \left( U^{[2]} A^{[1]} V^{[2]T} \right) V^{[3]} \right] + \lambda V^{[1]} \\
= V^{[1]T} \left[ \left( \frac{2}{\nu} \right) \left( \left( A^{[3]} - C \right) \circ g_3 \left( Z^{[3]} \right) \right)^T \right] U^{[3]} \partial_{V'[1]} \left[ g_2 \left( U^{[2]} A^{[1]} V^{[2]T} \right) \right] + \lambda V^{[1]} \\
= V^{[1]T} \delta^{[3]} \circ g_2 \left( Z^{[2]} \right) \partial_{V'[1]} \left[ U^{[2]} A^{[1]} V^{[2]T} \right] + \lambda V^{[1]} \\
= V^{[2]T} \left[ V^{[3]} T \delta^{[3]} \circ g_2 \left( Z^{[2]} \right) U^{[2]} \right] \partial_{V'[1]} \left[ g_1 \left( U^{[1]} X^{[1]T} \right) \right] + \lambda V^{[1]} \\
= V^{[2]T} \left[ V^{[3]} T \delta^{[3]} \circ g_2 \left( Z^{[2]} \right) U^{[2]} \right] \partial_{V'[1]} \left[ g_1 \left( U^{[1]} X^{[1]T} \right) \right] + \lambda V^{[1]} \\
= V^{[2]T} \delta^{[3]} \circ g_2 \left( Z^{[2]} \right) \partial_{V'[1]} \left[ g_1 \left( U^{[1]} X^{[1]T} \right) \right] + \lambda V^{[1]} \\
= V^{[2]T} \delta^{[2]} \circ g_1 \left( Z^{[1]} \right) \partial_{V'[1]} \left[ U^{[1]} X^{[1]T} \right] + \lambda V^{[1]} \\
= V^{[2]T} \delta^{[2]} \circ g_1 \left( Z^{[1]} \right) \partial_{V'[1]} \left[ U^{[1]} X^{[1]T} \right] + \lambda V^{[1]} \\
= V^{[2]T} \delta^{[2]} \circ g_1 \left( Z^{[1]} \right) U^{[1]} X^{[1]T} + \lambda V^{[1]} \\
= \delta^{[1]} X^{[1]T} + \lambda V^{[1]} \]
APPENDIX F: OC-DSMDD SUBGRADIENT DERIVATIONS
Let

\[
J = \frac{1}{2} ||\phi(X_*, \mathcal{U}, \mathcal{V}) - C||_F^2 + \frac{\lambda}{2} \sum_{l=1}^2 (||U^{(l)}||_F^2 + ||V^{(l)}||_F^2)
\]

Proof of Proposition 5.0.3

\[
\partial_{U^{[2]}} (J) = \partial_{U^{[2]}} \left[ \frac{1}{2} ||\phi(X_*, \mathcal{U}, \mathcal{V}) - C||_F^2 + \frac{\lambda}{2} \sum_{l=1}^2 (||U^{(l)}||_F^2 + ||V^{(l)}||_F^2) \right]
\]

\[
= \partial_{U^{[2]}} \left[ \frac{1}{2} ||\phi(X_*, \mathcal{U}, \mathcal{V}) - C||_F^2 + \left( \frac{\lambda}{2} \right) ||U^{[2]}||_F^2 \right]
\]

\[
= \partial_{U^{[2]}} \left[ \frac{1}{2} ||A^{[2]} - C||_F^2 + \left( \frac{\lambda}{2} \right) ||U^{[2]}||_F^2 \right]
\]

\[
= \left( \frac{2}{2} \right) (A^{[2]} - C) \partial_{U^{[2]}} [ \phi^{(2)}(Z^{[2]}) - C ] + \left( \frac{2\lambda}{2} \right) U^{[2]}
\]

\[
= (A^{[2]} - C) \circ g_1(Z^{[2]}) U^{[2]} A^{[1]} + \lambda U^{[2]}
\]

\[
= \delta_u^{[2]} A^{[1]} + \lambda U^{[2]}
\]

\[
\partial_{V^{[2]}} (J) = \partial_{V^{[2]}} \left[ \frac{1}{2} ||\phi(X_*, \mathcal{U}, \mathcal{V}) - C||_F^2 + \frac{\lambda}{2} \sum_{l=1}^2 (||U^{(l)}||_F^2 + ||V^{(l)}||_F^2) \right]
\]

\[
= \partial_{V^{[2]}} \left[ \frac{1}{2} ||\phi(X_*, \mathcal{U}, \mathcal{V}) - C||_F^2 + \left( \frac{\lambda}{2} \right) ||V^{[2]}||_F^2 \right]
\]

\[
= \partial_{V^{[2]}} \left[ \frac{1}{2} ||A^{[2]} - C||_F^2 + \left( \frac{\lambda}{2} \right) ||V^{[2]}||_F^2 \right]
\]

\[
= \left( \frac{2}{2} \right) (A^{[2]} - C)^T \partial_{V^{[2]}} [ \phi^{(2)}(Z^{[2]}) - C ] + \left( \frac{2\lambda}{2} \right) V^{[2]}
\]

\[
= \left( \left( A^{[2]} - C \right) \circ g_2(Z^{[2]}) \right)^T \partial_{V^{[2]}} \left( U^{[2]} A^{[1]} V^{[2]} \right) + \lambda V^{[2]}
\]

\[
= \left( \left( A^{[2]} - C \right) \circ g_2(Z^{[2]}) \right)^T \left( U^{[2]} A^{[1]} + \lambda V^{[2]} \right)
\]

\[
= \delta_u^{[2]} A^{[1]} + \lambda V^{[2]}
\]
\[ \partial_{U^{(2)}}(J) = \partial_{U^{(2)}} \left[ \frac{1}{2} \| \phi(X_\ast; U, V) - C \|_2^2 + \frac{\lambda}{2} \sum_{\ell=1}^2 \left( \| U^{[\ell]} \|_2^2 + \| V^{[\ell]} \|_2^2 \right) \right] \\
= \partial_{U^{(2)}} \left[ \frac{1}{2} \| \phi(X_\ast; U, V) - C \|_2^2 + \left( \frac{\lambda}{2} \right) \| U^{[1]} \|_2^2 \right] \\
= \partial_{U^{(2)}} \left[ \frac{1}{2} \| A^{[2]} - C \|_2^2 + \left( \frac{\lambda}{2} \right) \| U^{[1]} \|_2^2 \right] \\
= \left( \frac{2}{2} \right) \left( A^{[2]} - C \right) \partial_{U^{(1)}} \left[ g_2(Z^{[2]}) - C \right] + \left( \frac{2\lambda}{2} \right) U^{[1]} \\
= \left[ \left( A^{[2]} - C \right) \circ g'_2 \left( Z^{[2]} \right) \right] \frac{d}{d U^{[1]}} \left[ U^{[2]} A^{[1]} V^{[2]^T} \right] + \lambda U^{[1]} \\
= \left[ \left( A^{[2]} - C \right) \circ g'_2 \left( Z^{[2]} \right) \right] \frac{d}{d U^{[1]}} \left[ U^{[2]} g_1 \left( U^{[1]} X V^{[1]^T} \right) V^{[2]^T} \right] + \lambda U^{[1]} \\
= U^{[2]^T} \left[ \left( A^{[2]} - C \right) \circ g'_2 \left( Z^{[2]} \right) \right] V^{[2]} \frac{d}{d U^{[1]}} \left[ g_1 \left( U^{[1]} X V^{[1]^T} \right) \right] + \lambda U^{[1]} \\
= U^{[2]^T} \left[ \left( A^{[2]} - C \right) \circ g'_2 \left( Z^{[2]} \right) \right] V^{[2]} \circ g'_1 \left( Z^{[1]} \right) \frac{d}{d U^{[1]}} \left[ U^{[1]} X V^{[1]^T} \right] + \lambda U^{[1]} \\
= \left[ U^{[2]^T} \delta^{[2]} \circ g'_1 \left( Z^{[1]} \right) \right] V^{[1]^T} X^T + \lambda U^{[1]} \\
= \delta^{[1]} X^T + \lambda U^{[1]} \]
\[ \partial_{V[1]}(J) = \partial_{V[1]} \left[ \frac{1}{2} \| \phi(X_\ast; U, V) - C \|_F^2 + \frac{\lambda}{2} \sum_{\ell=1}^2 \left( \| U[\ell] \|_F^2 + \| V[\ell] \|_F^2 \right) \right] \\
= \partial_{V[1]} \left[ \frac{1}{2} \| \phi(X_\ast; U, V) - C \|_F^2 + \left( \frac{\lambda}{2} \right) \| V[1] \|_F^2 \right] \\
= \partial_{V[1]} \left[ \frac{1}{2} \| A[2] - C \|_F^2 + \left( \frac{\lambda}{2} \right) \| V[1] \|_F^2 \right] \\
= \left( \frac{2}{\lambda} \right) \left( A[2] - C \right)^T \partial_{V[1]} \left[ g_2 \left( Z[2] \right) - C \right] + \left( \frac{2\lambda}{2} \right) V[1] \\
= \left( \left( A[2] - C \right) \circ g_2 \left( Z[2] \right) \right)^T \partial_{V[1]} \left[ U[2] g_1 \left( U[1] X V[1]^T \right) V[2]^T \right] + \lambda V[1] \\
= V[2]^T \left( \left( A[2] - C \right) \circ g_2 \left( Z[2] \right) \right)^T \partial_{V[1]} \left[ g_1 \left( U[1] X V[1]^T \right) \right] + \lambda V[1] \\
= \left( \left( A[2] - C \right) \circ g_2 \left( Z[2] \right) \right)^T \partial_{V[1]} \left[ g_1 \left( U[1] X V[1]^T \right) \right] + \lambda V[1] \\
= \left( \left( A[2] - C \right) \circ g_2 \left( Z[2] \right) \right)^T \partial_{V[1]} \left[ g_1 \left( U[1] X V[1]^T \right) \right] + \lambda V[1] \\
= \left( \left( A[2] - C \right) \circ g_2 \left( Z[2] \right) \right)^T \partial_{V[1]} \left[ g_1 \left( U[1] X V[1]^T \right) \right] + \lambda V[1] \\
= \delta_{V}^{[1]} X + \lambda V[1] \]
LIST OF REFERENCES


[8] Y. Cao, Y. Li, S. Coleman, A. Belatreche, and T. M. McGinnity. Detecting price manipulation


