TOPICS IN OPTIMIZATION VIA DEEP NEURAL NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

By
Ömer Ekmekcioğlu
June 2022

Topics in Optimization via Deep Neural Networks
By Ömer Ekmekcioğlu
June 2022

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Mustafa Çelebi Pınar(Advisor)

Taghi Khaniyev

Cem İyigün

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

TOPICS IN OPTIMIZATION VIA DEEP NEURAL NETWORKS

Ömer Ekmekcioğlu
M.S. in Industrial Engineering
Advisor: Mustafa Çelebi Pınar
June 2022

We present two studies in the intersection of deep learning and optimization, Deep Portfolio Optimization, and Subset Based Error Recovery. Along with the emergence of deep models in finance, the portfolio optimization trend had shifted towards data-driven models from the classical model-based approaches. However, the deep portfolio models generally suffer from the non-stationary nature of the data and the results obtained are not always very stable. To address this issue, we propose to use Graph Neural Networks (GNN) which allows us to incorporate graphical knowledge to increase the stability of the models in order to improve the results obtained in comparison to the state-of-the-art recurrent architectures. Furthermore, we analyze the algorithmic risk-return trade-off for the deep portfolio optimization models to give insights on risk for the fully data-driven models.

We also propose a data denoising method using Extreme Learning Machine (ELM) structure. Furthermore, we show that the method is equivalent to a robust two-layer ELM that implicitly benefits from the proposed denoising algorithm. Current robust ELM methods in the literature involve well-studied L1, L2 regularization techniques as well as the usage of the robust loss functions such as Huber Loss. We extend the recent analysis on the Robust Regression literature to be effectively used in more general, non-linear settings and to be compatible with any ML algorithm such as Neural Networks (NN). These methods are useful under the scenario where the observations suffer from the effect of heavy noise. Tests for denoising and regularized ELM methods are conducted on both synthetic and real data. Our method performs better than its competitors for most of the scenarios, and successfully eliminates most of the noise.

Keywords: Optimization, Machine Learning, Deep Learning.

ÖZET

DERİN SİNİR AĞLARI ÜZERİNDEN ENİYİLEME KONULARI

Ömer Ekmekcioğlu
Endüstri Mühendisliği, Yüksek Lisans
Tez Danışmanı: Mustafa Çelebi Pınar
Haziran 2022

Derin öğrenme ve optimizasyonun kesiştiği, Derin Portföy Optimizasyonu ve Alt Küme Tabanlı Hata Önleme çalışmalarına odaklanıyoruz. Finansta derin modellerin ortaya çıkmasıyla birlikte portföy optimizasyon literatürü, klasikleşmiş model temelli yaklaşımlardan, veriye dayalı modellere doğru kaymıştır. Bununla birlikte, derin portföy modelleri genellikle verilerin durağan olmayan yapısı nedeniyle sorun yaşamaktadır ve elde edilen sonuçlar her zaman istikrarlı değildir. Tekrarlayan Sinir Ağları teknolojisine kıyasla elde edilen sonuçları iyileştirmek ve modellerin kararlılığını artırmak için verideki grafiksel bilgiden yararlanmamıza izin veren Grafik Sinir Ağlarını kullanmayı öneriyoruz. Ek olarak, tamamen veriye dayalı modellerde risk algısı hakkında fikir vermek adına derin portföy optimizasyon modelleri arasında algoritmik risk-getiri dengesini analiz ediyoruz.

İkinci olarak, Aşırı Öğrenme Makinesi (AÖM) yapısını kullanan bir gürültü giderme yöntemi öneriyoruz. Ayrıca, önerilen yöntemin, iki katmanlı bir gürbüz AÖM'ye eşdeğer olduğunu gösteriyoruz. Literatürdeki mevcut gürbüz AÖM yöntemleri, iyi bilinen L1, L2 düzenleştirme tekniklerinin yanı sıra Huber Kaybı gibi gürbüz kayıp fonksiyonlarının kullanımını içerir. Böylece AÖM'nin kullanımını, Makine Öğrenme algoritmasından bağımsız olarak genel bir veri temizleme yöntemi haline getiriyoruz. Sunduğumuz yöntemler, gözlemlerin yoğun gürültünün etkisinden muzdarip olduğu senaryolarda kullanışlıdır. Gürültü giderme ve düzenleştirilmiş AÖM yöntemleri için yapılan testleri hem sentetik hem de gerçek veriler üzerinde gerçekleştiriyoruz. Metodumuz, senaryoların çoğunda rakiplerinden daha iyi performans gösterir ve gürültünün çoğunu başarıyla ortadan kaldırır.

Anahtar sözcükler: Eniyileme, Makine Öğrenimi, Derin Öğrenme.

Acknowledgement

I want to express my gratitude to my advisor Mustafa Pınar. His sincerity and mentorship beyond academic matters meant so much to me during the last three years.

Among the endless list of friends who continuously supported me, I have special thanks to Deniz Akkaya, Mert Albaba, Emre Mumcuğlu, Efsun Kavaklıoğlu, Buse Şen, Unay Dorken, Burak Altıntaş and Cem Kesici, Tolunay Alankaya and Fatih Selim Aktaş.

I'm grateful to my family for all the support, love, and care. I devote this thesis to my father, who guided me my whole life but never saw the ending to this journey.

Contents

1 Introduction **1**

2 Background **3**

2.1 Subset Based Error Recovery 3

2.1.1 Sparse Recovery 3

2.1.2 Non-Linear Robust Model Description 4

2.1.3 ELM Model Description 5

2.1.4 Robust Methods and Related Loss Functions 6

2.1.5 Convex Neural Networks 7

2.2 Deep Portfolio Optimization 9

2.2.1 Neural Networks and Time Series 9

2.2.2 Portfolio Optimization Problem 12

3 Deep Portfolio Optimization **14**

3.1 Contribution 16

3.2 Our Method 16

3.3 Architecture 18

3.4 Results 20

3.5 Remarks 24

3.6 Conclusion 25

4 Subset Based Error Recovery **34**

4.1 Introduction 34

4.2 Contribution 35

4.3 Algorithm 36

4.4 Theoretical Analysis 39

4.5 Results 43

4.5.1 Data Denoising 43

4.5.2 ELM Method 46

4.6 Conclusion 49

5 Conclusion **54**

List of Figures

3.1 Deep Learning Architectures for Portfolio Optimization 18

3.2 Proposed GNN Architecture 19

3.3 GAT Returns with Std Error on Dataset1 28

3.4 LSTM Returns with Std Error on Dataset1 29

3.5 GAT Returns with Max Error on Dataset1 29

3.6 LSTM Returns with Max Error on Dataset1 30

3.7 GAT Returns with Std Error on Dataset2 30

3.8 LSTM Returns with Std Error on Dataset2 31

3.9 GAT Returns with Std Error on Dataset3 31

3.10 LSTM Returns with Std Error on Dataset3 32

3.11 GAT Returns with Std Error on Dataset4 32

3.12 LSTM Returns with Std Error on Dataset4 33

4.1 Relative Err Lin 48

4.2 MSE Lin 49

4.3 Corr Lin 50

4.4 err found lin 51

4.5 relative err nonlin 51

4.6 MSE Nonlin 52

4.7 corr nonlin 52

4.8 err found nonlin 53

List of Tables

3.1 Dataset Structure 20

3.2 Evaluation of Total Returns on Datasets 21

3.3 Evaluation of Sharpe Ratios on Datasets 21

3.4 Return Statistics on Every Instance for Dataset 1 22

3.5 Return Statistics on Every Instance for Dataset 1 23

3.6 Return Statistics on Every Instance for Dataset 2 24

3.7 Return Statistics on Every Instance for Dataset 2 25

3.8 Return Statistics on Every Instance for Dataset 3 26

3.9 Return Statistics on Every Instance for Dataset 3 27

3.10 Return Statistics on Every Instance for Dataset 4 27

3.11 Return Statistics on Every Instance for Dataset 4 28

4.1 NN Result Part 1 44

4.2 NN Results Part 2 44

4.3 Boston Price Dataset Results 45

4.4 Diabetes Dataset Results 45

4.5 ELM Results 46

4.6 ELM Results 46

4.7 ELM Results 47

4.8 ELM Results 47

Chapter 1

Introduction

Recently, the success of deep learning methods in various fields, especially the fields of computer vision and natural language processing prepared a basis for deep learning tools that could be applied to various problems. Inspired by these advancements, we aim to merge ideas from mathematical optimization and deep learning to tackle traditional optimization problems while contributing to the deep learning methods. This thesis presents two major studies, Deep Portfolio Optimization and Subset Based Error Recovery. Even though these studies are not directly linked with each other, they both show ideas on how classical optimization ideas could be integrated into deep learning literature to improve the state-of-the-art. In the first study, we will be delving into the portfolio optimization literature and utilizing deep learning frameworks to obtain fully data-driven solutions. We will be utilizing deep learning methods to solve a problem originally designed for model-based optimization frameworks. Following that, we will focus on a sparse optimization tool, which we will be utilizing in deep learning models. This study will aim to benefit deep learning methods by following the ideas from sparse optimization literature.

In the first study, we delve into the realm of Portfolio Optimization literature, which is a well-known problem that has been studied extensively since the 1960s

[1]. Starting with the Markowitz Model, many optimization models were proposed to deal with different objectives. Seeing that, deep models improved the time series prediction task quite significantly, paving the way for improvements in financial studies [2], we decided to further utilize this tool in portfolio optimization similar to the studies [3]. We investigate various models, point out a few challenges and propose a graph neural network solution to solve the data-driven portfolio optimization in an end-to-end manner.

In the second study, we approach the sparse recovery problem from a different perspective and focus on data denoising. Modern datasets are generally huge and it is common to have faulty or corrupted entries among the observations. This may cause stability and robustness issues in many applications. Having a robust deep learning model is important to mitigate issues from these corrupted observations. Being inspired by the developments in robust regression literature [4], we study robust non-linear machine learning techniques to extend their results to non-linear problems. Sparse regression problem is heavily studied as a method that selects features of a given data matrix. We focus on a slightly different version of this problem where the goal is to select corrupted observations. Integrating ideas from the sparse recovery literature to extreme learning machines and deep learning methods allows us to study methods with theoretical guarantees.

The rest of this thesis includes an extensive background section that covers the fundamental ideas used in the studies. We then present our work done in the Deep Portfolio Optimization and Subset Based Error Recovery studies.

Chapter 2

Background

In this section, we cover a wide range of topics regarding the relevant topics covered in both of our studies. Therefore, we present background section in two sub-sections, explaining the preliminaries for each work in their respective section.

2.1 Subset Based Error Recovery

2.1.1 Sparse Recovery

The Literature on sparse recovery mainly focuses on the following problem

$$\min \|y - Xw\|_2^2 \tag{2.1}$$

$$\text{s.t. } \|w\|_0 \leq k, \tag{2.2}$$

where $X \in \mathbb{R}^{n \times p}$ represents the data matrix, $y \in \mathbb{R}^n$ contain the observations, and $w \in \mathbb{R}^p$ are the unknown coefficients to be estimated. The notation $\|w\|_0$ denotes the ℓ_0-norm of w which counts the number of non-zero elements of w. The number of non-zero elements (the cardinality of the support of w) is restricted to be at most k.

Due to the ℓ_0 constraint, the problem is NP-Hard [5, 6]. In the literature, various solution techniques have been proposed, ranging from convex relaxations of the problem to heuristic algorithms to handle the cardinality constraint. Some of the previously proposed and prominent solution methods are Fista [7] and Iterative Hard Thresholding (IHT) [8].

In the present study of Subset Based Error Recovery, the IHT algorithm is used as one of the main building blocks of our algorithm. However, to clean the data from corrupting errors, the problem will be cast as selecting sparse corrupted observations from the data instead of finding a sparse regression solution. Instead of finding a sparse set of features, we find a sparse set of corrupted observations, which allows us to utilize this problem as a data-denoising tool [4].

2.1.2 Non-Linear Robust Model Description

The model of this study is a non-linear one where the observations are heavily corrupted by a noise similar to those analysed in [4]:

$$y^* = \Phi(Xw^*) \tag{2.3}$$

$$y = y^* + b + \epsilon, \tag{2.4}$$

where X is a matrix of features, w^* is a vector of weights, $\Phi(\cdot)$ is a non-linear map, b denotes the corruptive noise in the observation due to the measurements, and ϵ denotes the regular Gaussian white noise. In the following sections, the model will be analyzed for the case $y = \hat{y} + b$ without the Gaussian white noise to be able to devise a simple yet efficient hard thresholding method. This relaxation allows the IHT approach to be viable during subset selection. In robust network literature, noise vector b is generally taken as a sparse vector such that $\|b\|_0 \leq 0.4n$ where n is the number of observations [9]. Due to this sparsity pattern in b which is induced from the ℓ_0-norm, one can reformulate the problem in the form of a compressed sensing problem [4]. Bhatia *et. al.* [4] proves the convergence guarantees and compares performances for the subset based regression techniques TORRENT and ADACRR [10] using this sparse recovery reformulation to the

robust regression methods. We shall follow a similar approach to analyze the recovery problem and its applications.

2.1.3 ELM Model Description

Let $X = [x_1, x_2, ..., x_n]^T$ be a feature matrix of dimension $n \times p$, such that $x_i \in \mathbb{R}^p$. Let $y \in \mathbb{R}^n$ be the target vector for all n observations. Fixed weight matrix $W \in \mathbb{R}^{p \times l}$ represents the randomly generated layer and $w_2 \in \mathbb{R}^l$ is optimized in the second layer. The dimension of the first layer is denoted by l whereas ϕ denotes any transfer function such as ReLU, Leaky ReLU, tanh or sigmoid:

$$Z = \phi(XW) \tag{2.5}$$

$$\hat{y} = Zw_2. \tag{2.6}$$

In order to calculate the second layer weights, w_2, one can use various gradient descent algorithms in addition to the widely used ℓ_2 norm minimization formula. The widely used closed form solution of the second layer weight is shown in [11] as

$$w_2 = \begin{cases} (Z^T Z)^{-1} Z^T y & n \geq l, \\ Z^T (Z Z^T)^{-1} y & n \leq l. \end{cases} \tag{2.7}$$

The above closed form expressions are derived from least squares minimization. Depending on the existence of the generalized inverse, one of the identities is used.

The transformation in the random layer is analogous to dimensionality reduction using random projections when $l < p$ and the related JL Lemma. This property will be useful to show that the data structure is preserved throughout the network regardless of the non-linear transforms.

In addition, there are algorithms involving Iterative Hard Thresholding in ELMs [12]. However, these algorithms are applied to the decision weights on the second layer to obtain sparse weights. The present study is completely different

in terms of the use of the iterative hard thresholding and sparsity sought in the variables. However, our theoretical study supports the foundation of the proposed algorithm in [12] implicitly where they lack theoretical results.

2.1.4 Robust Methods and Related Loss Functions

Regularization methods are studied in detail in many different areas including machine learning, compressed sensing and optimization.

The objective function of a classical regularized regression problem could be defined as:

$$\mathcal{L}(w) + \lambda \mathcal{R}(w)$$

where w represents the regression weights, $\mathcal{L}$ is the desired loss function, $\mathcal{R}$ is a regularization term to induce robustness and sparsity, and λ is a hyper-parameter tuning the trade-off between the cost function and the regularization term.

Robust loss functions are selected from the functions which are less sensitive to the outliers to induce robustness in the system. Huber loss is one of the most commonly used robust loss functions in the literature. Intensive analysis on the function and its implementations for many ML studies are available, e.g., [13],[14].

As an overview, one can summarize the most commonly used regularization methods in NN's as ℓ_1, ℓ_2 regularization and dropout. Regularization methods in NN are similar to the classical regression counterparts. In addition to the regularization term in the output weights, ℓ_1 and ℓ_2 norms of a layer weights can be easily integrated to the loss function. For a generic two layer Neural Network, we have a similar transformation described in 2.5, however, instead of the fixed

and random first layer weights W, first layer weights W_1 are learned.

$$Z = \phi(XW_1) \tag{2.8}$$

$$\hat{y} = Zw_2. \tag{2.9}$$

Loss function of this Neural Network with layer regularization would be

$$\mathcal{L}(W_1, w_2) + \lambda_1 \mathcal{R}_1(W_1) + \lambda_2 \mathcal{R}_2(w_2)$$

Where $\mathcal{R}_2$ is a general ℓ_2 or ℓ_1 norm, and $\mathcal{R}_1$ is a generally in the form:

$$\mathcal{R}_1 = \beta_1 \|W_{1i}\|_1 + \beta_2 \|W_{1i}\|_2$$

similar to an elastic net regularization where W_{1i} denotes the i'th column of the weight matrix W_1.

In the literature there exist methods to transform robust functions and regularization methods into a compact format to be used in robust networks [9]. These are efficient in terms of computation and implementation as the form of each loss can be written in terms of iteratively re-weighted least squares function. Furthermore, involved methods on parameter selection are known for online-sequential learning [15] with more specific implementation details that are not within the scope of this thesis.

2.1.5 Convex Neural Networks

In very recent literature [16] the convexity of the two-layer Neural Networks is analyzed. Pilancı *et al.* [16] shows the equivalence of the classical two-layer ReLU neural network (2.8) to the following convex program

$$\min_{\{v_i,w_i\}_1^P} \frac{1}{2}\left\|\sum_{i=1}^{P} D_i X(v_i - w_i) - y\right\|_2 + \beta\sum_{i=1}^{P}(\|v_i\|_2 + \|w_i\|_2)$$
$$\text{s.t.} \quad (2D_i - I)Xv_i \geq 0, \quad \forall i \in \{1,..,P\}$$
$$(2D_i - I)Xw_i \geq 0, \quad \forall i \in \{1,..,P\}$$

where the diagonal matrices D_i's correspond to a hyperplane arrangement and P is the number of all hyperplane arrangements. Since the analysis of the hyperplane arrangements and the equivalence of the two problems are out of the scope of this study, we refer the readers to [16] for details. Furthermore, the original weights for the neural network can be obtained based on a result detailed in [16]. The most crucial point of this convex formulation is the hyperplane arrangements denoted by D_i in the formulation. This feature is introduced to aggregate the data with its small subsets so that the problem becomes a sparse recovery problem with the group sparsity regularization term.

There are robust Neural Network studies extending this approach in [17]. With this approach, the robustness around a given perturbation ball can be implemented using convex optimization. However, from a practical viewpoint, these implementations are not on a par with the classical neural networks, and the scaling performance to large data sizes is worse compared to that of neural networks.

The afore-mentioned convex approach is generally not practical but very insightful for theoretical analysis. We also find that our approach parallels those theoretical insights presented in [16] and [17].

2.2 Deep Portfolio Optimization

2.2.1 Neural Networks and Time Series

In time series analysis, standard neural network architectures generally incorporate recurrent layers, various convolutional layers, such as dilated convolutions, and attention mechanisms [18]. These layers allow models to effectively use the sequential data. Attention mechanisms and GNN's are also effectively used in the recent literature for time series prediction tasks.

2.2.1.1 Feed-Forward NN

Feed-forward neural networks are the basis of neural network architectures. The transformation in a two-layer feed-forward neural network in its simplest form is defined as it is in 2.8. These layers are generally used along with a more involved layer such as a convolution or recurrent layers.

2.2.1.2 Convolutional NN

In addition to the success of convolutional layers in computer vision and image processing literature, 1-dimensional convolutional layers are also used in the literature on time series analysis [19]. Methods such as dilated convolutions are effective in increasing the receptive field, yielding results comparable to recurrent models.

2.2.1.3 Recurrent NN

Recurrent Neural Networks (RNN) are essential architectures for time-series tasks in deep learning. Recurrent layers combine time series information with features

to effectively perform the task. In the vanilla version of RNNs, there are potantial drawbacks of vanishing and exploding gradients. In order to address this issue, LSTM and GRU layers are proposed in the literature [20], [21]. LSTM is among the most commonly used layers in the deep sequence models due to its success in natural language processing and time-series data. Even though the inner structure of the layer is quite convoluted, it essentially learns to remember and forget sequential information to improve the performance of the classical recurrent layers.

2.2.1.4 Residual Connections

Residual connections also referred to as skip connections, behave very similar to classical feedback systems. These connections are used to propagate the gradient backward from a source. The gradient is split into two when there is a skip connection and it is useful when a vanishing gradient problem occurs [22]. In computer vision, ResNet architecture was a significant breakthrough as it made training deeper neural networks significantly easier which improved the performance of the networks. A simple representation of a residual connection could be written as

$$X' = \phi(X) + XW_s$$

where $\phi(X)$ is a general layer transformation and XW_s denostes the skip connection with weights in a general form. Commonly, W_s is used as an identity matrix.

2.2.1.5 Dropout

In general, the network dimensionality is desired to be large enough to capture the information within the data to effectively perform learning without underfitting. Doing this might result in overfitting. Dropout is among the most common techniques used in neural networks to prevent overfitting. As the name suggests,

the method involves dropping some neurons with a pre-determined probability during training to prevent excessive focus on a few neurons [23]. This idea is described in the original paper as a way of sampling. The idea is to sample multiple sub-networks (thinned networks) from the original network and train them separately. In the testing phase, the full network is used to perform the given task. However, by having the neurons trained separately with the given stochastic regularization idea, the robustness of the network is improved significantly.

2.2.1.6 Graph Neural Networks

Graph convolutions are the generalized version of the convolutional networks. Instead of using the transformation of the adjacent features, a graph structure is used to generalize the adjacency notion. The most basic two-layer Graph Convolutional Network is represented as follows:

$$H = f(X, A) = \phi(AReLU(AXW_1)W_2),$$

where $X \in \mathbb{R}^{p \times l}$ with p denoting the company number (for the assets) and l denotes the lookback window (features). $A \in \mathbb{R}^{p \times p}$ is the (normalized) adjacency matrix. $W_i \in \mathbb{R}^{l \times h_i}$ and h_i denote the number of filters used in each layer i.

Some of the prominent layers used in GNN's are GAT, GCN and ARMA.

- Graph Attention layer (GAT):
  In [24], it was proposed to use an attention mechanism in GNN's. For every node, this layer concatenates the joint information of its neighbors while weighting the information important to that node. The operation could be described as follows:

$$\alpha_{ij} = \frac{exp(LeakyReLU(a^T[XW_i \| XW_j]))}{\sum_{\mathcal{N}_i} exp(LeakyReLU(a^T[XW_i \| XW_j]))}$$
$$X' = \alpha XW + b$$

where $\|$ operator denotes concatenation, $\mathcal{N}$ denotes the adjacency matrix with self-loops. Due to its performance in node classification problems, this layer has been used in most of our architectures.

- Graph Convolution with Residual Connection (GCS):
  This is a simple Graph Convolutional layer with residual connections. The operation performed by this layer is as follows:

$$\tilde{A} = D^{-1/2}AD^{-1/2}$$
$$X' = D^{-1/2}AD^{-1/2}XW_1 + XW_2 + b,$$

  where D denotes the degree matrix, and the adjacency matrix A does not contain self-loops [25]. With the residual connections, this layer is very easy to train and could be considered as the baseline graph convolutional layer.

- Auto-Regressive Moving Average Convolution (ARMA):
  This layer carries the spirit of the original auto-regressive models and GCS layers [26]. For a given order K, the model performs the following:

$$X^0 = X$$
$$\tilde{A} = D^{-1/2}AD^{-1/2}$$
$$X_k^{t+1} = \sigma(\tilde{A}X^tW^t + XV^t)$$
$$X' = \frac{1}{K}\sum_{k=1}^{K} X_k^t.$$

2.2.2 Portfolio Optimization Problem

In the literature, the Mean-Variance portfolio problem was extensively studied. There are various studies involving different loss functions and different constraints. The classical optimization problem targets maximizing future returns while keeping a cap on variance of portfolio return using the first and second-order

information of the time series data:

$$\begin{aligned} \max_{w} \quad & \mu^T w - \alpha w^T \Sigma w \\ \text{s.t.} \quad & w_i \geq 0 \quad \forall i \in \{1, \ldots, p\} \\ & e^T w = 1, \end{aligned}$$

or minimizing the variance of portfolio return while ensuring a minimum expected portfolio return:

$$\begin{aligned} \min_{w} \quad & w^T \Sigma w - \beta \mu^T w \\ \text{s.t.} \quad & w_i \geq 0 \quad \forall i \in \{1, \ldots, p\} \\ & e^T w = 1 \end{aligned}$$

where e denotes a vector with all components equal to one, and α, β are real scalars that reflect the balance between risk and returns (the models are to be equivalent under judicious choices of parameters α and β). There are many variants to the above problems that relax the non-negativity constraint or change the objective function to a fractional function which maximizes the Sharpe ratio. Essentially, these problems are similar in nature, and the goal is to obtain the highest return with the least amount of portfolio risk. In this study, we will focus on the notion of algorithmic risk with less attention to portfolio risk as there are different robustness problems that arise in fully data-driven approaches.

Chapter 3

Deep Portfolio Optimization

In this section, we will briefly review relevant literature, present the deep learning framework we used for the portfolio optimization problem, point out a few challenges in the literature and explain the method we follow to utilize graph neural networks in this general deep learning framework. We will explain how we obtained the graph structure and how we improved the results of the state-of-the-art methods.

Recently in portfolio optimization literature, data-driven methods started to replace the classical model-based optimization methods. Data-driven methods range from reinforcement learning to natural language processing models which predict the market movements from the text data [27]. Depending on the dataset structure, various deep learning methods have been tested for this problem. Earlier methods focus on the time-series prediction task. However, recent studies show that it is not always clear how to allocate resources for the portfolio. To bridge this gap, [28] proposes a framework where forecasting is performed first, and allocations are made afterward by a two-stage method. The latest approaches in the literature involve the usage of an end-to-end deep learning framework to directly learn the asset allocation [3].

Furthermore, end-to-end methods are preferred over their competitors due to

the flexibility in loss functions. These allow the neural network to be adjusted towards different goals such as return maximization or Sharpe Ratio minimization. Furthermore, constrained portfolio problems can also be integrated into the end-to-end deep portfolio network with a slight change in loss functions, making the end-to-end approach in portfolio optimization quickly become a desirable approach [29].

In the classical time-series prediction or trend prediction problems, GNN's have proven their success [30],[31],[32]. However, these models generally require hand-crafted adjacency matrices and are generally used for the classical regression/classification tasks. We propose to leverage the effectiveness of these graphical models by incorporating the graph knowledge in the end-to-end portfolio optimization framework proposed in [29].

Besides all the nice features deep end-to-end models provide, there are still computational problems in terms of the stability of the algorithms. In the literature, some of the layers, especially dense layers, turned out to be very noisy, causing performance and reliability issues. Furthermore, the datasets used in the field are generally smaller than the ones used in the other fields of deep learning literature. This makes the networks difficult to train due to the sample sizes obtained from the daily market data. We propose to use a graphical neural network approach to address these problems and we compare our results with the state-of-the-art recurrent models. We also leverage the classical graphical lasso algorithm while generating the inverse covariance matrix, which saves us from handcrafting all the adjacency information and makes our algorithm easy to implement.

The main challenge in this problem is to handle the financial market datasets. In synthetic or long time series, having large amounts of observations makes the problem simpler. However, in market datasets, relevant market data might be quite small compared to the other literature. The main reason is that the relevant historic market prices might be quite small for a given company. For example, 10 years of data make roughly 3000 observations per company, assuming that the company stays in the observed market index over that period. This is nowhere near the amount of data used in most applications. Time granularity is

another restrictive factor as the daily variations are mostly noisy and the relevant data used in the prediction are the last few days most of the time [3]. From a practical point of view, privacy is a final concern as the data in financial studies might be quite restrictive due to the company policies. There are even new studies proposing to address this issue by generating realistic time series data while preserving the privacy of the clients [33],[34].

3.1 Contribution

Our contribution to the literature could be summarized as follows.

- Introducing GNN models to the problem: GNN's yield higher portfolio returns compared to the state of the art recurrent models.
- Recurrent models suffer from stability issues. We address this problem by using GNN's, and analyze the algorithmic risk-return trade-off in a fully data-driven portfolio optimization framework.
- Instead of carefully handcrafting the adjacency matrix, we propose to use the inverse covariance matrix of the returns, which is a method known in the optimization literature but not deployed in the deep learning literature.

3.2 Our Method

Let $\mathcal{G} = (V, E)$ be a graph. In our model, V denotes the companies, and edges, E, are weightless and bidirectional to denote the relations between the companies. We assume given a dataset $D \in \mathbb{R}^{n \times p}$ where there are n time steps, i.e., trading days, and p companies. The first step is to create a graph representation using the given dataset. To infer the adjacency relation, we use the Graphical Lasso Algorithm to calculate the inverse covariance matrix. Since this sparse matrix gives the conditional relation between each company, it is an intuitive model-based

approach we can use freely. In the literature, some studies manually construct graphs using market information available [30]. However, to have an easy-to-use framework, we refrain from using expert knowledge and utilize graphical lasso. The graphical lasso algorithm solves the following problem.

$$\hat{\Theta} = \arg\min_{\Theta} Tr(S\Theta) - \log\det(\Theta) + \lambda \sum_{i \neq j} |\Theta| \,,$$

where S denotes the covariance matrix. This optimization problem is a well-known and well-studied problem in the literature, and its adaptive extensions are also available [35]. These adaptive extensions are important in cases where the change in the adjacency graph is crucial such as car sensor data. In the present study, due to the structure of the real-life dataset that we have worked on, this extension is not used. However, especially in large datasets, an adaptive version could be preferred to have more precise adjacency matrices.

Combining the return data with the adjacency graph obtained, we obtain our input graph $\mathcal{G}$, and we use our proposed GNN model to predict the portfolio allocation by modifying the previous formulation using flattening and dense layers:

$$out = softmax(flatten(H)W_3),$$

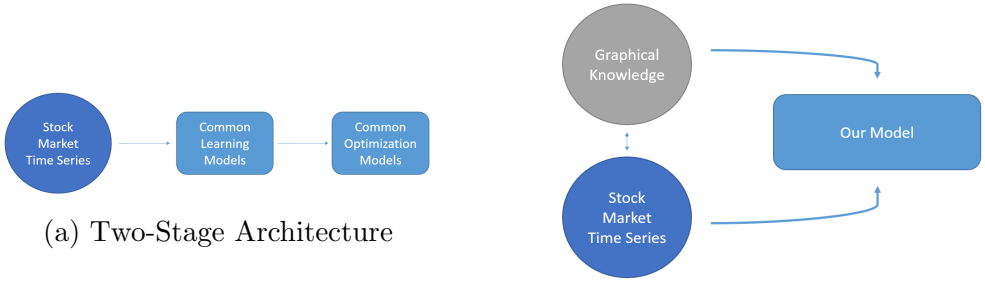
where $W_3 \in \mathbb{R}^{h_f p \times p}$ since the flattened graph will be a vector of length $h_f \times p$, and the output will be p-dimensional. The "softmax" activation is very useful to directly obtain the allocations in the portfolio optimization problem which are non-zero and summed to one. Furthermore, allocation constraints could be implemented or relaxed by modifying the output activation of the network [29].

In this study, we mainly focus on maximizing returns. In [36], it is shown that the Sharpe ratio criterion can effectively minimize the variance of the returns, which is the portfolio risk. In addition to the portfolio risk, the stability of the algorithm is an additional source of concern for the investor in data-driven methods. We believe that having a robust and stable model is instrumental

in achieving high returns while effectively minimizing the algorithmic risk. We aim to obtain results giving less algorithmic risk by manipulating the network architecture of the deep learning approaches so that we can maximize the return per risk notion in the fully data-driven case. We also look for the algorithmic risk-return trade-off for multiple algorithms.

3.3 Architecture

We adopt an end-to-end approach in our proposed network similar to [3]. This is the most recent approach in the literature and it is flexible and effective. The other possible approach is a two-stage approach, which is shown in Figure 3.1 where we first predict and then optimize. We mostly focus on the end-to-end architectures due to their desirable properties and performance [29].



(a) Two-Stage Architecture

(b) End-To-End Architecture

Figure 3.1: Deep Learning Architectures for Portfolio Optimization

We have tested multiple GNN layers and dense layers. However, due to the dimensionality of the dataset, we prefer to stick to a simple architecture similar to the ones used in node classification problems [24]. Input to our model is the past return information, the first and second moments of the returns as well as the adjacency matrix. This input is fed into three layers of graph convolutions. Afterward, we flatten the data and feed it into two dense layers. In these dense layers, we use residual connections to improve the training performance. We also observed that utilizing the GELU activation function instead of the classical ReLU in the convolutional layers yields better results [37]. On different tests,

we have used different graphical convolutions and the results are reported in the results section.

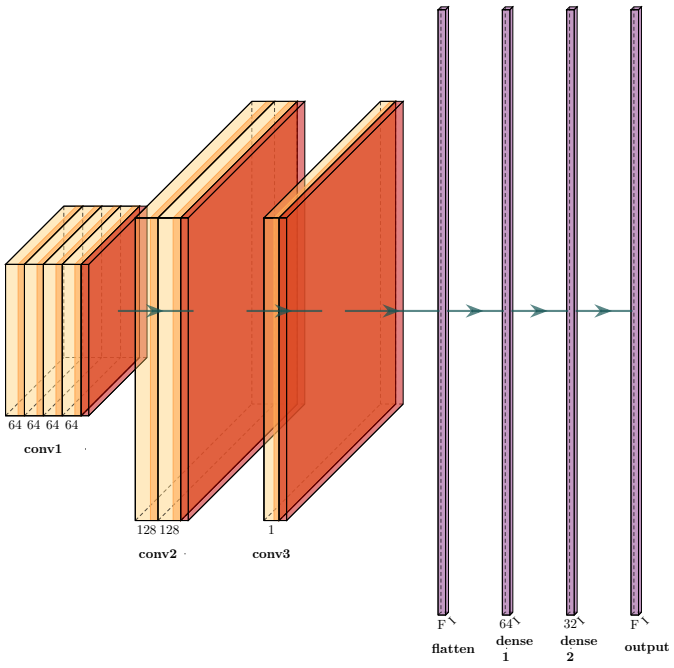


Figure 3.2: Proposed GNN Architecture

In Figure 3.2, we represent our GNN architecture. We have tested the model with the graphical convolutional layers described in the Background section. In the Figure 3.2, the number of filters represents the number of attention heads used in the GAT model. We also used a very similar architecture in our ARMA and GCS models, however, the hyperparameters slightly vary. In the figure, filter sizes do not reflect the real sizes of the figure and are drawn to present the architecture. Essentially, in all our architectures, we stack multiple graphical convolutional layers and follow up with dense layers with drop-out and residual connections.

3.4 Results

We have used datasets obtained from `https://host.uniroma3.it/docenti/cesarone/DataSets.htm`, which are also used in [38]. Among the available datasets, we have used Dow Jones 2005 (Dataset 1), Euro Bonds (Dataset 2), Commodities and Italian Bonds mix (Dataset 3), and World Bonds Mix (Dataset 4). These datasets contain 1564 days of data of assets with sizes varying from 11 to 104 securities. We also reproduce from [38] the data features in Table 3.1.

Table 3.1: Dataset Structure

Datasets	Assets	Days	Range
DowJones2005 (1)	21	1564	1/2013-12/2018
EuroBonds (2)	62	1564	1/2013-12/2018
ItBondComodities (3)	11	1564	1/2013-12/2018
WorldMixBonds (4)	104	1564	1/2013-12/2018

We have separated these datasets and created 12 trading instances. For every instance, we have separated training and testing data. Training data for every instance consists of the training and testing data of the previous instance. Each first instance contains 400 days of training and 100 days of prediction. We then merge that mini data to have training data for the following mini data and continue until we cover all the available data. For example, the second instance will contain 500 days of training data and 100 days of validation; the third instance will contain 600 days of training and 100 days of validation so on. This methodology was also adopted in [29]. To evaluate the algorithmic risk, we run each model 10 times for each mini dataset, and the average results are reported. We denote our proposed models after the graphical convolutional layers used in the model: *ARMA*, *GAT* and *GCS*. *LSTM* denotes the architecture proposed in [3] and [29]. *CVX* refers to the classical long-only mean-variance return maximization problem. We followed a $90\% - 10\%$ split in training data to validate and tune our hyperparameters. Throughout all the experiments, Tensorflow and Spektral libraries are used to train and evaluate the neural networks [25]. In the implementation of the optimization-based models, CVXPY is used [39].

In terms of the performance criteria, our main focus was to obtain high returns. Therefore Table 3.2 summarize our primary results. Furthermore, we have investigated the Sharpe ratios obtained from the models to have an idea of the risk of the returns. We report these results in Table 3.3. However, our main focus wasn't on the Sharpe ratios and deep models are not trained for that. It is only used to have an understanding of the performance of the model. Our secondary focus was to reduce the algorithmic risk, which we defined as the risk of not converging to a similar local optimal point consistently. We present our evaluations for this criteria using the figures 3.3,3.7,3.9,3.11,3.4,3.8,3.10,3.12.

Table 3.2: Evaluation of Total Returns on Datasets

Dataset	ARMA	GAT	GCS	LSTM	CVX
1	1.3601	1.5557	1.3921	1.2708	**1.8715**
2	1.6936	0.8172	**2.1713**	0.8418	1.1609
3	**0.5046**	0.3610	0.4217	0.3898	0.4112
4	1.2971	0.7250	**1.7297**	0.8156	1.2526

Table 3.3: Evaluation of Sharpe Ratios on Datasets

Dataset	ARMA	GAT	GCS	LSTM	CVX
1	0.6164	0.6142	0.6337	0.5820	**0.7612**
2	0.7542	0.4543	**1.0111**	0.5772	0.0146
3	0.5407	0.5373	0.4396	0.5205	**0.7260**
4	0.4957	0.4015	**0.7360**	0.4859	0.0815

Results in Figure 3.2 indicate that the proposed GCS and ARMA networks yield results surpassing the returns obtained by the state-of-the-art LSTM networks. Moreover, the GAT network fixes the stability problem of the deep models by converging to a similar local optimal solution consistently. Due to the limited observations in the financial datasets, it is quite difficult to train a robust deep architecture. This is also applicable to the model-based architectures as in the 4th dataset, the CVX model failed to give meaningful results when the dataset is small. We omitted the first two time instances because of this problem. As a

Table 3.4: Return Statistics on Every Instance for Dataset 1

GAT Mean	GAT Std	LSTM Mean	LSTM Std
0.048215	0.00012	0.069446	0.01740
0.046904	0.00815	0.036681	0.01861
0.180032	0.00672	0.198491	0.03683
-0.002247	0.0026	-0.055583	0.05020
-0.120561	0.00771	-0.117312	0.02362
-0.033928	0.00001	-0.026766	0.04347
-0.011228	0.00029	0.051438	0.05923
0.135172	0.01676	0.238524	0.06555
0.139644	0.00026	0.204196	0.09279
0.079272	0.00017	0.219471	0.04841
0.620897	0.16890	0.297301	0.16500
0.473544	0.16920	0.154340	0.16508

result of these challenges, simpler models like GCS are effective in those datasets. Furthermore, GAT results appear to be consistent in most instances. The significant stability difference between the LSTM model and our proposed GAT model is apparent in the figures. In the reported figures, the x-axis represents the trading instances where we evaluate our models and the y-axis represents the obtained returns. The error bar is plotted to show the standard deviation of the obtained returns. The difference in the error bars is useful to understand that the algorithmic risk decreases with the usage of the GAT models. This indicates that the graph models converge to a similar point in every different training session eliminating the algorithmic risk. Figures 3.3 and 3.4 show the standard deviation of the different returns obtained from the first dataset for every instance for each run. Similarly, Figures 3.5 and 3.6 show the maximum difference encountered in every evaluation period. We present the results obtained for each time instance in Tables 3.4, 3.5,3.6,3.7,3.8,3.9,3.10,3.11. In general, the error bars in the figures representing the LSTM models are larger indicating a larger standard deviation in the obtained results, whereas the GAT model is more consistent and the error bars reflect only a small amount of variation.

In some datasets, LSTM mean returns are slightly higher than the GAT model

Table 3.5: Return Statistics on Every Instance for Dataset 1

GCS Mean	GCS Std	ARMA Mean	ARMA Std
0.038412	0.05123	0.041035	0.03390
0.04756	0.06023	0.049132	0.05677
0.149021	0.09615	0.125844	0.09016
-0.03995	0.08870	-0.01665	0.06642
-0.099934	0.08627	-0.039357	0.08307
0.037406	0.05082	-0.016773	0.06644
0.089273	0.07589	0.125617	0.13105
0.169544	0.06934	0.173572	0.05251
0.237483	0.13495	0.161368	0.12450
0.315011	0.16075	0.192065	0.10870
0.263893	0.14250	0.274490	0.18700
0.184350	0.17246	0.289794	0.18170

returns. However, the variance in the returns is significantly high for every training period, making the models quite unreliable. Similarly, if we want to obtain higher returns with higher risk, the GCS and ARMA models could be used to outperform the returns of the recurrent models. There are ocassions where the standard deviation of the LSTM and graphical models are 0. In those cases, they yield the same return, converging to the same local optima. In other points, depending on the dataset, LSTM and graph models are competing in returns and standard deviations. Since it is difficult to observe a model to strongly dominate another model in each time instance, it is meaningful to compare the cumulative returns as shown in 3.2. These results show that graph models are superior to the LSTM model in various scenarios both in terms of the returns and in terms of the sharpe ratios.

Portfolio optimization is often analyzed along with Pareto-optimal solutions due to the risk-return trade-off. In our results, we can see a variant of the portfolio's risk-return trade-off in terms of algorithms' risk-return trade-off. Our proposed GAT model significantly reduces the algorithmic risk while maintaining, or even exceeding the return levels achieved with state-of-the-art LSTM methods. Furthermore, the GCS and ARMA models manage to beat the returns of the

Table 3.6: Return Statistics on Every Instance for Dataset 2

GAT Mean	GAT Std	LSTM Mean	LSTM Std
0.038904	0.00417	0.087058	0.02000
-0.128798	0.00018	0.041902	0.03134
-0.200803	0.01719	-0.031716	0.13476
-0.103257	0	-0.103261	0
0.114179	0	0.114179	0
0.084612	0	0.065986	0.01382
0.20806	0.00484	0.19411	0.08750
0.03447	0.024714	0.050097	0.10716
0.249354	0.01272	0.164206	0.04918
0.033278	0.05461	0.102287	0.07098
0.310822	0.06093	0.125728	0.037847
0.176451	0.06454	0.031241	0.03167

LSTM model which makes the graphical models highly effective for deep portfolio optimization.

3.5 Remarks

In deep portfolio optimization, some of the traditional portfolio optimization metrics should be evaluated from a different perspective. As it could be observed from the results reported in [36], the effect of Sharpe ratio minimization effectively decreases the variance of returns. However, when the Sharpe ratio minimization is the key objective, returns also decrease significantly. Furthermore, the resulting portfolio remains risky due to the algorithmic risk stemming from the instability of the models. We can mitigate this issue separately by focusing on the stability and robustness of the network. This will yield returns not far off the predicted levels, which is beneficial to evaluate the portfolio's performance.

Recent studies suggest that the first few days are important in a recurrent model [3]. Similarly, from our results, we observed that the graphical relations

Table 3.7: Return Statistics on Every Instance for Dataset 2

GCS Mean	GCS Std	ARMA Mean	ARMA Std
0.094956	0.02412	0.084313	0.02368
0.083139	0.05309	0.017688	0.08396
0.128395	0.04228	0.094413	0.15007
0.022334	0.34606	0.361661	0.22888
0.527551	0.27234	0.367827	0.35571
0.14709	0.05072	0.191131	0.11247
0.239199	0.10860	0.286001	0.09095
0.132828	0.11793	-0.026266	0.11242
0.199725	0.05234	0.00733	0.11185
0.151399	0.10003	0.053952	0.08785
0.249482	0.08962	0.226717	0.09371
0.19520	0.06458	0.225212	0.05041

are effective to explain the information within the data, which makes the model more robust than the recurrent counterparts. These insights are important for the deep portfolio optimization frameworks since the notion of algorithmic "risk" should also be considered for fully-data-driven approaches. We also observe that different layers excel at interpreting different return structures. For example, when the returns are increasing in the data, ARMA convolutions return high rewards compared to their alternatives. However, due to its higher loss when the returns are declining, the model overall may fail to return the highest rewards. These observations could be useful if the allocation decision could be made with an ensemble model, which we will be focusing on in our future work.

3.6 Conclusion

End-to-end deep learning methods started to replace model-based portfolio optimization tools. We have analyzed the classical Markowitz portfolio optimization problem from a deep learning perspective and addressed one of the key issues. Algorithmic risk is a crucial problem in the data-driven approaches to the problem.

Table 3.8: Return Statistics on Every Instance for Dataset 3

GAT Mean	GAT Std	LSTM Mean	LSTM Std
0.056009	0.00611	0.059118	0
0.103474	0.01560	0.115378	0.00001
0.243125	0.04964	0.259768	0
-0.064475	0.00002	-0.064491	0
0.152873	0	0.152878	0
0.079757	0.00004	0.079773	0
0.001202	0	0.001199	0
-0.109843	0	-0.109845	0
-0.161551	0	-0.161555	0
0.010926	0	0.010925	0
0.102952	0.00004	0.10297	0
-0.053466	0.00822	-0.056293	0

We improved the stability of the state-of-art recurrent models by utilizing graph neural networks. Our contribution to the literature is twofold. We proposed to use the graphical lasso to create an adjacency matrix of the companies. This allowed us to obtain an adjacency matrix without the need for expert knowledge and hand-crafting. Second, we showed that the GNN results are more stable compared to the LSTM models and the achieved returns are comparable, even better in some cases. These results show that the algorithmic risk minimization could be enhanced with neural network techniques, which is crucial in a fully data-driven architecture.

Table 3.9: Return Statistics on Every Instance for Dataset 3

GCS Mean	GCS Std	ARMA Mean	ARMA Std
0.05912	0	0.058494	0.00143
0.034659	0.07603	0.056641	0.08171
0.259772	0	0.259741	0.00002
-0.064491	0	-0.064491	0
0.125136	0.06479	0.152876	0
0.079529	0.07281	0.094402	0.05800
0.051029	0.09568	0.041275	0.05942
-0.099974	0.10227	-0.115771	0.04093
-0.185021	0.13051	-0.154651	0.02973
0.004418	0.06935	-0.000504	0.05060
0.113949	0.09368	0.094835	0.04887
0.043611	0.09673	0.081722	0.13678

Table 3.10: Return Statistics on Every Instance for Dataset 4

GAT Mean	GAT Std	LSTM Mean	LSTM Std
0.042279	0.01278	0.053117	0.02176
-0.128956	0.00007	-0.069907	0.04750
-0.19507	0.00007	-0.16372	0.10501
-0.103258	0	-0.013079	0.27055
0.114179	0	0.114179	0
0.084615	0	0.024966	0.03719
0.210526	0.00006	0.173922	0.10851
0.029786	0.03394	0.120206	0.10492
0.239829	0.00126	0.143658	0.12427
-0.000763	0.00089	0.109348	0.08395
0.301698	0.06475	0.18956	0.06376
0.130126	0.04339	0.133377	0.03863

Table 3.11: Return Statistics on Every Instance for Dataset 4

GCS Mean	GCS Std	ARMA Mean	ARMA Std
0.012129	0.04658	-0.001259	0.04340
-0.007884	0.09972	0.046798	0.10370
0.118145	0.21802	-0.066621	0.15955
0.331967	0.34264	0.133187	0.25834
0.281846	0.20566	0.337363	0.35518
0.225618	0.04142	0.217437	0.11502
0.076641	0.08017	0.157311	0.13209
0.091074	0.13573	0.014852	0.08778
0.186322	0.09682	0.094761	0.11105
0.169197	0.06723	0.093558	0.10268
0.109906	0.11902	0.176996	0.09885
0.134762	0.07332	0.092671	0.08549

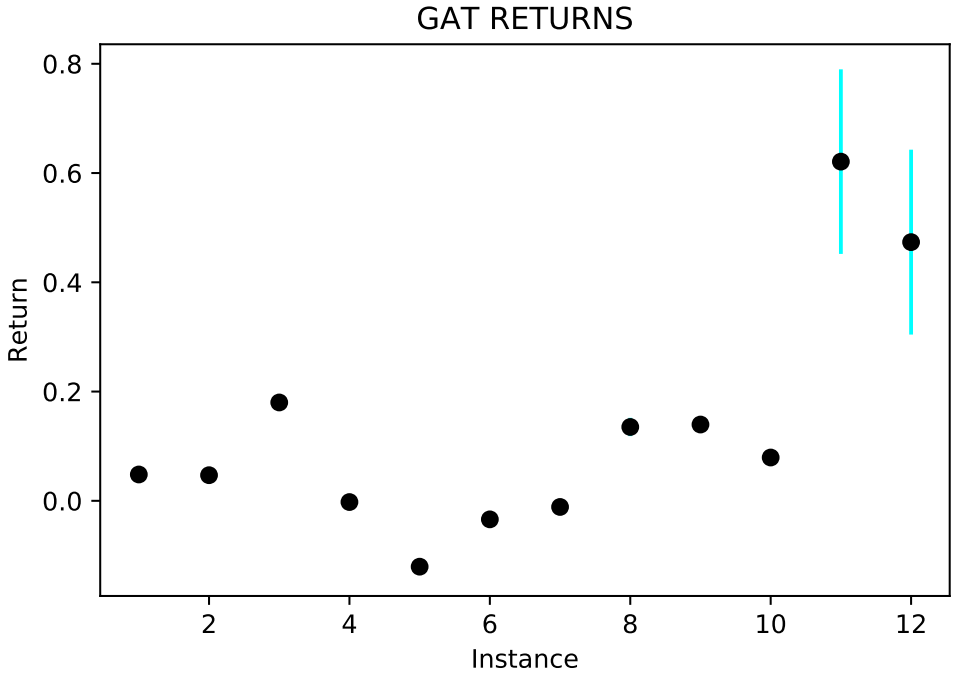


Figure 3.3: GAT Returns with Std Error on Dataset1

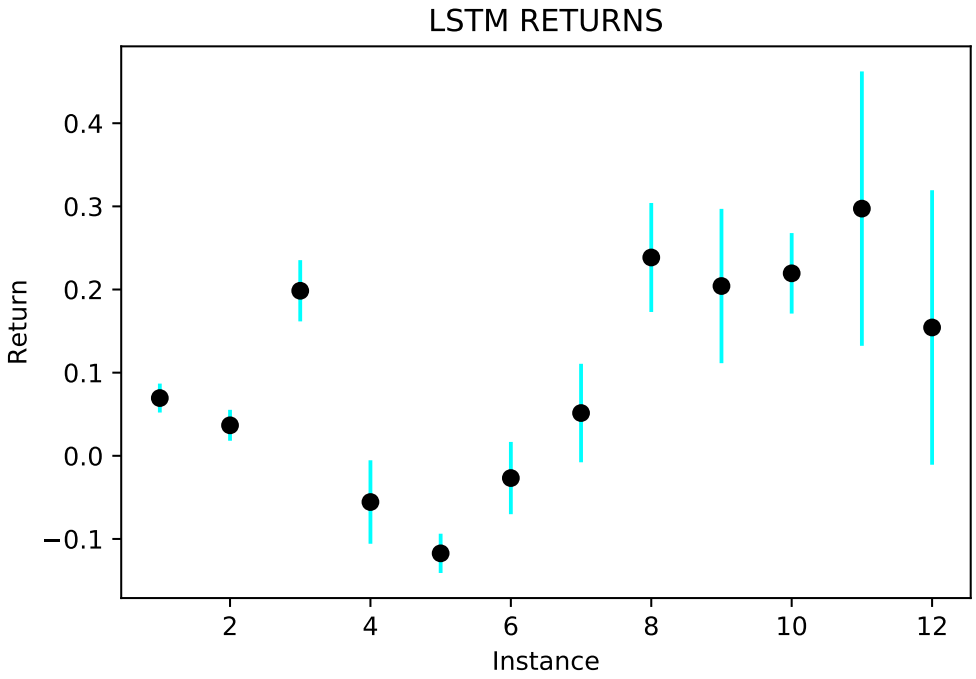


Figure 3.4: LSTM Returns with Std Error on Dataset1

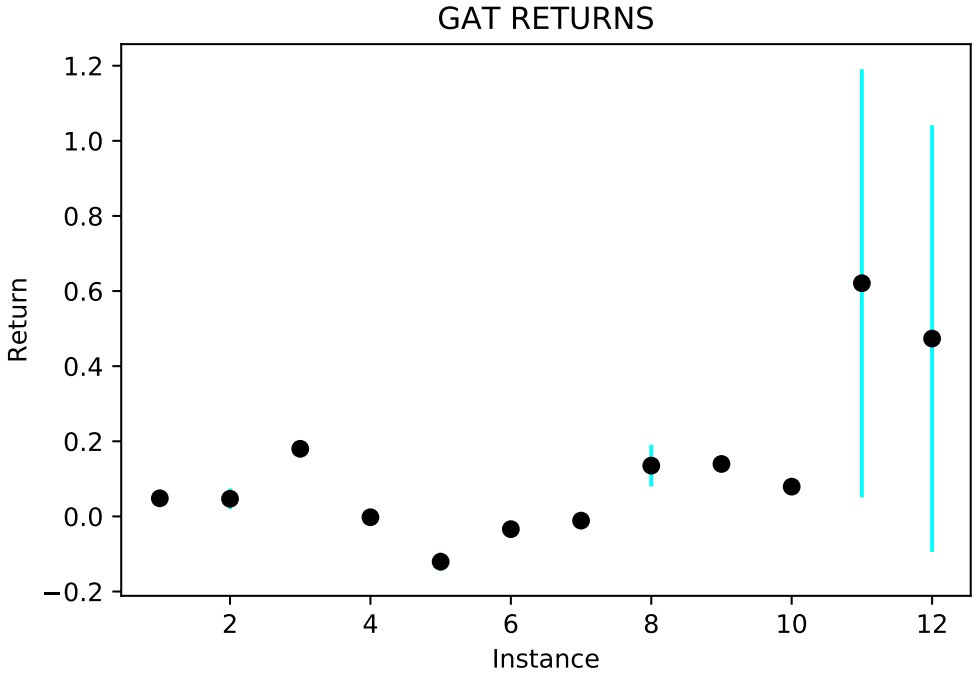


Figure 3.5: GAT Returns with Max Error on Dataset1

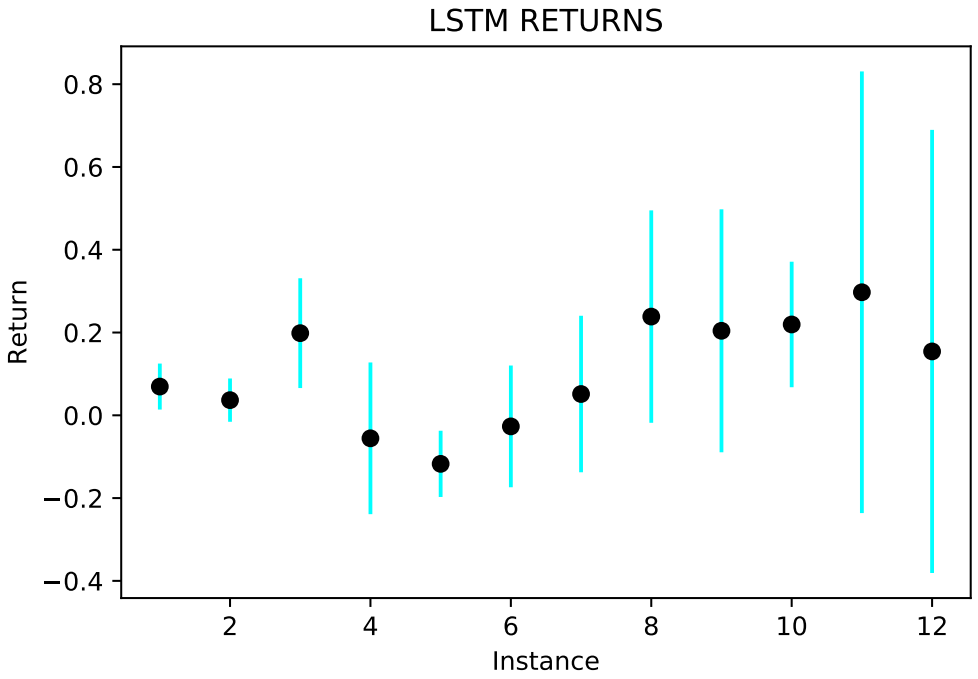


Figure 3.6: LSTM Returns with Max Error on Dataset1

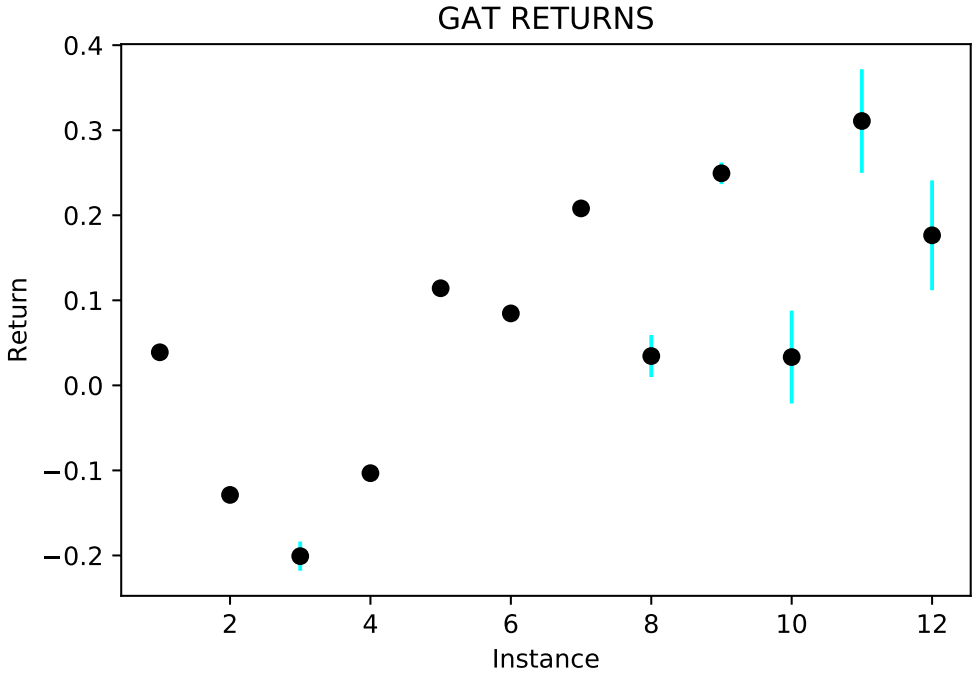


Figure 3.7: GAT Returns with Std Error on Dataset2

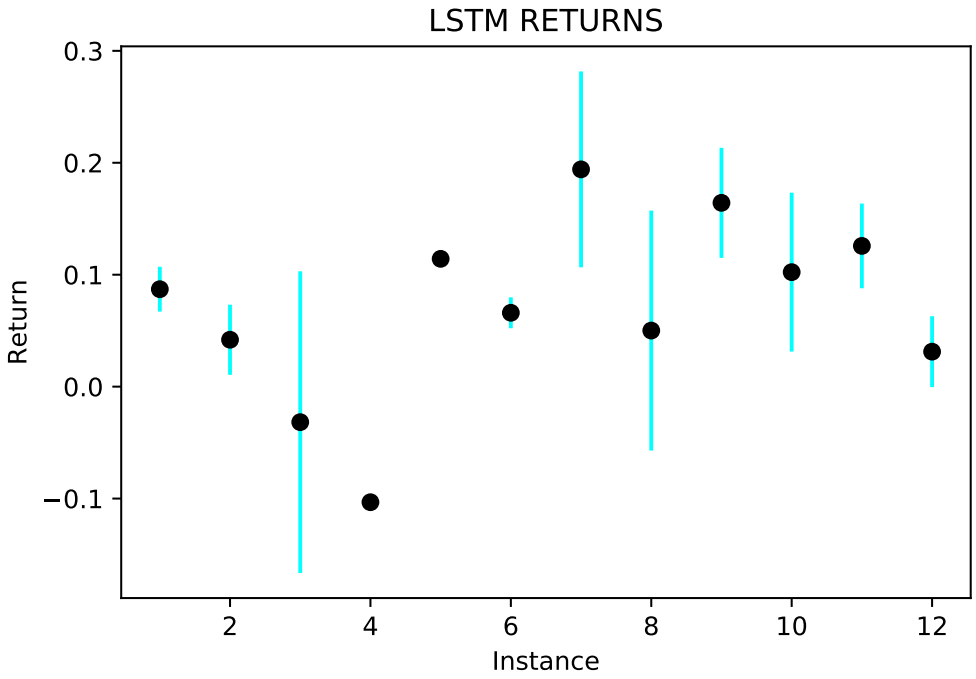


Figure 3.8: LSTM Returns with Std Error on Dataset2

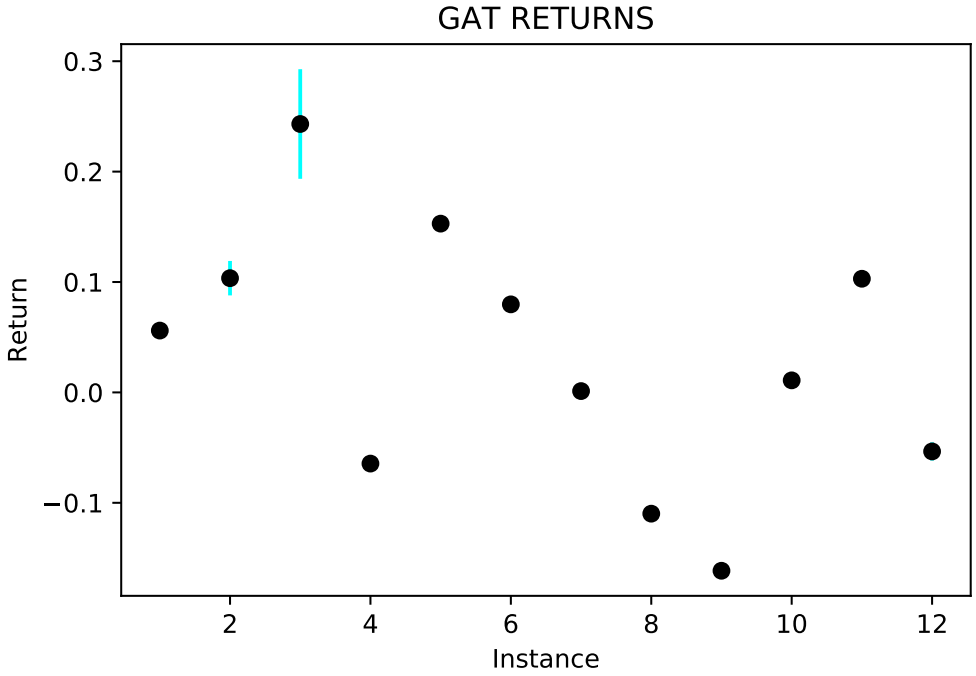


Figure 3.9: GAT Returns with Std Error on Dataset3

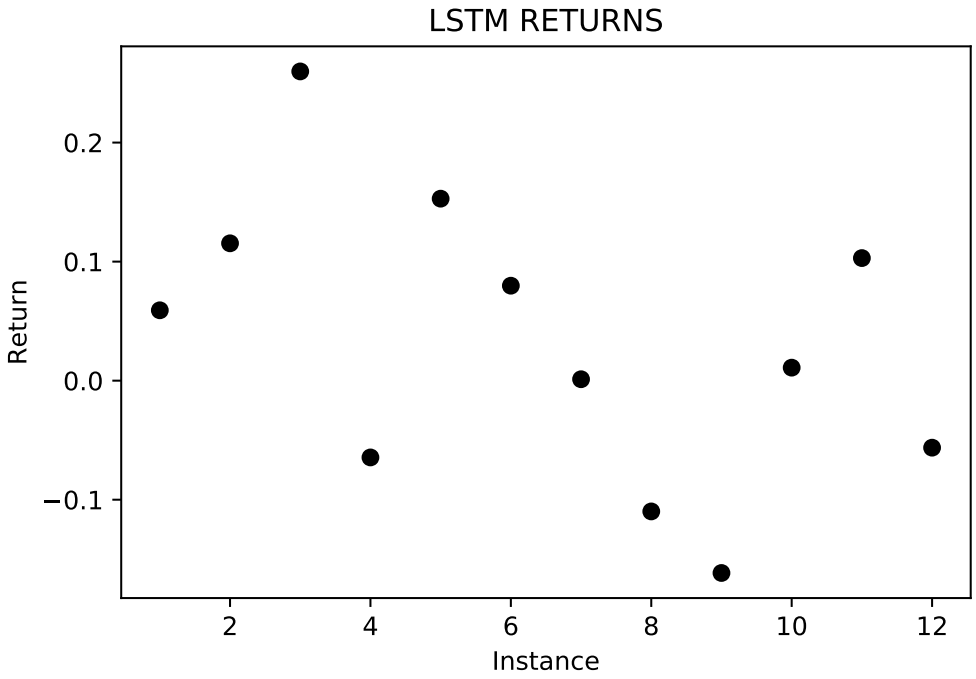


Figure 3.10: LSTM Returns with Std Error on Dataset3

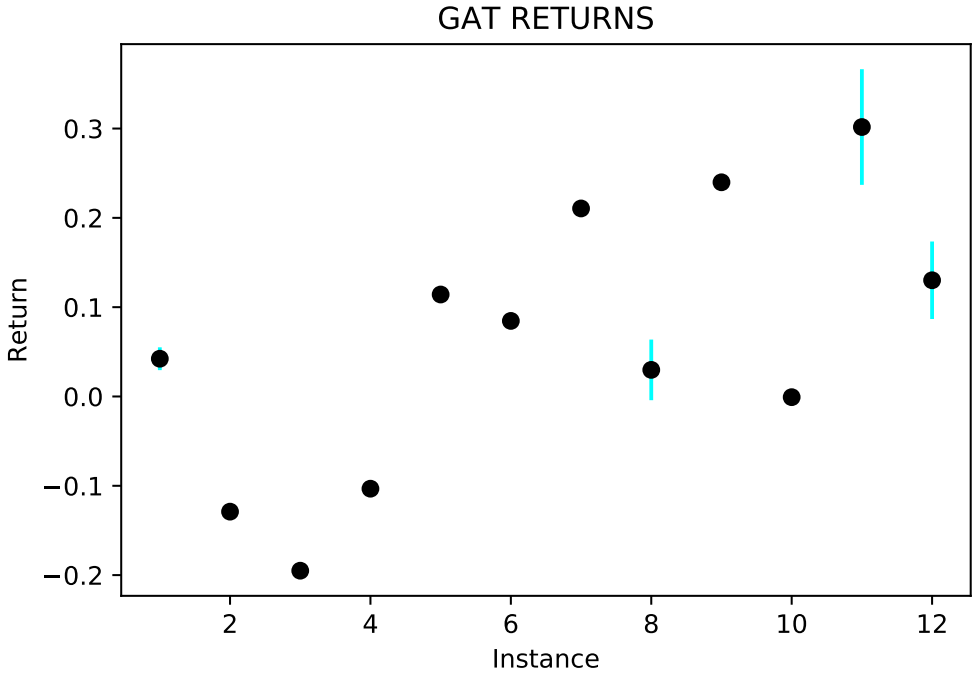


Figure 3.11: GAT Returns with Std Error on Dataset4

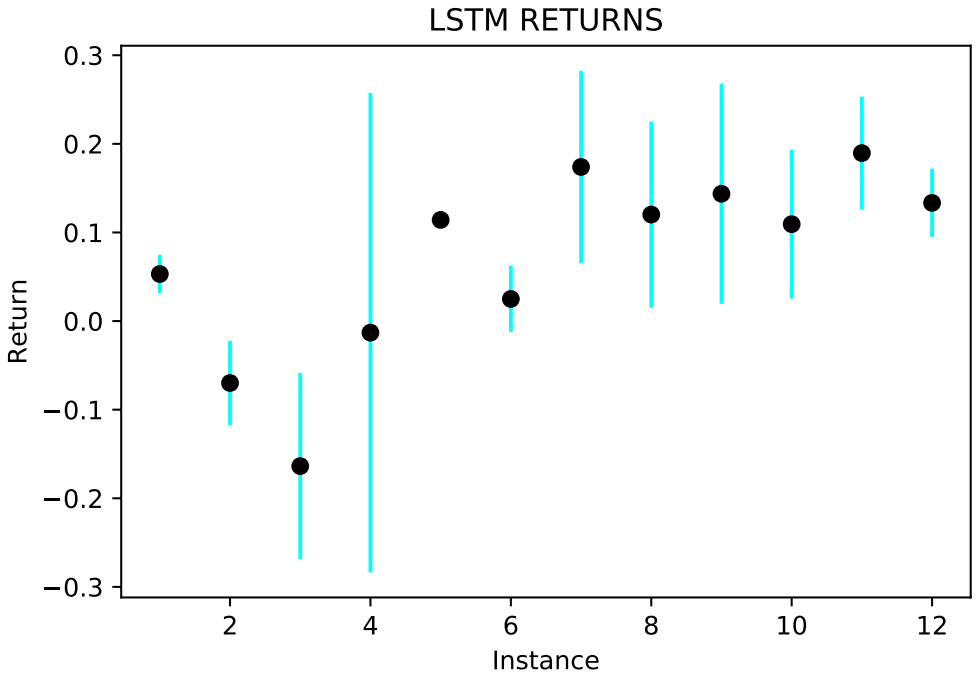


Figure 3.12: LSTM Returns with Std Error on Dataset4

Chapter 4

Subset Based Error Recovery

4.1 Introduction

Foundations of the ELM are rooted in function approximation theory. ELM uses a randomly generated network layer to obtain successful approximations on continuous functions [11]. This random layer is shown to be effective and efficient in terms of both accuracy and the computation complexity [40]. Leveraging this efficiency, we propose to use the ELM architecture in two possible ways: as a data denoising tool and a standalone algorithm.

Randomly generated weights are not only used in ELM's but also in dimensionality reduction, method of random projections, and compressed sensing due to their performance on accuracy/computation complexity trade-off. Furthermore, Johnson-Lindenstrauß Lemma allows ELMs to reduce the dimension of the problem for efficiency in computations while preserving the structure of the data [41], [42] with theoretical guarantees. The second layer of the ELM architecture introduces non-linear interactions of the features to improve the prediction capabilities of the system. At this point, using ELMs as a sparse error recovery and data denoising tool becomes highly efficient. Therefore, a robust ELM application along with an extendable data denoising method applicable to different machine

learning frameworks (especially NN's) is proposed in this section. In the following section, we describe the contribution of our approach to the literature, and proceed by describing our algorithm and give theoretical results. We conclude by comparing our algorithm with multiple ELM methods and show the effectiveness of data denoising with the comparison of multiple learning algorithms on both synthetic and real data.

4.2 Contribution

Under the non-linear CS framework, recovery guarantees of the proposed denoising algorithm are analyzed. We shall use these results to provide the denoising algorithm for non-linear ML problems by extending the robust regression analysis [4],[43] into a general denoising method applicable for neural networks, ELM and other ML algorithms. The motivation for applying such a denoising technique originates from the fact that sparse recovery methods are highly disturbed under heavy corruption. Denoising methods effective to address this issue are also expected to be effective in non-linear optimization problems. Furthermore, in light of the recent convex NN interpretations and following the studies on activation regions, we propose to use randomized activation regions to effectively evaluate the quality of the data points.

First, we describe how multiple layers of ELMs can be used to formulate sparse recovery problems for non-linear machine learning problems to denoise data from highly corruptive noise. Second, we provide a hard threshold based subset selection algorithm for an ELM application that outperforms robust loss functions and regularization methods [9], and derive convergence guarantees. One of the main contributions to the analysis performed on the convergence guarantees involves the JL Lemma and its relation with the RIP property which allows the former proofs to be still valid. To the best of our knowledge, only robust functions and their combinations with regularization methods were previously studied in the robust ELM literature. Therefore, a method providing robustness with a theoretical background is deemed a welcome and timely contribution to the literature.

4.3 Algorithm

The main reason for using ELM architecture in the data selection is to calculate the most important entries as fast as possible while capturing the possible non-linearities in the data. The preservation of the data after the random projections is a consequence of the JL Lemma:

Lemma 1 (JL). *Given $0 < \delta < 1$, a set X of n points in $\mathbb{R}^d$, and a number $k \geq \frac{3}{c\delta^2} \ln n$ for an appropriate positive constant c, there exists a random projection $f : \mathbb{R}^d \to \mathbb{R}^k$ which has the following property with probability at least $1 - \frac{3}{2}n$,*

$$\left| \|f(v_i) - f(v_j)\| - \sqrt{k} \|v_i - v_j\| \right| \leq \delta \sqrt{k} \|v_i - v_j\|$$

for all distinct pairs of points v_i and v_j in X.

Using this projection property, in a two-layer ELM, we can preserve the data structure in the first random layer, transform the data with a transfer function and create non-linearities in the second "calculated" layer which will be helpful to capture non-linearities. In the numerical tests in section 6, the addition of multiple random layers is studied to analyze the effectiveness of the method in capturing highly dependent data structures.

Remark 1. *The idea of random projections is similar to creating random activation patterns using randomized hyperplane arrangements in the convex neural network formulations. Using randomized activation patterns allow us to benefit from only a specific combination of fixed activation region from the data. Using this fixed activation region selected, we evaluate the performance of the data points. Finally, we select a useful subset of the data with respect to that evaluation.*

The hard thresholding step is introduced to the robust regression literature in [43] and extensively studied in [4],[10]. A similar idea can be extended to the proposed ELM architecture to obtain the best subset of the data which is not

corrupted for non-linear setting:

$$\begin{aligned} \min_b \quad & \|y - ZW_2 + b\|_2^2 \\ \text{s.t.} \quad & Z = \phi(XW_1) \\ & \|b\|_0 \leq k \end{aligned}$$

The first constraint above varies in the problem formulation depending on the number of layers that will be used in the denoising algorithm. The number of layers is viewed as a hyper-parameter depending on the structure of the non-linearities within the data. The general denoising problem is formulated as follows

$$\begin{aligned} \min \quad & \|y - ZW_n + b\|_2^2 \\ \text{s.t.} \quad & Z = \phi(\dots\phi(\phi(XW_1)W_2)\dots W_{n-1}), \\ & \|b\|_0 \leq k. \end{aligned}$$

First, the algorithm considers a θ-layered neural network where the $\theta - 1$ hidden layers are fixed and randomly generated. The function ϕ is selected as Leaky ReLU for theoretical analysis. However, ReLU, Sigmoid, tanh or any other injective transfer function could be used. The second layer output is obtained using least squares loss. The weight calculation is performed under the assumption $n \geq l$, otherwise the generalized inverse should be used as explained in the background section. Furthermore, the proposed method will be used as a pre-processing method in most applications, therefore the layer dimension is kept smaller than the data dimension with the given bounds of JL-Lemma to make the system work as fast as possible while preserving RIP property [41].

Algorithm 1: Subset Based Error Recovery (SuBER)

Input: X

Result: $\hat{w_{2_t}}$

e : residual error

initialization:

$W_1 = \mathcal{N}(0, I)$;

$t = 0$;

k: hyperparameter for subset size;

compute first layer: $Z = \phi(XW_1)$;

$w_2 = (Z^T Z)^{-1} Z^T y_{train}$;

while $t \leq$ *max iter* **do**

  calculate predictions: $\hat{y}_t = Z w_{2_t}$;

  select minimum k elements: $S_t := \text{mink}\left(\|y - \hat{y}_t\|\right)$ **calculate** w_{2_t}**:**

  $w_{2_t} = (Z_{S_t}^T Z_{S_t})^{-1} Z_{S_t}^T y_{S_t}$;

end

Algorithm 1 relies on the idea that one could disregard the indices where the error is large using IHT. This can be interpreted as an IHT method applied on X^T instead of X after the w_2 weights are calculated using the closed-form solution of the least-squares regression. Iteratively calculating the final layer weights and the best subset of data points allows us to converge to a denoised subset of the data. A more detailed explanation is provided in section 4.4.

In the algorithm, the hyperparameter for the subset size is required as a hyperparameter λ that would be used analogously in ℓ_1 or ℓ_2 regularization methods or the parameter γ that would be used in the Huber Loss. In addition, the number of random layers is adjusted as a hyperparameter.

The special case of the algorithm when $\theta = 2$ and $\phi =$ Leaky ReLU reduces the problem into a regression problem with a regular ELM architecture where the data subsets are selected dynamically. This special case is analyzed below as the ELM application and its performance on the existing ELM methods in the literature will be presented.

4.4 Theoretical Analysis

In order to provide the convergence guarantees, we use an approach similar to the convergence proof of the Robust Regression algorithm [4]. First, we recall the following definitions in order to use the RIP results. We use $\mathcal{B}_0(k)$ to denote the "ball" consisting of k-sparse vectors.

Definition 1 (RSC and RSS Properties,[44]). *A matrix $X \in R^{n \times p}$ is said to satisfy the α restricted strong convexity (RSC) property and the β restricted smoothness (RSS) property of order k if for all $w \in \mathcal{B}_0(k)$, we have*

$$\alpha_k \|w\|_2^2 \leq \frac{1}{n} \|Xw\|_2^2 \leq \beta_k \|w\|_2^2 .$$

The definition above is a more stringent version of the definition used in similar settings. Definition 3 below implies the same properties for any subset $K \subseteq X$.

It is important to note that the first layer of random weights in this study is a matrix instead of a vector as it usually is in the Compressed Sensing framework. However, one can assume to have the collection of vectors w to form W_1 in the NN and ELM cases.

Definition 2 (NSC and NSS Properties). *A non-linear transformation of a matrix $X \in R^{n \times p}$ is said to satisfy the α non-linear strong convexity (NSC) property and the β non-linear smoothness (NSS) property of order k if for all $w \in \mathcal{B}_0(k)$, we have*

$$\alpha_k \|w\|_2^2 \leq \frac{1}{n} \|\phi(Xw)\|_2^2 \leq \beta_k \|w\|_2^2 .$$

It was shown in [45] that if the function ϕ is injective, the necessary and sufficient conditions for NSC and NSS properties are satisfied. Similar to the subset version of the RSC and RSS, NSC and NSS imply the same properties for the subsets $K \subseteq X$.

In this study, injective transfer functions such as Leaky ReLU, Sigmoid, tanh

are used to satisfy this property. However, it is observed that the non-injective function ReLU performs well in practice.

Definition 3 (SSC, SSS, [4]). *A matrix $X \in \mathbb{R}^{n\times p}$ is α strong convex and β strong smooth of order k for $S \subseteq \{1,\dots,n\}$ with $|S| \leq k$ iff*

$$\begin{aligned}\alpha_k \leq \lambda_{\min}(X_S^T X_S) &\leq \|X_S\|_2^2 \\ &\leq \lambda_{\max}(X_S^T X_S) \leq \beta_k,\end{aligned}$$

where λ_{min} and λ_{max} are the minimum and the maximum eigenvalues for the given matrix and X_S is a matrix consisting rows of X corresponding to indices chosen from S.

Definition 4. *For any $w \in \mathbb{R}^l$ and $c_0 > 0$ the random variable $\|XW_1w\|_2^2$ is strongly concentrated about its expected value if*

$$P(\big|\|XW_1w\|_2^2 - \|w\|_2^2\big| \geq \epsilon \|w\|_2^2) \leq 2e^{-nc_0}$$

for $0 < \epsilon < 1$.

Lemma 2. *[46] ReLU and Leaky ReLU functions can be characterized as*

$$\phi(Xw) = D_w U\Sigma V^T w$$

where SVD of X is expressed as $X = U\Sigma V^T$ and D_w is a diagonal matrix with ReLU/Leaky ReLU coefficients on the diagonals.

For the Leaky ReLU activation function, the matrix D_w is invertible. This is not possible for ReLU when there are 0 entries on the diagonal.

Theorem 1 (L. ReLU Preserves SSC,SSS). *Let ϕ be the Leaky ReLU function and assume $\|XW_1w\|_2^2$ is strongly concentrated about its expected value. Then for all $w \in \mathcal{B}_0(k)$ and any $0 < \delta < 1$ we have*

$$\frac{1-\delta}{10}\|w\|_2 \leq \|Zw\|_2 \leq (1+\delta)\|w\|_2$$

with probability at least $1 - 2(12/\delta)^k e^{-c_0(\delta/2)n}$.

Proof. Since $\|XW_1w\|_2^2$ is strongly concentrated we have

$$(1-\delta)^2 \|w\|_2^2 \leq \|XW_1w\|_2^2 \leq (1+\delta)^2 \|w\|_2^2$$

for all $\delta \in (0,1)$ and $w \in \mathcal{B}_0(k)$ with the given probability [41]. Then, for the Leaky ReLU function ϕ, we have

$$\begin{aligned}\sigma_{\max}^2(D_w) \|XW_1w\|_2^2 &\geq \|D_wXW_1w\|_2^2 \\ &\geq \sigma_{\min}^2(D_w) \|XW_1w\|_2^2 ,\end{aligned}$$

where $\sigma_{\max}^2$ and $\sigma_{\min}^2$ are the maximum and minimum singular values for a given matrix. Combining these two results, we obtain

$$\begin{aligned}(1-\delta)^2\sigma_{\min}^2(D_w) \|w\|_2^2 &\leq \|Zw\|_2^2 \\ &\leq (1+\delta)^2\sigma_{\max}^2(D_w) \|w\|_2^2 .\end{aligned}$$

For any w, D_w is a diagonal matrix having entries 0.1 and 1's. Thus one can find global upper and lower bounds as desired. □

Remark 2. *For any piecewise linear transfer function with $\sigma_{\max} = \sigma_{\min} = 1$ at all pieces, SSC and SSS bounds are equivalent after the transformation*

$$\begin{aligned}\alpha \leq \sigma_{\min}^2(XW_1) &\leq \|\phi(XW_1)\|_2^2 \\ \leq \sigma_{\max}^2(XW_1) &\leq \beta.\end{aligned}$$

We note that the above bound is equivalent to the bound in Definition 3.

Hard Thresholding Step: The reduced formulation without the Gaussian noise, i.e, $y = \hat{y} + b$, is used to transform the problem into a hard thresholding problem properly. The hard thresholding step consists of the following optimization problem

$$\begin{aligned}\min_{b} \quad & \left\|(I - Z(Z^TZ)^{-1}Z^T)(y+b)\right\|_2^2 \\ \text{s.t.} \quad & \|b\|_0 \leq k,\end{aligned} \tag{HTS}$$

where $Z = \phi(XW_1)$ or in the more convoluted form of multiple ϕ functions. After the forward propagation, the equivalence of the residuals and the b value can be seen by the definition that $y - \hat{y} = b$. As a result the formulation above is simply reduced to selecting the observation indices with the largest b values.

Convergence Guarantees: For the proof we will combine the following relations: (Exactly the same as in [4] using the non-linear case Definition 2 instead of their Definition 1). We show that essentially the same convergence guarantees hold.

Theorem 2. *Let $Z \in \mathbb{R}^{n \times l}$ satisfy the SSC property at order k with parameter α_k and the SSS property at order $l-k$ with parameter β_{l-k} such that $\frac{\beta_{l-k}}{\alpha_k} < \frac{1}{1+\sqrt{2}}$. Let $w_2 \in \mathbb{R}^l$ be an arbitrary vector and $y = Zw_2 + b^*$ where $\|b^*\| \leq l-k$ is a sparse vector of possibly unbounded corruptions. Then Subset Based Regularization yields an ϵ-accurate solution $\|w_2 - w_{2_t}\| \leq \epsilon$.*

It is important to note that the convergence guarantee is exactly the same as the one required for the robust regression problem in [4]. In view of the proof provided in [4], we can use our Theorem 1 to obtain the convergence proof. Therefore, we have omitted the details.

Convergence Guarantees For Multiple Layers: The idea for wider networks follows a similar pattern using the previous result.

Remark 3. *If we work with θ layers defining*

$$Z = \phi(\phi(\dots\phi(XW_1)W_2)\dots W_\theta),$$

then we may apply Theorem 2 after assuming SSC and SSS properties for Z. Also, one can see that if we apply the steps in the proof of Theorem 1 we may obtain an SSS-SSC guarantee for such Z under mild conditions.

With each additional non-random layer, the minimum and the maximum singular values have an impact on the convergence guarantees on top of the structure of the original covariance matrix.

Remark 4. *In [46], the analysis shows that the magnitude of the eigenvalues diminishes with each successive layer. This suggests that the proposed algorithm converges for θ-Layers with very high probability if the 2-Layer ELM convergence condition holds, which is similar to the condition of the convergence of a robust regression algorithm [4].*

4.5 Results

In this section, we first present the performance of our algorithm when it is used as a denoising tool. Second, we deploy our algorithm as a stand-alone ELM algorithm and compare it with other robust ELM architectures in the literature. In both sections, the synthetic data $X \in \mathbb{R}^{n \times p}$ where $n > p$ is generated similarly to the tests conducted in [4] and [9]. For corruptions, we set $\|b\|_0 = 0.2n$ and randomly apply corruption to randomly selected indices with the randomly selected magnitudes of $\pm 5 \|y\|_\infty$. More specifically, initial tests were made on randomly generated observations $x_i \in \mathbb{R}^{1000}$ where $i \in \{1, \ldots, 2000\}$. The error size is selected as 400 and the entries are corrupted such that observation instances are selected at random and corrupted with additive corruption $b \sim Unif(-5 \|y\|_\infty, 5 \|y\|_\infty)$. The original outputs, y, are produced such that $y = Xw + b + \epsilon$ for linear case and $y = X^T Xw + Xw + b + \epsilon$, where $w \sim \mathcal{N}(0, 1)$ denotes the randomly generated weights. To be able to demonstrate the flexibility of the algorithm there is no additional sparsity pattern requirement enforced on the weight vector w in contrast to other studies. In the output function, Huber loss has been adopted for all of the models.

4.5.1 Data Denoising

After the original data is generated, two-layer, three-layer, and four-layer denoising methods are used to select the noiseless subset candidates to be used in the network. These models are trained and tested using Python Keras Library. Feed-forward networks with two hidden layers are used with neuron sizes equal

to 64 in each layer for the results. Tables 4.1 and 4.2 show the performance of the denoising algorithm where the data had low non-linearity and high nonlinearity, respectively.

Table 4.1: NN Result Part 1

Denoising Results for Low Non-Linearity		
Data	Loss	MSE
Original	45.947	1141.648
2-Layer Denoise	**13.202**	**72.372**
3-Layer Denoise	13.765	**72.372**
4-Layer Denoise	13.558	78.429
Original+Dropout	30.166	543.952

Table 4.2: NN Results Part 2

Denoising Results for High Non-linearity		
Data	Loss	MSE
Original	112.575	5211.718
2-Layer Denoise	93.261	3455.253
3-Layer Denoise	92.295	3455.25
4-Layer Denoise	**92.193**	**3349.651**
Original+Dropout	100.328	4090.479

The performance of the neural networks in Tables 4.1, 4.2 indicates that our denoising algorithm introduces a significant amount of robustness.

The performance of the multi-layer denoising does not appear to be affected by the number of layers in terms of the MSE. Synthetic data may not always be very suitable for deep learning, therefore the following tests were conducted on real data. *Boston Housing Prices* and *Diabetes* datasets are used for this purpose. The original dataset is corrupted using heavy noises as explained previously using the sparse noise vector $\|b\|_0 < 0.4n \approx 160$. In parallel with the previous tests, we take, $b \sim Unif(-5\|y\|_\infty, 5\|y\|_\infty)$.

Table 4.3: Boston Price Dataset Results

Denoising Results for Boston Pricing Dataset		
Data	Loss	MSE
Original	5.779	66.788
2-Layer Denoise	4.195	12.838
3-Layer Denoise	4.199	12.838
4-Layer Denoise	**4.131**	**12.562**
Original+Dropout	10.667	64.376

Table 4.4: Diabetes Dataset Results

Denoising Results for Diabetes Dataset		
Data	Loss	MSE
Original	38.335	126.748
2-Layer Denoise	37.239	103.539
3-Layer Denoise	37.236	103.539
4-Layer Denoise	**36.509**	**93.557**
Original+Huber+Dropout	36.609	112.05

In the tests, the models compared are benefiting from robust loss functions and regularization methods. The denoising method alone was able to surpass the competing methods. It was also observed that 2-Layer Denoise gives a better performance than the competitor robust methods. The differences in the layers create different initializations of the NN activation patterns. The results show that using the 2-Layer approach is also highly effective. Moreover, the same robust loss functions and regularization methods are applicable to denoised data theoretically, and better results could have been obtained if dropout was included in our algorithm tests. The main goal here is not to find the best possible fit for the real data, but to demonstrate the power of data-denoising even compared to relatively complex models. The results in Table 4.3 and the MSE results in Table 4.7 point out to similar outcomes obtained both from denoising and the proposed ELM algorithm.

4.5.2 ELM Method

In this section, the goal is to show that the ELM inheriting the denoising method similar to the robust linear regression [4] approaches remains prevalent compared to similar methods in the literature. For the tests, layer sizes are selected equal to 500 in order to benefit from the fast denoising due to the JL Lemma. MSE and Relative error results are displayed in Table 4.5 where Relative error is defined as $\frac{\|y_{test}-\hat{y}\|_2}{\|y_{train}\|}$ and MSE values are normalized with the observation number n. As an alternative measure, the corruption effect of the corrupted observations on the weights and the original weights are presented as the *corruption rate* below i.e. $\textit{corruption rate} = \frac{\|w_o-w\|_2}{\|w_o\|_2}$, where w_o denotes the least squares solution obtained through the y values before the corruption occurs. In the tests, 100 simulations were made for each method, and the average of these results is reported. The corruptions are set such that $\|b\|_0 = 0.2n$ for the Tables 4.5,4.6 and $\|b\|_0 = 0.4n$ for Table 4.7 as [9] and [4] perform tests up to this level of corruption.

Table 4.5: ELM Results

ELM Results for Linear Case			
Methods	MSE	Rel. Err.	Corr. Rate
ELM	1.5788	2.5679	3.2650
SuBER	**0.2550**	**1.0323**	0.6491
ELM+$\ell 2$	1.5870	2.5746	3.2745
RP+Bisquare	0.2846	1.0912	**0.5666**
IRLS+Huber	0.3192	1.1547	0.9080

Table 4.6: ELM Results

ELM Results for Non-Linear Case			
Methods	MSE	Rel. Err.	Corr. Rate
ELM	7.0031	5.3384	5.6655
SuBER	1.0626	2.0804	0.9662
ELM+$\ell 2$	7.0534	5.3575	5.6405
RP+Bisquare	**0.7061**	**1.6966**	**0.8745**
IRLS+Huber	1.2196	2.2286	1.4429

Table 4.7: ELM Results

ELM Results for Boston Price Dataset			
Methods	MSE	Rel. Err.	Corr. Rate
ELM	34.1896	0.4582	2.4161
SuBER	**14.8351**	**0.3076**	0.5088
ELM+$\ell 2$	56.8503	0.5936	4.0187
RP+Bisquare	1.4507e+03	3.0544	**0.2306**
IRLS+Huber	15.2007	0.3111	0.6351

Table 4.8: ELM Results

ELM Results for Diabetes Dataset			
Methods	MSE	Rel. Err.	Corr. Rate
ELM	615.0192	0.7787	1.1799
SuBER	**277.9764**	**0.5237**	**0.3165**
ELM+$\ell 2$	603.1095	0.7700	1.0743
RP+Bisquare	412.9272	0.6479	0.6624
IRLS+Huber	294.3530	0.5389	0.5048

The method *RP+Bisquare* is simply the random projections followed by robust regression library in MATLAB as the models are equivalent. From this analysis, it is apparent that our method is at least on-par with the competing methods, and even better under some of the categories. The linear model performance of the proposed model is slightly worse than the regular regression problem [4]. However, it is quite difficult to observe such linear data in real datasets. Even the Boston Price dataset is not completely linear even though it is one of the simplest datasets. Also, the increasing rate of corruption makes the convergence problematic for the robust regression libraries due to the corrupted entries. The proposed algorithm and the IRLS algorithm in [9] give on-par performances on the real dataset. As our method is originally proposed for denoising the data for different algorithms, a performance matching that of one of the most established robust ELM algorithms can be considered an encouraging result.

Figures 4.1,4.2,4.3 are plotted with respect to the increasing corruption size for the linear model, and the rest of the figures concern the non-linear model.

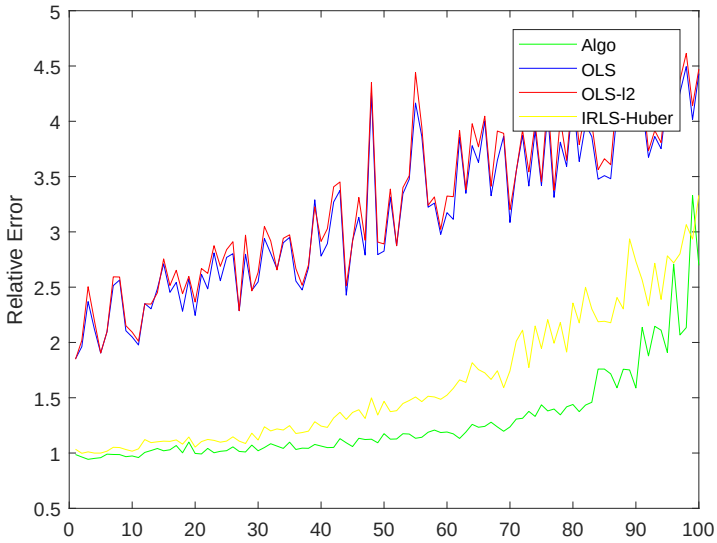


Figure 4.1: Relative Err Lin

The corrupted index number was increased by 8 in each iteration. An average of 10 different runs per method is taken to smooth the effect of the random layer. In each figure, the dominance of our algorithm is visible. The regular OLS and ℓ_2 regularized OLS methods fail to adapt to the corruptions as expected. Commonly used Huber loss respectively performs better than the OLS. However, our algorithm performs better compared to the results of Huber loss as well. The Huber Loss used in these tests is borrowed from [9] IRLS-ℓ_2-Huber algorithm as it is one of the most competitive algorithms in the literature. In practice, Huber is the most common loss among the ML tools and libraries. Therefore it is the most meaningful loss selection for comparison.

The computational complexity varies with respect to the convergence of the inner step. In our algorithm, the inner step enjoys the property of "quick" steps as discussed in [4]. Since in each update weights are calculated with respect to the least squares solution without the need of a gradient method, the convergence of the weights occurs in very few iterations. In other robustness studies [9], proposed algorithms involving iteratively re-weighted least squares methods have a similar inner step. As a result, the time complexity of the proposed algorithm is comparable to the available methods in the literature. The advantages of our method can be summarised as follows:

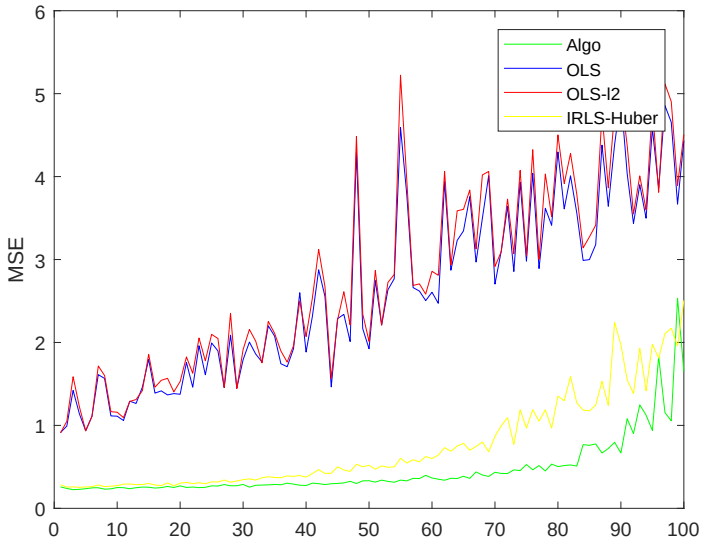


Figure 4.2: MSE Lin

1. Effective under heavy corruptions in terms of magnitude
2. Scales well with the corruption percentage
3. Hard-Threshold is simple to implement
4. Theoretically compatible with all injective activation functions
5. Time complexity increases with respect to the inner loop. Update method converges in 5-10 iterations
6. Fast in large scale data due to random projections (JL Lemma) with respect to the regular regression variant [4],[43].

4.6 Conclusion

We have proposed an ELM architecture that can be used for data denoising and robust ELM regression problems. In the light of recent developments in convex neural networks, we have advocated that creating randomized activation patterns using ELMs would be a practical approach to evaluate the performance of the data points. To evaluate the data points, we cast the denoising problem as a

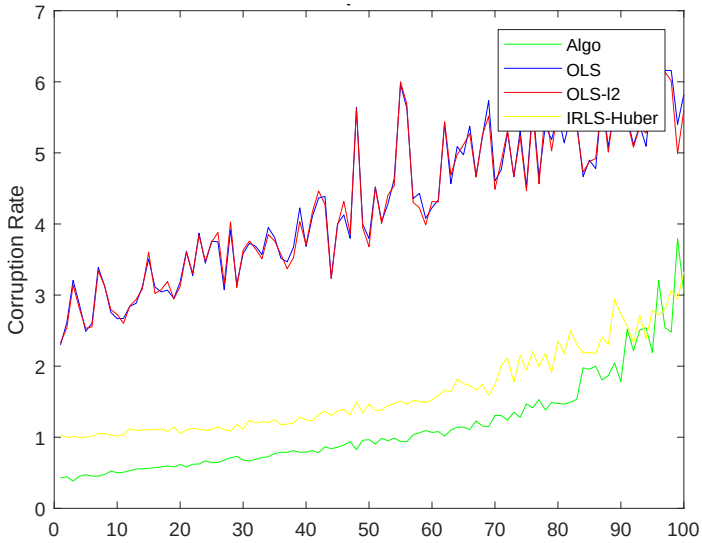


Figure 4.3: Corr Lin

sparse recovery problem over the data points. This allows us to give theoretical guarantees for our algorithm, a feature that is rarely encountered in the literature. Furthermore, the denoised data obtained from our method can be fed into any NN architecture to benefit from the robustness properties of certain NNs. Therefore, the results we have obtained using our pre-processing step can be further improved when paired with proper NN architectures. In the second part of the study, we have shown that the proposed method can also be used as a standalone robust ELM architecture. Our numerical results indicated that both the denoising and standalone ELM methods achieve better performance compared to their competitors. In future work, the denoising properties of a network will be analyzed when the first layers are not random but trainable.

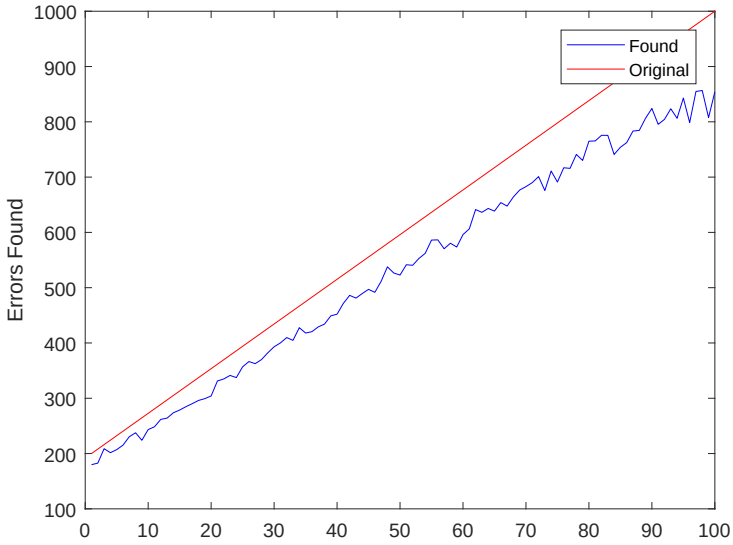


Figure 4.4: err found lin

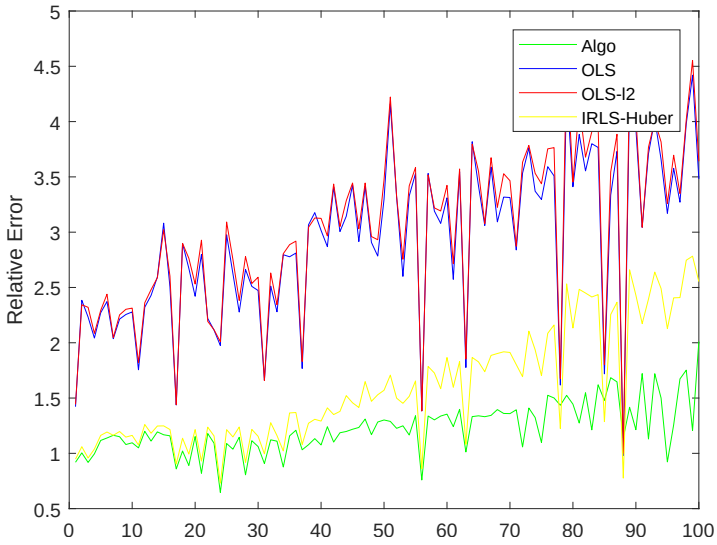


Figure 4.5: relative err nonlin

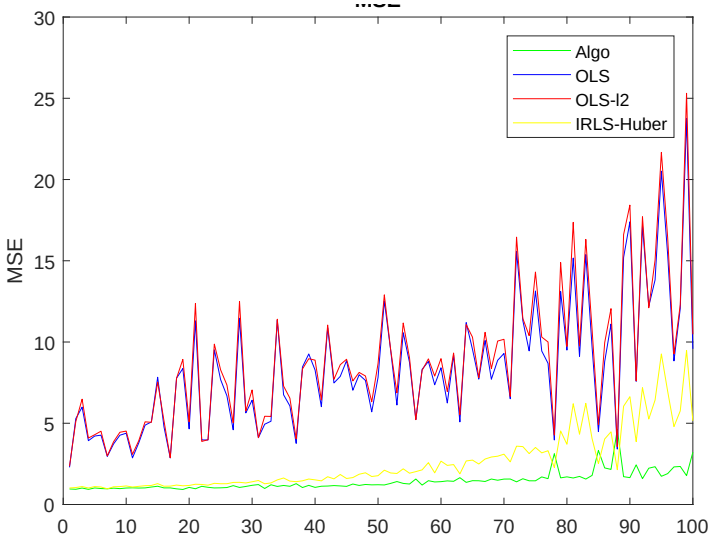


Figure 4.6: MSE Nonlin

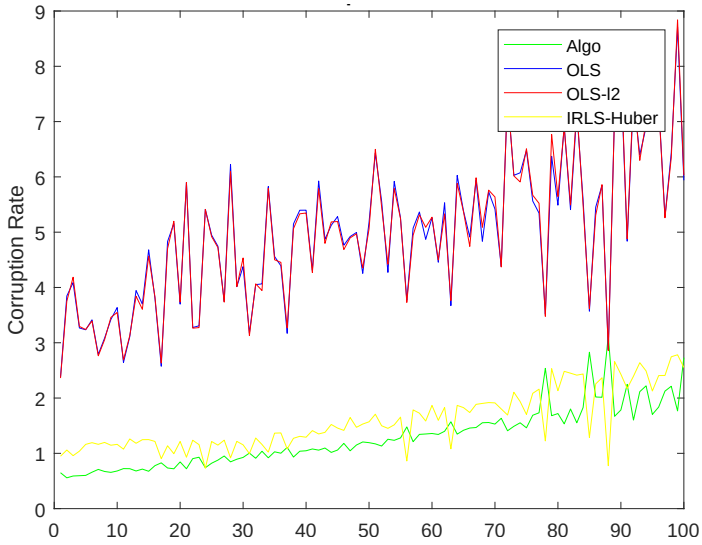


Figure 4.7: corr nonlin

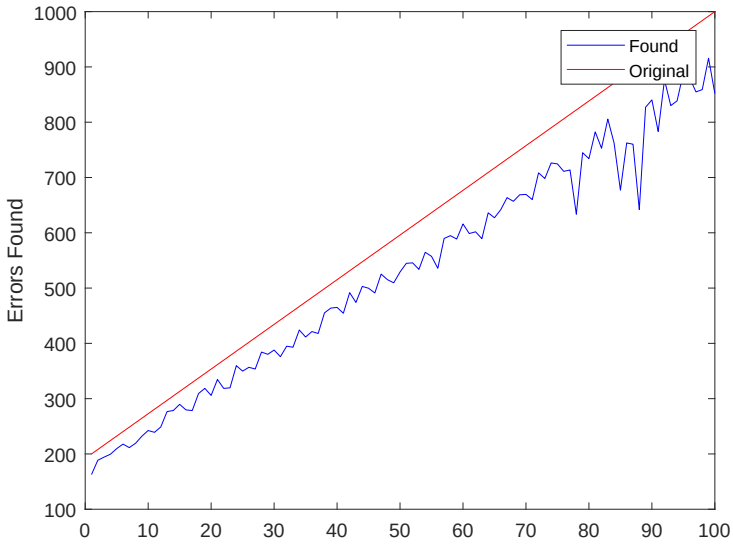


Figure 4.8: err found nonlin

Chapter 5

Conclusion

We have compiled two research studies in this thesis on the fields of deep learning and optimization. We merge ideas from both of these literature to improve the state-of-the-art methods in portfolio optimization and robust extreme learning machines.

In our first study, we explore a deep learning approach to a classical portfolio optimization problem. We combine ideas from classical optimization literature, such as Graphical Lasso, to obtain an adjacency matrix of a market. This graph reveals the connections between companies, which is further used in Graph Neural Networks to obtain better allocations for portfolio optimization problems compared to the recurrent models and model-based optimization frameworks. We focus on obtaining robust solutions where the datasets are relatively small and evaluate the algorithmic in deep portfolio models. We combine state-of-the-art graph neural network models with deep learning techniques to obtain architectures that perform better than the state of the art architectures.

In our second work, we proposed to use a randomized approach to have a robust extreme learning machine framework as well as a data denoising tool. The idea is to solve a sparse recovery problem, not in the columns of the data matrix, but in the rows of the matrix. This allows us to search and recover heavy

corruptive noise present in the system. We then propose two ways this method can be used along with theoretical guarantees on recovery. When our proposed algorithm is used as a standalone robust ELM, we surpass robust ELM methods in the literature. Furthermore, we show that our data-denoising tool is effective when used with a deep learning problem.

Finally, it was observed that both optimization and deep learning methods could be merged to improve the state-of-the-art methods in various lines of literature. Optimization tools are effective and efficient in many cases when used as a data processing tool. Deep models are also highly effective when adjusted to solve classical optimization problems. Integration of ideas from both optimization and deep learning paves the way for further improvement in portfolio optimization, data denoising, and potentially, in many other fields.

Bibliography

[1] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, pp. 77–91, Mar. 1952.

[2] S. M. Bartram, J. Branke, and M. Motahari, *Artificial intelligence in asset management*. No. 14525, CFA Institute Research Foundation, 2020.

[3] Z. Zhang, S. Zohren, and S. Roberts, "Deep learning for portfolio optimization," *The Journal of Financial Data Science*, vol. 2, p. 8–20, Aug 2020.

[4] K. Bhatia, P. Jain, and P. Kar, "Robust regression via hard thresholding," *CoRR*, vol. abs/1506.02428, 2015.

[5] B. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comp.*, vol. 24, no. 2, pp. 227–234, 1995.

[6] B. Moghaddam, Y. Weiss, and S. Avidan, "Generalized spectral bounds for sparse lda," Tech. Rep. TR2006-046, MERL - Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, June 2006.

[7] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[8] T. Blumensath and M. E. Davies, "Iterative hard thresholding for compressed sensing," *Applied and Computational Harmonic Analysis*, vol. 27, no. 3, pp. 265–274, 2009.

[9] K. Chen, Q. Lv, Y. Lu, and Y. Dou, "Robust regularized extreme learning machine for regression using iteratively reweighted least squares," *Neurocomputing*, vol. 230, pp. 345 – 358, 2017.

[10] A. S. Suggala, K. Bhatia, P. Ravikumar, and P. Jain, "Adaptive hard thresholding for near-optimal consistent robust regression," *CoRR*, vol. abs/1903.08192, 2019.

[11] G. Huang, Q. Zhu, and C. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489 –501, 2006.

[12] O. F. Alcin, A. Sengur, S. Ghofrani, and M. C. Ince, "Ga-selm: Greedy algorithms for sparse extreme learning machine," *Measurement*, vol. 55, pp. 126 – 132, 2014.

[13] E. Tsakonas, J. Jaldén, N. D. Sidiropoulos, and B. Ottersten, "Convergence of the huber regression m-estimate in the presence of dense outliers," *IEEE Signal Processing Letters*, vol. 21, no. 10, pp. 1211–1214, 2014.

[14] D. Akkaya and M. Pınar, "Minimizers of sparsity regularized huber loss function," *Journal of Optimization Theory and Applications*, vol. 187, no. 1, p. 205–233, 2020.

[15] Z. Shao and M. Er, "An online sequential learning algorithm for regularized extreme learning machine," *Neurocomputing*, vol. 173, 08 2015.

[16] M. Pilanci and T. Ergen, "Neural networks are convex regularizers: Exact polynomial-time convex optimization formulations for two-layer networks," in *ICML*, 2020.

[17] Y. Bai, T. Gautam, Y. Gai, and S. Sojoudi, "Practical convex formulation of robust one-hidden-layer neural network training," 2021.

[18] B. Lim and S. Zohren, "Time-series forecasting with deep learning: a survey," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, p. 20200209, Feb 2021.

[19] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," 2018.

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

[21] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2018.

[25] D. Grattarola and C. Alippi, "Graph neural networks in tensorflow and keras with spektral," 2020.

[26] F. M. Bianchi, D. Grattarola, L. Livi, and C. Alippi, "Graph neural networks with convolutional arma filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, p. 1–1, 2021.

[27] Z. Zhang, S. Zohren, and S. Roberts, "Deep reinforcement learning for trading," 2019.

[28] W. Wang, W. Li, N. Zhang, and K. Liu, "Portfolio formation with preselection using deep learning from long-term financial data," *Expert Systems with Applications*, vol. 143, p. 113042, 2020.

[29] C. Zhang, Z. Zhang, M. Cucuringu, and S. Zohren, "A universal end-to-end approach to portfolio optimization via deep learning," 2021.

[30] R. Kim, C. H. So, M. Jeong, S. Lee, J. Kim, and J. Kang, "Hats: A hierarchical graph attention network for stock movement prediction," 2019.

[31] F. Feng, X. He, X. Wang, C. Luo, Y. Liu, and T.-S. Chua, "Temporal relational ranking for stock prediction," *ACM Transactions on Information Systems*, vol. 37, p. 1–30, Apr 2019.

[32] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," 2020.

[33] J. Yoon, J. Jordon, and M. van der Schaar, "PATE-GAN: Generating synthetic data with differential privacy guarantees," in *International Conference on Learning Representations*, 2019.

[34] H. Ni, L. Szpruch, M. Wiese, S. Liao, and B. Xiao, "Conditional sig-wasserstein gans for time series generation," Jun 2020.

[35] D. Hallac, Y. Park, S. Boyd, and J. Leskovec, "Network inference via the time-varying graphical lasso," 2017.

[36] G. Tegner, "Recurrent neural networks for financial asset forecasting," Master's thesis, KTH, Mathematical Statistics, 2018.

[37] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2016.

[38] Çağın Ararat, F. Cesarone, M. Çelebi Pınar, and J. M. Ricci, "Mad risk parity portfolios," 2021.

[39] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.

[40] G. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 513–529, 2012.

[41] R. Baraniuk, M. Davenport, R. DeVore, and M. Wakin, "A simple proof of the restricted isometry property for random matrices," *Constructive Approximation*, vol. 28, pp. 253–263, 12 2008.

[42] T. Cheng, "Restricted conformal property of compressive sensing," *CoRR*, vol. abs/1408.5543, 2014.

[43] Y. Chen, C. Caramanis, and S. Mannor, "Robust sparse regression under adversarial corruption," in *Proceedings of the 30th International Conference on*

Machine Learning (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 774–782, PMLR, 17–19 Jun 2013.

[44] P. Jain and P. Kar, "Non-convex optimization for machine learning," *Found. Trends Mach. Learn.*, vol. 10, p. 142–336, Dec. 2017.

[45] J. Yi and G. Tan, "Nonlinear compressed sensing based on composite mappings and its pointwise linearization," *CoRR*, vol. abs/1506.02212, 2015.

[46] S. Dittmer, E. J. King, and P. Maass, "Singular values for relu layers," *CoRR*, vol. abs/1812.02566, 2018.