UC Berkeley

UC Berkeley Electronic Theses and Dissertations

Title

Application-Driven Development of Computational Tools and Algorithms for Machine Learning and Mean-Field Games

Permalink

https://escholarship.org/uc/item/8b39r1m1

Author

Tajrobehkar, Mahan

Publication Date

2023

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

# Application-Driven Development of Computational Tools and Algorithms for Machine Learning and Mean-Field Games

By

Mahan Tajrobehkar

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Industrial Engineering and Operations Research

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Xin Guo, Chair
Professor Zeyu Zheng
Professor Anant Sahai

Summer 2023

Application-Driven Development of Computational Tools and Algorithms for Machine Learning and Mean-Field Games

Copyright 2023
by
Mahan Tajrobehkar

Abstract

Application-Driven Development of Computational Tools and Algorithms for Machine Learning and Mean-Field Games

by

Mahan Tajrobehkar

Doctor of Philosophy in Engineering - Industrial Engineering and Operations Research

University of California, Berkeley

Professor Xin Guo, Chair

In today's rapidly evolving technological landscape, the development and advancement of computational tools and algorithms have become paramount across a wide range of research fields. This holds particularly true in various domains of computational mathematics, encompassing areas such as machine learning, optimization, and algorithmic game theory. The computational tools serve as essential enablers, empowering researchers and practitioners by facilitating efficient modeling, analysis, and prediction. Algorithms are essential components of computational tools, which provide instructions for data processing, pattern recognition, and decision-making.

This dissertation focuses on developing computational tools and algorithms for specific applications in the interconnected fields of optimization, machine learning (ML), and mean-field games (MFGs). First, to address the absence of a comprehensive computational tool for MFGs, we present `MFGLib`, an open-source Python library designed to provide a user-friendly and customizable interface for solving Nash equilibria in generic MFGs. Second, we demonstrate that the search for Nash equilibria in MFGs and various ML problems can be formulated as non-convex optimization problems, where the presence of saddle points significantly impedes the effectiveness of gradient descent algorithm and its variants. To help optimization algorithms escape saddle points efficiently, we introduce a novel perturbation mechanism based on the dynamics of vertex-repelling random walk. This leads to the development of two new algorithms, perturbed gradient descent adapted to occupation time (PGDOT) and its accelerated version (PAGDOT). Theoretical guarantees for these algorithms are established, and through extensive numerical experiments, we showcase their superiority over several state-of-the-art optimization methods. Last, we explore a relatively independent machine learning task—detecting overutilization and fraud in healthcare. We focus on developing an ensemble model based on Stacked Generalization (stacking) to detect overutilization in Medicare within the field of Ophthalmology. Our results highlight the superiority of the stacking ensemble model over traditional ML models in accurately distinguishing overutilizing ophthalmologists from non-fraudulent ones.

To my parents, for ensuring every educational opportunity was available to me.

# Contents

# List of Figures

# List of Tables

# Acknowledgments

I have many people to thank for their guidance and support of this research and my academic career.

First, I would like to express my deepest gratitude to Xin Guo, my advisor, for her exceptional mentorship, unwavering patience, and profound expertise throughout my research journey. Her guidance and support have been invaluable, particularly during the times when I felt uncertain or lost. Xin's dedication to my growth as a researcher was evident in her efforts to find research problems that aligned with my interests, even if they were not directly related to her own research pursuits. Her introduction of intriguing and challenging problems in the fields of optimization, machine learning, and mean-field games broadened my horizons. I am truly fortunate to have her as my advisor.

Second, I would like to thank my collaborators and committee members. I consider myself fortunate to have had the opportunity to work in a stimulating research environment surrounded by exceptional individuals. Many thanks to my collaborators, Wenping Tang, Jiequn Han, Anran Hu, Junzi Zhang, Matteo Santamaria, Scott Lee, and Varun Shravah. As members of my qualifying exam and dissertation committee, Zeyu Zheng and Anant Sahai provided me with helpful feedback in my research, and I am thankful for their support.

I would also like to thank many more individuals who have greatly enriched my experience at UC Berkeley. Special shout-out to Yoon Lee and Yuhao Ding for co-founding the "Stronger" group. My gratitude extends to our research group members and my peers at the IEOR department: Yusuke Kikuchi, Haotian Gu, Haoyang Cao, Renyuan Xu, Xinyu Li, Salar Fattahi, Mahbod Olfat, Reza Mohammadi-Ghazi, Han Feng, Cedriz Jozs, Armin Askari, SangWoo Park, Igor Molybog, Tomas Valencia, Marie (Pelagie) Elimbi, Julie Mulvaney-Kemp, Yann Fraboni, Vanshika Bansal, Ruijie Zhou, Ilgin Dogan, Erik Bertelli, Ruojie Zeng, Michael Murray, Jehum Cho, and Bhaskar Chaturvedi.

Finally, I would like to express my profound appreciation for my parents and my dear Saghi, whose unwavering love, support, and encouragement have been a constant source of strength throughout my academic journey.

# Chapter 1

# Introduction

In today's era of rapid technological advancements and data-driven decision-making, the development and advancement of computational tools have emerged as a top priority across diverse fields of research. This holds particularly true in various domains of computational mathematics, encompassing areas such as machine learning, optimization, and algorithmic game theory. These tools serve as essential enablers, empowering researchers and practitioners by facilitating efficient modeling, analysis, and prediction. Computational tools heavily rely on algorithms to fulfill their purpose effectively. Algorithms drive the functionality and capabilities of computational tools. They determine how data is processed, patterns are identified, and insights are extracted. Algorithms enable automation, streamline computations, and facilitate efficient decision-making. Advancing and optimizing algorithms within computational tools is vital for enhancing performance, accuracy, and scalability.

This dissertation focuses on the development of computational tools and algorithms for specific applications in optimization, machine learning (ML), and mean-field games (MFGs). These three fields are interconnected and they leverage each other's concepts and techniques. Optimization plays a fundamental role in ML. Optimization algorithms are used to train ML models by minimizing a loss function and finding optimal model parameters. Similarly, in MFGs, optimization techniques are employed to solve for Nash equilibria and find optimal strategies for a large population of interacting agents. Additionally, in recent years, the relationship between ML and MFGs has become a fruitful area of research. ML techniques like deep learning and reinforcement learning can be applied to solve MFGs, while mean-field techniques provide insights into neural networks and multi-agent reinforcement learning.

Each of the following chapters presents a unique computational tool or algorithm. Chapter 2, **"MFGLib: A Library for Mean-Field Games,"** introduces `MFGLib`, an open-source Python library dedicated to solving Nash equilibria for generic MFGs with a user-friendly and customizable interface, aiming at promoting both applications and research of MFGs. This chapter discusses the design principles and essential features of the library, highlighting its distinguishing aspects compared to other computational tools available for MFGs. Additionally, a concise mathematical background on discrete finite-horizon MFGs as well as the comprehensive documentation of `MFGLib` are provided in the corresponding appendices.

Chapter 3, **“Escaping Saddle Points Efficiently with Occupation-Time-Adapted Perturbations,”** presents a novel perturbation mechanism inspired by the dynamics of vertex repelling random walk. This mechanism is integrated into the framework of perturbation-based optimization algorithms, enhancing their ability to escape saddle points more effectively. The chapter introduces two new algorithms, namely perturbed gradient descent adapted to occupation time (PGDOT) and its accelerated version (PAGDOT), which are developed based on this perturbation mechanism. The subsequent sections of this chapter are dedicated to establishing the theoretical guarantees for these algorithms and conducting numerical experiments to showcase the advantages of the new perturbation mechanism. The numerical experiments encompass a range of problems, including finding Nash equilibria in MFGs and image classification tasks with deep learning models.

Chapter 4, **“Leveraging Stacked Generalization to Effectively Detect Overutilization in Medicare,”** focuses on the problem of detecting overutilization and fraud in Medicare within the field of Ophthalmology. The primary objective is to develop an ML model capable of accurately distinguishing fraudulent physicians from non-fraudulent ones. The chapter proceeds by detailing the creation and preprocessing of a labeled dataset specifically designed for this purpose. Subsequently, a comparative study of various ML methods is conducted to evaluate their performance. The results highlight the superiority of the Stacked Generalization (stacking) ensemble method over traditional approaches. Additionally, the chapter extends the analysis beyond model performance by employing the stacking ensemble model to estimate essential overutilization statistics within the field of ophthalmology.

We remark that each chapter is designed to be self contained—they each contain background, motivation, and key definitions for the topic at hand—and can be read independently, if desired.

## 1.1 Related Publications

This dissertation incorporates content that is currently undergoing the review process at scientific journals. The preprints are available at the following locations:

- **Chapter 2**: Guo, Xin, Anran Hu, Matteo Santamaria, Mahan Tajrobehkar, and Junzi Zhang. "MFGLib: A Library for Mean-Field Games." arXiv preprint arXiv:2304.08630 (2023).

- **Chapter 3**: Guo, Xin, Jiequn Han, Mahan Tajrobehkar, and Wenpin Tang. "Escaping Saddle Points Efficiently with Occupation-Time-Adapted Perturbations." arXiv preprint arXiv:2005.04507 (2020).

# Chapter 2

# MFGLib: A Library for Mean-Field Games

Large population games are ubiquitous in real-world problems. Examples include multiplayer online role-playing games [68], high frequency trading [88], Ad auctions [29] and sharing economy [53], to name just a few. As the number of players in the game grows, however, the computational complexity grows exponentially and it becomes notoriously hard to solve such problems. Mean-field games, pioneered by the seminal work of [62] and [84], is a relatively recent branch of mathematics, which attempts to understand the limiting behavior of systems involving very large number of rational agents which play differential games under partial information and symmetry assumptions.

A Mean-Field Game (MFG for short) refers to a game involving an infinite number of indistinguishable players who exhibit similar behavior, i.e., they are symmetric. The individual player identities are not crucial to consider as part of their state. Since the player count is infinite, it becomes possible to represent the individual players by their distributions across the state (and sometimes action) space. By focusing on the distribution of the population, we can examine the interaction between a representative player, randomly selected from the population's distribution, and the population's distribution itself. The ultimate objective is to determine a Nash equilibrium (NE), a policy from which no player will unilaterally deviate. Seeking an NE policy assumes that all players are perfectly rational, aiming to maximize their own rewards (or minimize their costs). It has been shown that the NE policy of a mean-field game is an $\epsilon$-NE of the corresponding $N$-player game [122], with $\epsilon = O(\frac{1}{\sqrt{N}})$ [62]. In practice, games with small $N$ on the order of tens can be well approximated by MFGs [52, 81, 19].

Recently, the literature of MFGs has experienced an exponential growth both in theory and in practice. In particular, there has been a surge of interests in the computation and learning of NEs in MFGs, with wide applications including bid recommendation [52], population dynamics [30], high frequency trading [88], crowd motion modeling [1], product pricing [51], autonomous vehicles [61], dynamic routing [19], animal behavior simulation [114], energy production and management [4], and security and communication. [99]

In MFGs, the NE policies are typically characterized through optimality conditions, which manifest as a coupled system of forward-backward equations. The forward equation describes the progression of the population distribution, while the backward equation represents the evolution of the value function, indicating the utility of policy for a representative player. In the continuous time and continuous space setting, the equations can take the form of either partial differential equations (PDEs) or stochastic differential equations (SDEs) of the McKean-Vlasov type. The choice between these two types depends on whether one adopts an analytical approach or a probabilistic approach. [14, 23] e discrete time case is close to the framework of Markov Decision Processes. We refer to [85] for a detailed breakdown of the discrete time MFGs. Our focus will be on discrete finite-horizon MFGs with finite state and action spaces.

Iterative methods are commonly employed to compute NE solutions. These methods begin with an initial policy and population distribution (also called mean-field distribution) and iteratively update them until they reach an equilibrium. The update of the mean field involves utilizing the population distribution induced by the current policy. The policy update can be executed in two distinct manners. It can either be determined by computing the best response against the mean field or by the evaluation of the previous policy. Fictitious Play [113], GMF-V [52], Prior Descent [32], and Online Mirror Descent [112] are a few examples of the iterative methods. An alternative approach involves employing an optimization framework to determine the NE solutions. [50] demonstrate that the task of finding the NE solutions for a general class of discrete-time MFGs is equivalent to solving an optimization problem known as MF-OMO. This optimization problem entails bounded variables and simple convex constraints. Notably, the authors do not assume the uniqueness of the NE solution or rely on contractivity or monotonicity conditions, which are typically assumed in the iterative methods.

Despite the extensive utilization of MFGs across various applications and the availability of a diverse range of computational methods and algorithms, there remains a significant gap: the absence of a unified computational tool that caters to researchers and practitioners interested in MFGs. This lack of a comprehensive tool hinders the accessibility and convenience for users who seek to explore and implement different MFG algorithms and work with various environments. Implementations of the MFG environments and algorithms are currently by and large provided mainly for paper reproducibility or experimental/internal-use purposes. Consequently, researchers and practitioners are faced with the challenge of navigating and integrating disparate tools and resources, thereby impeding the seamless development and application of MFG methodologies. Addressing this gap and providing a unified computational tool would enhance the efficiency and effectiveness of MFG research and facilitate its practical implementation in real-world scenarios.

This chapter introduces `MFGLib`, an open-source Python library dedicated to solving NEs for generic MFGs with a user-friendly and customizable interface, aiming at promoting both applications and research of MFGs. On one hand, it facilitates the creation and analysis of arbitrary user-defined MFG environments with minimal prior knowledge on MFGs. On the other hand, it serves as a modular and extensible code base for the community to easily prototype and implement new algorithms and environments of MFGs as well as their variants and generalizations.[1] The package is distributed under the MIT license and the source code and documentation can be found at `https://github.com/radar-research-lab/MFGLib/`.

The rest of this chapter is organized as follows. Section 2.1 reviews the related work. Section 2.2 briefly overviews the `MFGLib` package and discusses its design and key features. Section 2.3 discusses the future work. We refer the readers to Appendix for a concise review of the mathematical framework involved in solving finite-horizon MFGs, as well as the detailed documentation of `MFGLib`.

## 2.1 Related Work

Various libraries have been developed for $N$-player games, such as `QuantEcon` [133], `Nashpy` [140] and `ilqgames` [42]. In contrast, only very limited tools focus on MFGs and are mainly for experimental and internal use, and hence not suitable for general users with their own customized environments and problems.

Among these very few existing MFG libraries, `OpenSpiel` [83], a collection of environments and algorithms for research in reinforcement learning and planning in games, is the closest one to `MFGLib`. `OpenSpiel` has dedicated a module to MFGs implementing several environments and algorithms. However, it lacks customizability and a user-friendly API. In fact, according to its documentation, their code is still experimental and is only recommended for internal use. Other MFG libraries such as `gmfg-learning` [75] and `entropic-mfg` [144] [2] are mainly developed to support the experimental results of a particular paper and are not suitable for general MFG experiments.

## 2.2 Brief Overview of `MFGLib`

`MFGLib` is a handy tool for solving the ($\epsilon$)-NE of generic user-defined MFG environments.[3] On one hand, it provides an off-the-shelf environment creator, with which the users can create their environments by simply inputting the reward functions, transition functions and some basic problem data (*e.g.*, number of states and actions) without additional problem reformulation. On the other hand, `MFGLib` is equipped with different algorithms for users to choose from to solve their environments, which output a sequence of candidate NE policies and log the progress of their exploitability values. In what follows, we briefly discuss the design and features of this library.

[1]A pre-release version of `MFGLib` has been internally used by Amazon Advertising for research and production and was reported to serve well for their purposes.

[2]These two libraries are for graphon MFGs and continuous-time variational MFGs, respectively.

[3]The current library is focused on discrete-time MFGs with a finite horizon and finite state and action spaces.

## Library Design

MFGLib consists of two main modules: environments and algorithms, which can be developed and extended entirely independently of one another. Target environments and algorithms can be separately imported and instantiated from their corresponding modules. Users can easily define their own environment by providing problem parameters. Then the algorithms can be deployed to solve the environment instances—find approximate NE solutions. Additionally, the exploitability scores can be computed, which evaluate the proximity of the obtained solutions to an NE solution. The user-facing API of MFGLib is designed such that each aforementioned step can be performed using simple codes.

**Environments.** Users can easily define environments with the syntax:

```python
from mfglib.env import Environment
user_env = Environment(T=T, S=S, A=A, mu0=mu0, r_max=r_max,
                       reward_fn=reward_fn,
                       tranistion_fn=tranistion_fn)
```

Here T,S,A,mu0,r_max are the time horizon, state space size, action space size, initial state distribution and max reward of the MFG, respectively. The rewards reward_fn and transitions transition_fn are the mappings from time (t) and population distribution (L_t) to rewards and transitions tensors, which are callables [4] with the following signatures:

```python
def reward_fn(t: int, L_t: torch.Tensor) -> torch.Tensor:
    ...
def transition_fn(t: int, L_t: torch.Tensor) -> torch.Tensor:
    ...
```

In addition, we also provide several ready-to-use environments [32, 113, 112, 52, 50] that can be directly instantiated as class methods of the Environment class via the environment names like user_env = Environment.left_right(**kwargs), which serve as playgrounds and educative examples to jump-start users before implementing their own environments. The optional environment-dependent keyword arguments can also be specified to adjust the problem parameters, giving more flexibility for the users.

**Solvers and evaluation.** Once a user-defined environment user_env is created, one can then instantiate a solver from the options provided by MFGLib with the following syntax:

```python
from mfglib.alg import MFOMO
# Can also use OnlineMirrorDescent, PriorDescent or FictitiousPlay
user_alg = MFOMO(**kwargs)
solutions, expls, runtimes = user_alg.solve(user_env)
```

Here the outputs store the solutions, exploitability scores and cumulative run-time over iterations. A formatted log of the iteration process is also printed out in real-time to help

[4] We allow both function and class callables. But for simplicity, we focus on function implementations of transitions and rewards here.

users monitor the performance. In addition, for an arbitrary list of policies `policies` (not even necessarily computed by `MFGLib`), the exploitability scores of them can be computed as follows:

```
from mfglib.metrics import exploitability_score
exploitability_scores = exploitability_score(user_env, policies)
```

**Auto-tuning.** Beyond what is shown above, `MFGLib` also provides an auto-tuning tool to automatically select the best algorithm hyperparameters for users if the default performances are not satisfactory. It can be simply invoked with the following syntax, where `env_suite` is a list of environment instances to be tuned on:

```
user_alg_tuned = user_alg.tune(env_suite)
solutions, expls, runtime = user_alg_tuned.solve(user_env)
```

See the next subsection for more details on the additional arguments of the `tune` method.

## Key Features

Below, we highlight the key features of `MFGLib`.

**High-dimensional representation of state and action spaces.** We use `PyTorch` tensors to represent policies, mean-fields, rewards, etc. Whenever possible, we opt to keep the state and action spaces in their original form without performing any transformation. For high dimensional spaces, instead of flattening them and representing them using one dimensional spaces, we keep the original spaces. This treatment yields higher interpretability and provides more flexible and simpler interactions with users.

**Implemented algorithms.** `MFGLib` implements four state of the art MFG algorithms including (Damped) fictitious play [113, 112], online mirror descent [112], prior descent [32], and mean-field occupation measure optimization (MFOMO) [50]. We remark that the implemented algorithms include many other existing algorithms as special cases, such as fixed point iteration and GMF-V algorithms [52].

**Embedded tuner.** Every implemented algorithm requires at least one hyperparameter. To simplify the tuning process, `MFGLib` endows all its algorithms with a built-in tuner, which can be used to tune the hyperparameters on one single environment instance or across several instances (an environment suite). The tuners are based on Optuna [3], an open source hyperparameter optimization framework used to automate hyperparameter search. After the algorithm is initialized, the tuner can be called using a single line of code, as shown in the following code snippet that tunes the learning rate of the online mirror descent algorithm with the `left_right` and `beach_bar` environments.

```python
from mfglib.env import Environment
from mfglib.alg import OnlineMirrorDescent
omd_tuned = OnlineMirrorDescent().tune(
                    env_suite=[Environment.left_right(),
                               Environment.beach_bar()])
```

The `tune` method provides the users with several additional optional arguments to adjust the tuning process, including max iterations (`max_iter`), the absolute and relative tolerances for early stopping (`atol, rol`), the tuning metric (`metric`, which can be either `shifted_geo_mean` or `failure_rate`), the number of trials (`n_trials`) and the maximum run-time of the whole tuning process (`timeout`).

**Code quality and accessibility.** `MFGLib` adheres to the highest standards of code quality. The library is regularly subjected to unit testing and static analysis through a continuous integration (CI) system. We employ the latest tools such as `black` and `ruff` to ensure that the source code remains clean and readable. We also strictly type-check the library with `mypy` to eliminate an entire class of type-safety bugs. Each proposed code change must fully pass the comprehensive CI check before it is permitted to merge.

`MFGLib` welcomes outside contributors and is easily accessible to anyone familiar with Python, beginners and experts alike. The folder structure is simple and easy to navigate. Since `MFGLib` is a pure Python implementation, there is no complicated build process to speak of. We hope the structural simplicity encourages users to engage with the library and submit new algorithms and environments.

## 2.3 Future Work

`MFGLib` is currently undergoing active development, and alongside the addition of new environments and algorithms, there are several potential extensions that can further enhance its scope and user experience.

Regarding the environment and modeling aspect, an exciting direction for `MFGLib`'s expansion would be to support MFGs with infinite horizons, multiple populations, continuous state-action spaces, and graphon structures. Additionally, beyond Nash equilibria, there are several other solution concepts that can be implemented using our framework including but not limited to socially optimal solutions (i.e., mean-field control), equilibrium selection for computing price of anarchy/stability metrics, (coarse) correlated equilibrium, and Stackelberg equilibria.

On the algorithm side, as highlighted in [50][Appendix A], the integration of optimization techniques like Stochastic Projected Gradient Descent and Anderson Acceleration into MFOMO holds potential for enhancing convergence performance in practice. These techniques are not included in the initial release of the package to ensure minimal dependencies and maintain a streamlined package design. Nevertheless, further testing and incorporating these approaches into the package is a key focus for future releases.

# Appendix

In what follows, we first provide a mathematical background on discrete finite-horizon MFGs, and then present the detailed documentation of `MFGLib`. The documentation can also be found online at `https://mfglib.readthedocs.io/en/latest`.

## 2.A Background on Discrete Finite-Horizon MFGs

Consider an MFG with a finite-time horizon $T$, a state space $\mathcal{S}$, and an action space $\mathcal{A}$. We assume $|\mathcal{S}| = S < \infty$ and $|\mathcal{A}| = A < \infty$. In this game, a representative player starts from a state $s_0 \sim \mu_0$, with $\mu_0$ being the initial distribution of all players of an infinite population. At each time step $t \in \mathcal{T} = \{0, 1, 2, \ldots, T\}$ and when at state $s_t$, she chooses an action $a_t \in \mathcal{A}$ from some policy $\pi_t : \mathcal{S} \to \Delta(\mathcal{A})$, where $\Delta(\mathcal{A})$ denotes the set of probability vectors on $\mathcal{A}$. she will then move to a new state $s_{t+1}$ according to a transition probability $P_t(.|s_t, a_t, L_t)$ and receive a reward $r_t(s_t, a_t, L_t)$, where $L_t \in \Delta(\mathcal{S} \times \mathcal{A})$ is the joint state-action distribution among all players at time $t$ referred to as the mean-field information hereafter.

Given the mean-field flow $\{L_t\}_{t\in\mathcal{T}}$, the objective of this representative player is to maximize her accumulated rewards, *i.e.,* to solve the following Markov Decision Process (MDP) problem:

$$\begin{aligned} \text{maximize}_{\{\pi_t\}_{t\in\mathcal{T}}} \quad & \mathbb{E}\left[\sum_{t=0}^{T} r_t(s_t, a_t, L_t) \middle| s_0 \sim \mu_0\right] \\ \text{subject to} \quad & s_{t+1} \sim P_t(\cdot|s_t, a_t, L_t), \quad t = 0, \ldots, T-1, \\ & a_t \sim \pi_t(\cdot|s_t), \quad t = 0, \ldots, T. \end{aligned} \tag{2.1}$$

For a given mean-field flow $L = \{L_t\}_{t\in\mathcal{T}}$, Let's denote the mean-field induced MDP (2.1) as $\mathcal{M}(L)$, $V_t^\star(L) \in \mathbb{R}^S$ as its optimal total expected reward, *i.e.,* the value function, starting from time $t$, with the $s$-th entry $[V_t^\star(L)]_s$ being the optimal expected reward starting from state $s$ at time $t$, and $V_{\mu_0}^\star(L) = \sum_{s\in\mathcal{S}} \mu_0(s)[V_0^\star(L)]_s$ as its optimal expected total reward starting from $\mu_0$. Correspondingly, denote respectively $V_t^\pi(L) \in \mathbb{R}^S$, $[V_t^\pi(L)]_s$, and $V_{\mu_0}^\pi(L) = \sum_{s\in\mathcal{S}} \mu_0(s)[V_0^\pi(L)]_s$, under a given policy $\pi = \{\pi_t\}_{t\in\mathcal{T}}$ for $\mathcal{M}(L)$.

To analyze such an MFG, the most widely adopted solution concept is the Nash equilibrium (NE). A policy sequence $\{\pi_t\}_{t\in\mathcal{T}}$ and a mean-field flow $\{L_t\}_{t\in\mathcal{T}}$ constitute an NE solution of this finite-time horizon MFG, if the following conditions are satisfied.

**Definition 1** (NE solution)**.**

1) *(Optimality/Best Response) Fixing $\{L_t\}_{t\in\mathcal{T}}$, $\{\pi_t\}_{t\in\mathcal{T}}$ solves the optimization problem (2.1), i.e., $\{\pi_t\}_{t\in\mathcal{T}}$ is optimal for the representative agent given the mean-field flow $\{L_t\}_{t\in\mathcal{T}}$;*

2) *(Consistency) Fixing $\{\pi_t\}_{t\in\mathcal{T}}$, the consistency of the mean-field flow holds, namely*

$$L_t = \mathbb{P}_{s_t,a_t}, \text{ where } s_{t+1} \sim P_t(\cdot|s_t,a_t,L_t),\ a_t \sim \pi_t(\cdot|s_t),\ s_0 \sim \mu_0,\ t = 0,\dots,T-1. \tag{2.2}$$

*Here $\mathbb{P}_x$ denotes the probability distribution of a random variable/vector $x$.*

Note that (2.1) requires that the policy sequence $\{\pi_t\}_{t\in\mathcal{T}}$ is the best response to the flow $\{L_t\}_{t\in\mathcal{T}}$, while (2.2) requires that the flow $\{L_t\}_{t\in\mathcal{T}}$ is the corresponding mean-field flow induced when all players adopt the policy sequence $\{\pi_t\}_{t\in\mathcal{T}}$. Also note that (2.2) can be written more explicitly as follows:

$$\begin{aligned} L_0(s,a) &= \mu_0(s)\pi_0(a|s), \\ L_{t+1}(s',a') &= \pi_{t+1}(a'|s')\sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}} L_t(s,a)P_t(s'|s,a,L_t), \quad \forall t = 0,\dots,T-1. \end{aligned} \tag{2.3}$$

The following existence result for NE solutions holds as long as the transitions and rewards are continuous in $L_t$. The proof is based on the Kakutani fixed-point theorem; it is almost identical to those in [122, 32], except for replacing the state mean-field flow with the state-action joint mean-field flow.

**Proposition 1.** *Suppose that $P_t(s'|s,a,L_t)$ and $r_t(s,a,L_t)$ are both continuous in $L_t$ for any $s,s' \in \mathcal{S}$, $a \in \mathcal{A}$ and $t \in \mathcal{T}$. Then an NE solution exists.*

In order to characterize the difference between any policy and an NE solution, the common approach is to use the concept of exploitability. More precisely, define a mapping $\Gamma$ that maps any policy sequence $\{\pi_t\}_{t\in\mathcal{T}}$ to its induced mean-field flow $\{L_t\}_{t\in\mathcal{T}}$ when all players take such a policy sequence. Following the consistency condition (2.3), such $\Gamma$ can be defined recursively, starting with the initialization

$$\Gamma(\pi)_0(s,a) := \mu_0(s)\pi_0(a|s), \tag{2.4}$$

such that

$$\Gamma(\pi)_{t+1}(s,a) := \pi_{t+1}(a|s)\sum_{s'\in\mathcal{S}}\sum_{a'\in\mathcal{A}} \Gamma(\pi)_t(s',a')P_t(s|s',a',\Gamma(\pi)_t), \quad \forall t = 0,\dots,T-1. \tag{2.5}$$

Then, the exploitability characterizes the sub-optimality of the policy $\pi$ under $L = \{L_t\}_{t\in\mathcal{T}} = \Gamma(\pi)$ as follows,

$$\mathrm{Expl}(\pi) := V^{\star}_{\mu_0}(\Gamma(\pi)) - V^{\pi}_{\mu_0}(\Gamma(\pi)) = \max_{\pi'} V^{\pi'}_{\mu_0}(\Gamma(\pi)) - V^{\pi}_{\mu_0}(\Gamma(\pi)). \tag{2.6}$$

In particular, $(\pi, L)$ is an NE solution if and only if $L = \Gamma(\pi)$ and $\mathrm{Expl}(\pi) = 0$. And a policy $\pi$ is an $\epsilon$-NE solution if $\mathrm{Expl}(\pi) \le \epsilon$.

## 2.B MFGLib Documentation

MFGLib provides pre-built wheels for Python 3.8+ and can be installed via `pip` on all major platforms:

```
$ pip install mfglib
```

### Your First MFG

To demonstrate the power of MFGLib, let's use the library to find a Nash equilibrium (NE) solution for an instance of the Beach Bar environment. We begin by importing the `Environment` class.

```
from mfglib.env import Environment
```

The `Environment` class comes equipped with several classmethods which can be used to create instances of environments well-studied in the mean-field game literature. In addition to the pre-implemented environments, you can also easily implement your own custom environments. More on pre-implemented and custom environments can be found in the Environments subsection.

With `Environment` imported, we can then instantiate a Beach Bar instance by calling the corresponding classmethod.

```
beach_bar = Environment.beach_bar()
```

To "solve" the instance, we must next introduce an algorithm. Solving an environment means finding an approximate NE solution for it.

In this example, let's use Online Mirror Descent. Other options include Fictitious Play, Prior Descent, and Mean-Field Occupation Measure Optimization. Just like with the environments, MFGLib also supports user-defined algorithms. More on the algorithms can be found in the Algorithms section.

```
from mfglib.alg import OnlineMirrorDescent
online_mirror_descent = OnlineMirrorDescent()
```

Now, we just need to call `solve()`. The `solve()` method returns a three-item tuple: a list of policies (solutions) found during iteration, exploitability scores of the solutions, and the runtime at each iteration.

```
solns, expls, runtimes = online_mirror_descent.solve(beach_bar)
```

The `solve()` method allows us to set the initial policy, change the number of iterations, use early stopping, and print the convergence information during iteration. More details can be found in API Documentation.

By default, `solve()` runs for 100 iterations and assumes the initial policy to be the uniform policy over the state and action space at each time step. We can verify this by comparing `solns[0]` with `solns[-1]`.

```
solns[0]
>>> tensor([[[0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333]],
         [[0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333]],
         [[0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333],
          [0.3333, 0.3333, 0.3333]]])
```

```
solns[-1]
>>> tensor([[[1.6836e-04, 9.9983e-01, 3.7345e-32],
          [1.0000e+00, 1.9363e-10, 6.1326e-33],
          [8.3867e-01, 1.3771e-01, 2.3616e-02],
          [1.1221e-21, 9.9755e-01, 2.4476e-03]],
         [[1.0706e-08, 1.0000e+00, 8.2818e-23],
          [6.9968e-01, 3.0032e-01, 7.3669e-19],
          [7.8170e-05, 9.9416e-01, 5.7577e-03],
          [3.3300e-20, 9.9985e-01, 1.5068e-04]],
         [[1.3887e-11, 1.0000e+00, 1.3887e-11],
          [1.3888e-11, 1.0000e+00, 1.3888e-11],
          [1.3888e-11, 1.0000e+00, 1.3888e-11],
          [1.3888e-11, 1.0000e+00, 1.3888e-11]]])
```

To compare the two solutions, we look at their exploitability scores.

```
expls[0]
>>> 0.9316978454589844
```

```
exps[-1]
>>> 0.0024423599243164062
```

The computed exploitability score is decreased significantly implying that the last policy is a fairly good approximation of an NE solution for the Beach Bar environment. You can monitor the progression of an algorithm by plotting the exploitability scores vs. the number of iterations or vs. the runtime as shown in Figure 2.B.1.

## Variable Representation

We use `PyTorch` tensors to represent policies, mean-fields, etc. Whenever possible, we opt to keep the state and action spaces in their original form without performing any transformation. For high dimensional spaces, instead of flattening them and representing them using one dimensional spaces, we keep the original spaces. This treatment yields higher interpretability and provides more flexible and simpler interactions with users.

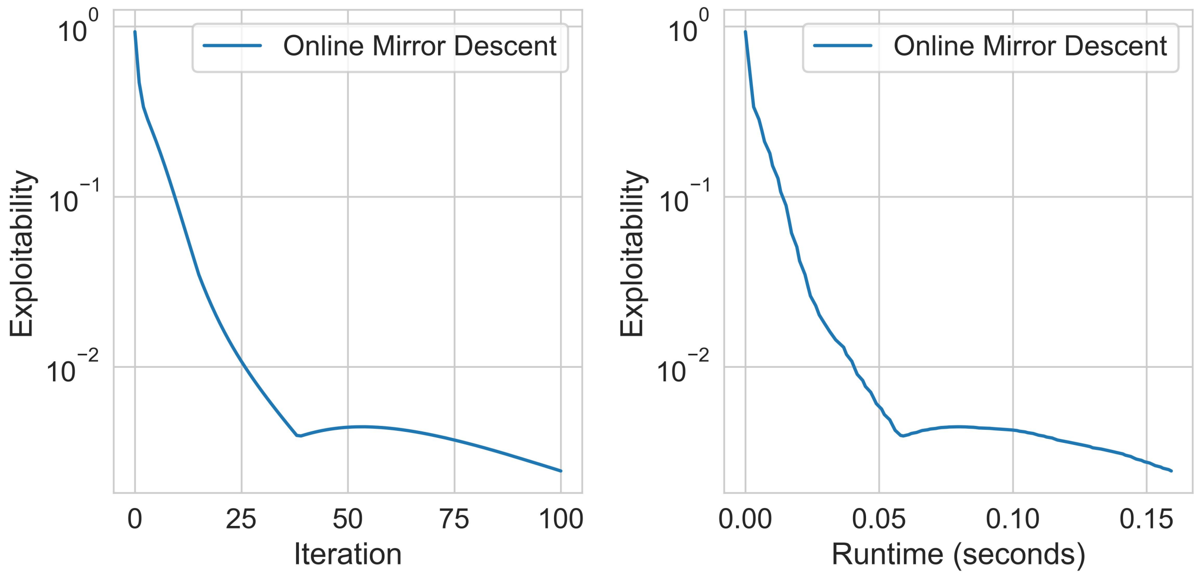


Figure 2.B.1: Exploitability scores obtained by running the Online mirror Descent algorithm on the Beach Bar environment

When creating an environment, the attributes `T` (integer), `S` (tuple of integers), and `A` (tuple of integers) determine the time horizon, state space shape, and action space shape, respectively. For example, if the state space is all the integers from 1 to 100, then `S=(100,)`, and if the state space is all the integer grid points $(x, y)$ such that $1 \leq x, y \leq 100$, then `S=(100, 100)`.

Policies, mean-fields, rewards, and transition probabilities are represented as high-dimensional tensors. Given `T`, `S`, and `A`, the shape of policy and mean-field tensors will be `(T+1,) + S + A`. For example, if `T=10, S=(20, 20), A=(5,)`, the policy and mean-field tensors will be of size `(11, 20, 20, 5)`. The reward and transition probability tensors at a given time `t` will be of shape `S + A` and `S + S + A`, respectively. We do not integrate the time into the reward and transition probability tensors.

In general, let `S=(S_1, S_2, ..., S_n)` and `A=(A_1, A_2, ..., A_m)`, and let `pi`, `L`, `r_t`, and `p_t` be the policy, mean-field, reward (at `t`), and transition probability (at `t`) tensors, respectively. Then, `pi[t, s_1, s_2, ..., s_n, a_1, a_2, ..., a_m]` represents the probability of choosing action `a = (a_1, a_2, ..., a_m)` given the state `s = (s_1, s_2, ..., s_n)` at time `t`. `L[t, s_1, s_2, ..., s_n, a_1, a_2, ..., a_m]` is the portion of players that are in state `s = (s_1, s_2, ..., s_n)` and choose action `a = (a_1, a_2, ..., a_m)` at time `t`. `r_t[s_1, s_2, ..., s_n, a_1, a_2, ..., a_m]` is the reward that agent gets from choosing action `a = (a_1, a_2, ..., a_m)` while being at state `s = (s_1, s_2, ..., s_n)` at time `t`. And lastly, `p_t[s2_1, s2_2, ..., s2_n, s1_1, s1_2, ...,` `s1_n, a_1, a_2, ..., a_m]` is the probability of going to the state `s2 = (s2_1, s2_2, ..., s2_n)` if the agent is currently at the state `s = (s_1, s_2, ..., s_n)` and chooses the action `a = (a_1, a_2, ..., a_m)` at time `t`.

## Algorithms

This library comes with 4 implemented MFG algorithms:

- Fictitious Play
- Online Mirror Descent [112].
- Prior Descent [32].
- Mean-Field Occupation Measure Optimization [50].

The first three algorithms are built on top of three main modules: `mean_field.py`, `q_fn.py`, and `greedy_policy_given_mean_field.py`. The last algorithm has a relatively different structure. In the following, we first provide more information about the fundamental modules for iterative methods. Then, we discuss the `solve()` method embedded in all algorithms. Next, we elaborate on each one of the implemented algorithms. Last, we describe how to use the auto-tuning tool provided in `MFGLib`.

### Fundamental Modules

The fundamental modules facilitate the implementation of iterative methods.

Let's start with `mean_field.py`, which implements the function `mean_field()` with the following signature:

```python
def mean_field(env: Environment, pi: torch.Tensor) -> torch.Tensor:
    ...
```

`mean_field()` computes the induced mean-field corresponding to a policy $\pi$. The induced mean-field $\Gamma(\pi)$ satisfies the equations (2.4) and (2.5), and therefore can be computed recursively.

The second module, `q_fn.py`, implements the `QFn` class. Two methods are implemented within `QFn`: `optimal()` computes the optimal Q-function given a mean-field $L$, and `for_policy()` computes the Q-function corresponding to a pair of mean-field $L$ and policy $\pi$. The signature of `QFn` is as follows:

```python
class QFn:
    def __init__(
        self, env: Environment, L: torch.Tensor, *, verify_integrity: bool
    = True
    ) -> None:
        ...
    def optimal(self) -> torch.Tensor:
        ...
    def for_policy(self, pi: torch.Tensor) -> torch.Tensor:
        ...
```

The optimal Q-function of mean-field $L$ can be computed using a backward recursive equation as in (2.7) and (2.8).

$$Q^\star(L)_T(s,a) := r_T(s,a,L_T), \tag{2.7}$$

$$Q^\star(L)_t(s,a) := r_t(s,a,L_t) + \sum_{s'} P_t(s'|s,a,L_t) \max_{a'} Q^\star(L)_{t+1}(s',a'). \tag{2.8}$$

Similarly, the Q-function corresponding to a mean-field $L$ and policy $\pi$ satisfies backward recursive equations as in (2.9) and (2.10).

$$Q(L,\pi)_T(s,a) := r_T(s,a,L_T), \tag{2.9}$$

$$Q(L,\pi)_t(s,a) := r_t(s,a,L_t) + \sum_{s',a'} P_t(s'|s,a,L_t)\pi_{t+1}(a'|s')Q(L,\pi)_{t+1}(s',a'). \tag{2.10}$$

Lastly, `greedy_policy_given_mean_field.py` implements `Greedy_Policy()` whose signature is provided below:

```
def Greedy_Policy(env_instance: Environment, L: torch.Tensor) -> torch.
   Tensor:
    ...
```

`Greedy_policy()` computes the policy $\pi^\star(L)$ which is the best response to the given mean-field $L$. Given the optimal Q-function, the best response can be computed as follows:

$$\pi^\star(L)_t(s,a) = \begin{cases} 1, & \text{if } a = \text{argmax}_{a'} Q^\star(L)_t(s,a'), \\ 0, & \text{otherwise.} \end{cases}$$

Note that if the argmax is not unique, then the result will be a uniform distribution between the actions that attain the maximum value.

Efficient implementation is a crucial consideration for the fundamental modules. Given that variables like policies, mean-fields, rewards, etc. often involve high-dimensional tensors, it is essential to minimize the usage of loops when solving the forward and backward recursive equations. In the implementation of the fundamental modules, we make optimal use of tensor vectorization to reduce the reliance on `for` loops. The code employs loops only to iterate over time. By utilizing tensor vectorization and minimizing loop usage, we can achieve faster and more efficient computations within the fundamental modules.

### The `solve()` Method

Each algorithm implemented in this library is equipped with a `solve()` method upon calling which the algorithm will be run on the given environment instance. The signature of the `solve()` methods is shown below.

```python
def solve(
        self,
        env_instance: Environment,
        *,
        pi: Literal["uniform"] | torch.Tensor = "uniform",
        max_iter: int = 100,
        atol: float | None = 1e-3,
        rtol: float | None = 1e-3,
        verbose: bool = False,
    ) -> tuple[list[torch.Tensor], list[float], list[float]]:
        ...
```

Let's take a closer look at this method's inpuit arguments.

- The only required argument is `env_instance`. In the example provided in Your First MFG, we set it to `beach_bar`.
- `pi` defines the initial policy that the algorithm starts with. By default, it is set to `"uniform"` meaning that the uniform distribution will be considered. You may want to initialize the algorithm at a non-uniform policy in which case you need to define the initial tensor and pass it to `pi`. Just make sure that the policy tensor defined is indeed a policy, and also that it matches the time horizon and state and action shapes of your environment.
- `max_iter` determines the maximum number of iterations to run the algorithm. The reason for setting the maximum number is that due to the early stopping feature, the algorithm could terminate earlier.
- `atol` and `rtol` define the absolute and relative tolerance for the exploitability score. If at least one of them is not `None`, at every iteration, we compute the exploitability score and compare it with the threshold `atol + rtol * score_0`, where `score_0` is the initial exploitability. If it was less than or equal to the threshold, we stop the algorithm.
- `verbose` determine whether the convergence information during iteration is printed.

The `solve()` method outputs the policy iterations and their corresponding exploitability scores, as well as the runtimes.

### Fictitious Play

Discrete time Fictitious Play algorithm [113] provides a robust approximation scheme for computing NEs by computing iteratively the best response against the distribution induced by the average of the past best responses. See [113][Algorithm 1] for the details. Our implementation is based on Fictitious Play Damped introduced in [112]. The damped version generalizes the original algorithm by adding a learning rate parameter $\alpha$. It is worth mentioning that the Fixed Point Iteration algorithm is a special case when $\alpha = 1$.

**Hyperparameters.** The only hyperparameter is `alpha` that corresponds to the learning rate used in Fictitious Play Damped. It takes any value in the close interval $[0, 1]$ or can be `None`. If `None`, the implementation would be inline with the original Fictitious Play algorithm as the learning rate used in the $n^{th}$ iteration would be $\frac{1}{n+1}$. The default is `None`. The example code below creates an instance of the Fictitious Play algorithm with learning rate 0.1.

```python
from mfglib.alg import FictitiousPlay
fp = FictitiousPlay(alpha=0.1)
```

### Online Mirror Descent

[112] introduce the Online mirror Descent algorithm to address scaling up equilibrium computation in MFGs. See [112][Algorithm 1] for the details. They empirically show that Online Mirror Descent scales up and converges significantly faster than Fictitious Play.

**Hyperparameters.** The only hyperparameter is `alpha` which represents the learning rate. It can take any non-negative real value. The default is set to 1.0. The example code below creates and instance of the Online mirror Descent algorithm with learning rate 0.01.

```python
from mfglib.alg import OnlineMirrorDescent
omd = OnlineMirrorDescent(alpha=0.01)
```

### Prior Descent

The Prior Descent algorithm, proposed by Cui et al. (2021), employs two nested loops, namely the inner and outer loops, to iteratively update the policy. Within the inner loop, Boltzman/RelEnt Iteration is executed for a predetermined number of iterations to update the prior policy. This process is repeated for the specified number of outer loops. See [32][Algorithms 2 and 6] for more details. While the Prior Descent algorithm improves the performance of algorithms employed in its inner loop, the presence of a nested double loop structure can be computationally demanding, particularly in the context of deep reinforcement learning.

**Hyperparameters.** The first hyperparameter is `eta`, the temperature. It can take any non-negative real value. The default is 1.0. The second one is `n_inner`, which determines the number of iterations between prior policy updates. It can be a positive integer or `None`. If `None`, prior policy remains intact, which is basically the GMF-V algorithm [52]. An instance of Prior Descent can be created as follows:

```python
from mfglib.alg import PriorDescent
pd = PriorDescent(eta=0.1, n_inner=10)
```

### Mean-Field Occupation Measure Optimization (MFOMO)

The MFOMO algorithm [50] reformulates the problem of finding NE solutions in MFGs as solving an equivalent optimization problem with bounded variables and trivial convex constraints. It is built on the classical work of reformulating an MDP as a linear program, and by adding the consistency constraint for MFGs in terms of occupation measures, and by exploiting the complementarity structure of the linear program. See [50][Theorem 5] for more details. It is worth mentioning that the initial motivation behind creating this library was to produce the experimental results for MFOMO.

The constrained optimization problem that MFOMO solves to find an NE solution is presented below:

$$\begin{aligned} \min_{L,z,y} \quad & \|A_L L - b\|_2^2 + \|A_L^T y + z - c_L\|_2^2 + z^T L \\ \text{s.t.} \quad & L \geq 0,\ \ z \geq 0, \\ & 1^T L_t = 1\ \ \forall t \in \{0, \dots, T\}, \\ & 1^T z \leq SA(T^2 + T + 2) r_{\max}, \\ & \|y\|_2 \leq \frac{S(T+1)(T+2)}{2} r_{\max}, \end{aligned} \tag{2.11}$$

where $L = \{L_t\}_{t\in\mathcal{T}}$ is a flattened vector in $\mathbb{R}^{SA(T+1)}$, $z \in \mathbb{R}^{SA(T+1)}$, $y \in \mathbb{R}^{S(T+1)}$, and $r_{\max} = \sup_{t\in\mathcal{T}, s\in\mathcal{S}, a\in\mathcal{A}, L\in\Delta(\mathcal{S}\times\mathcal{A})} |r_t(s,a,L)|$. See [50][Equations (7) and (8)] for the description of $A_L$, $b$, and $c_L$. Note that the flattening of vectors in `MFGLib` are with row-major order, in contrast to the column-major order used in the corresponding article.

If $(\pi, L)$ is an NE solution of the MFG ((2.1) and (2.2)), then there exist some $y, z$ such that $(y, z, L)$ solves (2.11). On the other hand, if $(y, z, L)$ solves (2.11) with objective value 0, then for any $\pi \in \Pi(L)$, $(\pi, L)$ is an NE solution of the MFG ((2.1) and (2.2)). Note that $\pi \in \Pi(L)$ if and only if

$$\pi_t(a|s) = \frac{L_t(s,a)}{\sum_{a'\in\mathcal{A}} L_t(s,a')},$$

when $\sum_{a'\in\mathcal{A}} L_t(s,a') > 0$, and $\pi_t(|s)$ is an arbitrary probability vector in $\Delta(\mathcal{A})$ otherwise.

Since MFOMO is structurally different than the iterative methods, we have created different fundamental modules to implement it:

- `mf_omo_params.py` implements the function `mf_omo_params()` with the following signature:

```
    def mf_omo_params(
    env_instance: Environment, L: torch.Tensor
    ) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
        ...
```

  given an environment instance and its mean-field, this function computes $A_L$, $b$, $c_L$.

- `mf_omo_obj.py` implements the function `mf_omo_obj()`, which basically computes the objective function in (2.11). Its signature is provided below:

```
    def mf_omo_obj(
    env_instance: Environment,
    L_u: torch.Tensor,
    z_v: torch.Tensor,
    y_w: torch.Tensor,
    loss: Literal["l1"] | Literal["l2"] | Literal["l1_l2"],
    c1: float,
    c2: float,
    c3: float,
    parameterize: bool,
    ) -> torch.Tensor:
        ...
```

  Its input arguments are later explained in details while we breakdown the hyperparameters of MFOMO. Note that by differentiating the output of this function, we get the gradients corresponding to variables $L$, $z$, and $y$, which can be further used to iteratively solve the optimization problem.

- `mf_omo_constraints.py` implements the function `mf_omo_constraints()`, whose signature is presented below:

```
    def mf_omo_constraints(
    env_instance: Environment,
    L: torch.Tensor,
    z: torch.Tensor,
    y: torch.Tensor,
    ) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
        ...
```

  This function returns the projections of $L$, $z$, and $y$ onto the constraint set of (2.11).

By utilizing the aforementioned modules and leveraging automatic differentiation capabilities in PyTorch, we can iteratively solve (2.11) using diverse optimization algorithms like Projected Gradient Descent. Furthermore, Guo et al. (2022) propose several techniques that can enhance convergence in this context. Below, as we describe the hyperparamters, we also review the optimization techniques employed in the current implementation of MFOMO.

**Hyperparameters.** The MFOMO implementation has more than 10 hyperparameters detailed below:

- **Optimization Algorithm:** You can run MFOMO with different optimization algorithms. Currently, we only support PyTorch optimizers. Using the `optimizer` hyperparameter, you can determine your desired optimizer. `optimizer` should be a dictionary with two keys:

- `"name"`: The name of a PyTorch optimizer, e.g., `Adam`, `SGD`, `RMSprop`, etc.
- `"config"`: The desired configuration for the selected optimizer. For example, if we choose `Adam`, then we can set the value of `"config"` as `{"lr": 0.1, "amsgrad":True}`.

By default, `optimizer` is set to `{"name": "Adam", "config": "lr": 0.1}`. Note that since we are solving a constrained optimization problem, the optimizer iterations are automatically projected onto the constraint set after each iteration.

- **Parameterized Formulation:** Replacing the constrained optimization problem (2.11) with a smooth unconstrained problem enables us to use a broader range of optimization solvers. As explained in [50][Appendix A.3], we can reparameterize the variables in MFOMO to completely get rid of the constraints. The new problem is called the "parameterized" formulation. Using the following input argument, we can switch between the different formulations:
  - `parameterize`: Optionally solve the alternate "parameterized" formulation. Default is `False`.
- **Hat Initialization:** According to [50][Proposition 6], if $L, z, y$ is a solution to (2.11), so is $L, \hat{z}, \hat{y}$. See the corresponding proposition for detailed definitions of $\hat{z}$ and $\hat{y}$. We have experimentally found that utilizing hat initialization, which means replacing the initial $L_0, z_0, y_0$ with $L_0, \hat{z}_0, \hat{y}_0$, could boost the performance of optimization algorithms. The hyperparameter `hat_init` determines whether this initialization scheme is used. Default is `False`.
- **Redesigned Objective:** In the optimization problem (2.11), one can assign different coefficients to the three terms in the objective function, and come up with a "redesigned objective". To be precise, the redesigned objective is

$$c_1\|A_L L - b\|_2^2 + c_2\|A_L^T y + z - c_L\|_2^2 + c_3 z^T y.$$

  Furthermore, one can apply different norms (L1 or L2) to the different terms in the objective function. Using specific caefficients and norms could potentially improve the theoretical guarantees of MFOMO. The following input arguments determine the parameters of the redesigned objective:
  - `c1, c2, c3` determine the redesigned objective coefficients. Default is 1 for all the coefficients. Also, note that without loss of generality, we can always let `c3=1`.
  - `loss` determines the type of norm (L1, L2, or both) used in the redesigned objective function. Three available options are listed below:
    1. `"l1"`: The objective will be $c_1\|A_L L - b\|_1 + c_2\|A_L^T y + z - c_L\|_1 + c_3 z^T y$.
    2. `"l2"`: The objective will be $c_1\|A_L L - b\|_2^2 + c_2\|A_L^T y + z - c_L\|_2^2 + c_3 (z^T y)^2$.

3. `"l1_l2"`: The objective will be $c_1\|A_L L - b\|_2^2 + c_2\|A_L^T y + z - c_L\|_2^2 + c_3 z^T y$.

The default value is `"l1_l2"`.

- **Adaptive Residual Balancing:** We can adaptively change the coefficients `c1, c2, c3` of the redesigned objective based on the value of their corresponding objective term. This can be done using the following input arguments:
  - `m1, m2, m3` determine the parameters used for adaptive residual balancing. Let's denote by $O_1$ the value of the first objective term (depending on the norm used, it could be either $\|A_L L - b\|_2^2$ or $\|A_L L - b\|_1$), and let $O_2$ and $O_3$ be the values of the second and third objective terms, respectively. When adaptive residual balancing is applied, we modify the coefficients in the following cases:
    1. If $O_1/\max(O_2, O_3)$ is greater than `m1`, then multiply `c1` by `m2`.
    2. If $O_1/\min(O_2, O_3)$ is less than `m3`, then divide `c1` by `m2`.
    3. If $O_2/\max(O_1, O_3)$ is greater than `m1`, then multiply `c2` by `m2`.
    4. If $O_2/\max(O_1, O_3)$ is less than `m3`, then divide `c2` by `m2`.
  - `rb_freq` determines the frequency of applying residual balancing. It can be a positive integer or `None`. If `None`, residual balancing will not be applied.
- **Initialization:** We can set the initial policy for any algorithm using the input argument `pi` through the `solve()` method. MFOMO uses the initial policy to compute the initial values of the variables $L, z, y$. However, if you want to initialize these variables directly, you can do so using the following input arguments:
  - `L, z, y`: The initial values of math $L, z, y$. Default is `None`. If not `None`, these values overwrite the initial values derived from the initial policy.
  - `u, v, w`: The initial values of the variables $u, n, w$ used in the "parameterized" formulation. Refer to the [50][Appendix A.3] for more information. Default is `None`. If not `None`, these values overwrite the initial values derived from the initial policy.

Apart from the techniques and parameters mentioned earlier, we also conducted experiments with additional approaches such as Anderson Acceleration, Stochastic Projected Gradient Descent, and Hat Enforcement to improve the convergence results of MFOMO. Anderson Acceleration (AA) [5] can help accelerate the convergence of optimization algorithms. We used the `aa` package developed by the Stanford University Convex Group, which implements AA based on the scheme described in [150]. Its potential benefits are briefly discussed in [50][Appendix A.3]. Stochastic Projected Gradient Descent takes advantage of the property that the objective function in (2.11) can be expressed as a summation of $S(2A+1)(T+1)$ terms. Consequently, it becomes possible to compute gradients by considering only a subset of these terms, known as a mini-batch. This approach is particularly advantageous when dealing with large-scale games, as it allows for more efficient and manageable computations by operating on a reduced subset of terms. See [50][Appendix A.1] for more details. The Hat Enforcement technique shares similarities with the previously mentioned Hat Initialization. However, in Hat Enforcement, the variables $L, z, y$ are transformed to $L, \hat{z}, \hat{y}$ at regular intervals (e.g., every 10 iterations), as opposed to Hat Initialization, where the variable transformation takes place solely during initialization. However, these techniques were not included in the initial release of the package to ensure minimal dependencies and maintain a streamlined package design. Nevertheless, further testing and incorporating these approaches into the package is a key focus for future releases.

### Hyperparameter Tuning

Choosing the right set hyperparameters is essential to get the best performance out of an algorithm. A set of hyperparameters could work for one environment but result in a poor performance in other environments. Even two distinct instances of the same environment could require very different sets of hyperparameters. Accordingly, manually tuning the hyperparameters for algorithms such as Fictitious Play and Online Mirror Descent, despite having only one tunable parameter, is not very straight forward, let alone for Prior Descent and specifically for MFOMO that have several hyperparameters with a wide value range.

All the algorithms in MFGLib are endowed with a built-in tuner which could be used to tune the algorithms on one single environment instance or a suite of several environment instances. The tuners are based on Optuna [3], an open source hyperparameter optimization framework used to automate hyperparameter search. You just need to call the `tune()` method to start the tuning process. Let's take a closer look at the `tune()` method and its input arguments.

- `env_suite` is the list of environments we want to tune our algorithm on.
- `max_iter` determines for how many iterations each algorithm trial should be run on each environment instance in the environment suite.
- `atol` and `rtol`: Determine the early stopping parameters.

  While running an algorithm on an environment instance, if the exploitability level reaches or goes below `atol + rtol * score_0`, where `score_0` is the initial exploitability, we consider the environment instance **solved** by the algorithm. Otherwise, we mark it as **unsolved**.

  When an algorithm is run on an environment instance, **Stopping Iteration** is the number of iterations needed for the algorithm to reach the desired exploitability level (`atol + rtol * score_0`). If the algorithm does not reach this level in less than `max_iter` iterations, we set the stopping iteration to `max_iter`.
- `metric` determines the metric used by the tuner—the tuner searches for a set of hyperparameters that minimizes the given metric. The two supported options are listed below:

  - `"shifted_geo_mean"`: Assume that we have $n$ environment instances in the environment suite and an algorithm is run on all these instances. Let's denote by $(s_1, s_2, \dots, s_n)$ the stopping iterations corresponding to these environment instances. Then, a way to evaluate the performance of the algorithm on the whole environment suite is to consider the shifted geometric mean of the stopping iterations. To be precise, we consider $\sqrt[n]{(s_1+m)(s_2+m)\dots(s_n+m)} - m$, where $m$ is the shift parameter. Note that $s_i$ are non-negative numbers less or equal to `max_iter`. Therefore, the shifted geometric mean would be a non-negative number less than or equal to `max_iter`.
  - `"failure_rate"`: This metric determines the portion of instances in the environment suite NOT solved by the algorithm, which will be a number in the interval $[0, 1]$.

- `n_trials` is the number of trials. If this argument is not given, as many trials are run as possible.
- `timeout` is used to stop tuning after the given number of second(s).

To demonstrate how the tuner works, let's consider an instance of the Rock Paper Scissors environment and tune the Online Mirror Descent algorithm on it. We will compare the performance of the tuned and default algorithms. The tuner runs for 20 trials and the time limit is 60 seconds. Below, we present the corresponding code snippet. Note that `env_suite=[Environment.rock_paper_scissors()]` as we want to tune the algorithm only on one specific environment instance.

```python
from mfglib.env import Environment
from mfglib.alg import OnlineMirrorDescent

# Default algorithm
omd = OnlineMirrorDescent()

# Tuned algorithm
omd_tuned = omd.tune(
    env_suite=[Environment.rock_paper_scissors()],
    max_iter=500,
    atol=0,
    rtol=1e-2,
    metric="shifted_geo_mean",
    n_trials=20,
    timeout=60,
)
```

Figure 2.B.2 compares the performance of the default and tuned algorithms. The tuned algorithm outperforms the default. By setting lower exploitability thresholds, we might get even a better performance, but we may need to run the tuner for more trials and time. A few remarks about the tuner:

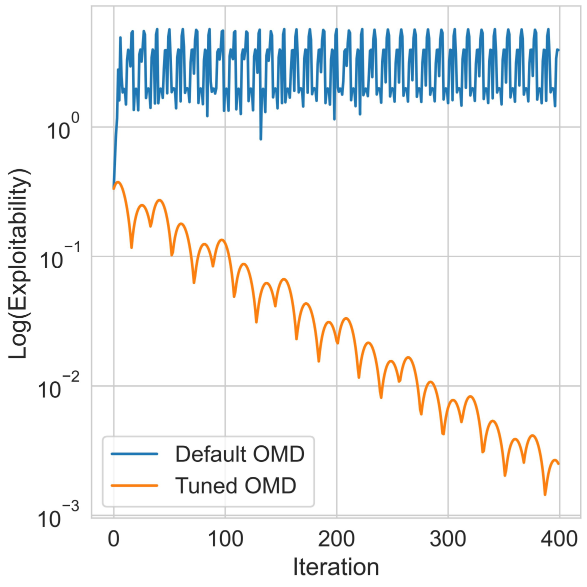


Figure 2.B.2: Comparing the exploitability scores obtained by the default and tuned Online Mirror Descent.

- To ensure that the tuned hyperparameters work well on a broader range of environments, we can pass a list of multiple environment instances to the tuner via the `env_suite` input argument.
- The default set of hyperparameters may not be used during the tuning process. Consequently, there might be cases in which the default algorithm outperforms the tuned algorithm.
- Depending on the the values of the tuner's inputs such as `max_iter`, `atol`, `rtol`, etc., it is possible that none of the algorithm trials solve any of the environment instances in which case the tuner does nothing. However, if at least one of the algorithm trials is successful in solving at least one of the instances, then tuner outputs the algorithm equipped with the best set of hyperparameters.
- An algorithms' tuner conducts hyperparameter search for all the existing hyperparameters and over predetermined search domains, which is determined via the `_tuner_instance()` method. By modifying this method and adapt the `tune()` method to these changes, one can change the set of tunable hyperparameters as well as their corresponding search domains.

## Environments

In this section, we first discuss how users can define their own custom environments, and subsequently we review the preimplemented environments offered in this library.

### User-Defined Environments

The `Environment` class is the general class for MFG environments. It can be imported as follows:

```python
from mfglib.env import Environment
```

Below, we describe the different attributes of the `Environment` class:

- `T`: Sets the time horizon of the environment from 0 to `T`.
- `S`: State space shape.
- `A`: Action space shape.
- `mu0`: Initial state distribution.
- `r_max`: The supremum of the absolute value of rewards. This parameter is only used in Mean-Field Occupation Measure Optimization algorithm and does not necessarily need to be exact. Even a loose upper bound would be sufficient.
- `reward_fn`: Defines the reward function. The user is allowed to pass either a function or a class implementing `__call__`. The inputs of the reward function must be `env` (an environment instance), `t` (a specific time in the time horizon), and `L_t` (the mean-field tensor at time `t`). The output will be a tensor of shape `S + A`. The signature of reward function is presented below:

  ```python
      def reward_fn(t: int, L_t: torch.Tensor) -> torch.Tensor:
          ...
  ```

- `transition_fn`: Defines the transition probability function. The user is allowed to pass either a function or a class implementing `__call__`. The inputs of the transition probability function must be `env` (an environment instance), `t` (a specific time in the time horizon), and `L_t` (the mean-field tensor at time `t`). The output will be a tensor of shape `S + S + A`. The signature of tranistion probability function is presented below:

  ```python
      def transition_fn(t: int, L_t: torch.Tensor) -> torch.Tensor:
          ...
  ```

In order to create a custom environment, you can define each one of the above-mentioned attributes and pass them to `Environment`. Let's take a look at the environment Random Linear, which is a custom environment already implemented in the library.

We first define the states and actions. We want to have n states and n actions. Therefore, S=(n,) and A=(n,). Also, we use a uniform initial state distribution. To get a specific instance, we consider n=5.

Now, we define the reward and transition probability functions. As the name of environment suggests, we want the reward and transition probabilities to be a random linear (affine indeed) function of the mean-field, that is given the mean field $L$, the reward and tranistion probabilities should be equal to $M_1 \times L + M_2$ for some randomly generated matrices $M_1, M_2$. We generate different pairs of matrices for reward and transition functions. Note that in order for transition probabilities to be well-defined, we apply a softmax function to the output of the affine function. Furthermore, we restrict all the entries of the randomly generated matrices to be in [-m, m]. With this constraint, it is fairly straightforward to see that the absolute value of reward cannot be larger than 2*m implying that we should set r_max=2*m. To get an environment instance, we set m=1.

By passing all the required attributes to the Environment class, we define and instantiate the Random Linear environmentas shown in the code snippet presented below:

```python
import torch

from mfg.env import Environment

# Environment parameters
T = 4 # time horizon
n = 5
m = 1

# State and action space shapes
S = (n,)
A = (n,)

# Initial state distribution
mu0 = torch.ones(n) / n

# Reward and transition probability functions
torch.manual_seed(0)

r1 = 2 * m * torch.rand(n, n) - m  # M_1 for reward_fn
r2 = 2 * m * torch.rand(n, n) - m  # M_2 for reward_fn
reward_fn = lambda env, t, L_t: r1 @ L_t + r2

p1 = 2 * m * torch.rand(n, n, n) - m  # M_1 for transition_fn
p2 = 2 * m * torch.rand(n, n, n) - m  # M_2 for transition_fn
transition_fn = lambda env, t, L_t: torch.nn.Softmax(dim=-1)(p1 @ L_t + p2
    )

# Define the custom environment
user_defined_random_linear = Environment(
    T=T
    S=S,
    A=A,
    mu0=mu0,
    r_max=2 * m,
    reward_fn=reward_fn,
    transition_fn=transition_fn,
)
```

Note that by varying `T`, `n`, and `m`, we can create different environment instances. `MFGLib` offers this environment as one of the pre-implemented environments—the class method `Environment.random_linear()`. The MFGLib implementation of Random Linear is an alternative class-based implementation.

## Pre-Implemented Environments

MFGLib comes with 10 pre-implemented environments which can be accessed by calling the corresponding classmethods of the `Environment` class. The pre-implemented environments are listed below:

- **Beach Bar:** The beach bar process is an MDP with $|\mathcal{X}|$ states disposed on a one dimensional torus ($\mathcal{X} = \{0, \ldots, |\mathcal{X}| - 1\}$), which represents a beach. A bar is located in one of the states. As the weather is very hot, players want to be as close as possible to the bar, while keeping away from too crowded areas. See [113] for details. This environment can be accessed by calling the classmethod `Environment.beach_bar()` with the following signature:

```
    def beach_bar(
        cls,
        T: int = 2,
        n: int = 4,
        bar_loc: int = 2,
        log_eps: float = 1e-20,
        p_still: float = 0.5,
        mu0: Literal["uniform"] | torch.Tensor = "uniform",
    ) -> Environment:
        ...
```

  `n` determines the number of bar locations. `bar_loc` is the exact bar location (a number between 0 and `n-1`). `log_eps` determines the parameter $\epsilon$ which is used in computing the logarithms. Specifically, we replace $\log(x)$ with $\log(x + \epsilon)$ to enforce a lower bound. `p_still` is the probability of a player staying still. The probability of them going to the left or right will be `(1 - p_still) / 2`.

- **Building Evacuation:** In this problem, there is a multilevel building and each agent of the crowd wants to go downstairs as quickly as possible while favoring social distancing. At each floor, two staircases are located at two opposite corners, such as the crowd has to cross the whole floor to take the next staircase. Each agent can remain in place, move in the 4 directions (up, down, right, left) as well as go up or down when on a staircase location. See [112] for details. This environment can be accessed by calling the classmethod `Environment.building_evacuation()` with the following signature:

```python
    def building_evacuation(
        cls,
        T: int = 3,
        n_floor: int = 5,
        floor_l: int = 10,
        floor_w: int = 10,
        log_eps: float = 1e-20,
        eta: float = 1.0,
        evac_r: float = 10.0,
        mu0: Literal["uniform"] | torch.Tensor = "uniform",
    ) -> Environment:
        ...
```

`n_floor` determines the number of floors. `floor_l` and `floor_w` determine the length and width of the floors, respectively. `log_eps` is used to replace $\log(x)$ with $\log(x+\epsilon)$. `eta` is a parameter of the reward function. `evac_r` is the reward received upon getting to the first floor.

- **Conservative Treasure Hunting:** This environment provides an example of an MFG that is neither contractive nor strictly monotone, and has multiple NE solutions. See [50] for details. This environment can be accessed by calling the classmethod `Environment.conservative_treasure_hunting()` with the following signature:

```python
    def conservative_treasure_hunting(
        cls,
        T: int = 5,
        n: int = 3,
        r: tuple[float, ...] = (1.0, 1.0, 1.0),
        c: tuple[float, ...] = (1.0, 1.0, 1.0, 1.0, 1.0),
        mu0: Literal["uniform"] | torch.Tensor = "uniform",
    ) -> Environment:
        ...
```

$n$ determines the number of states and actions. $r$ is the vector of $r^i$ for $i = 1, \ldots, n$. $c$ is the vector of $C_t$ for $t = 1, 2, \ldots, T$.

- **Crowd Motion:** This environment extends the Beach Bar environment in 2 dimensions. Our implementation is an adaptation of the crowd motion environment introduced in [112]. This environment can be accessed by calling the classmethod `Environment.crowd_motion()` with the following signature:

```python
    def crowd_motion(
        cls,
        T: int = 3,
        torus_l: int = 20,
        torus_w: int = 20,
        loc_change_freq: int = 2,
        c: float = 10.0,
        log_eps: float = 1e-10,
        p_still: float = 0.5,
        seed: int = 0,
        mu0: Literal["uniform"] | torch.Tensor = "uniform",
    ) -> Environment:
        ...
```

`torus_l` and `torus_w` determine the length and width of the bar location grid, respectively. `loc_change_freq` specifies the length of the interval between two consecutive bar location changes—in constrst to the Beach Bar environment, the bar location is not fixed in this problem. `log_eps` and `p_still` work as in the Beach Bar environment.

- **Equilibrium Price:** In this problem, a large number of homogeneous firms producing the same product under perfect competition are considered. The price of the product is determined endogenously by the supply-demand equilibrium. Each firm, meanwhile, maintains a certain inventory level of the raw materials for production, and decides about the quantity of raw materials to consume for production and the quantity of raw materials to replenish the inventory. See [51] for details. This environment can be accessed by calling the classmethod `Environment.equilibrium_price()` with the following signature:

```
     def equilibrium_price(
        cls,
        T: int = 4,
        s_inv: int = 3,
        Q: int = 2,
        H: int = 2,
        d: float = 1.0,
        e0: float = 1.0,
        sigma: float = 1.0,
        c: tuple[float, float, float, float, float] = (1.0, 1.0, 1.0,
    1.0, 1.0),
        mu0: Literal["uniform"] | torch.Tensor = "uniform",
    ) -> Environment:
        ...
```

`s_inv` determines the maximum inventory level. `Q` is the maximum amount of raw material to consume. `H` represents the maximum amount of raw material to replenish. `d, e0`, `sigma` are parameters of the supply-demand equilibrium equation. `c` is a vector of reward function coefficients.

- **Left-Right**: In this problem, a large number of agents choose simultaneously between going left or right. Afterwards, each agent shall be punished proportional to the number of agents that chose the same action, but more-so for choosing right than left. See [32] for details. This environment can be accessed by calling the classmethod `Environment.left_right()` with the following signature:

```
    def left_right(
        cls, mu0: tuple[float, float, float] = (1.0, 0.0, 0.0)
    ) -> Environment:
        ...
```

- **Linear Quadratic:** See [113] for details. This environment can be accessed by calling the classmethod `Environment.linear_quadratic()` with the following signature:

```
    def linear_quadratic(
        cls,
        T: int = 3,
        el: int = 5,
        m: int = 2,
        sigma: float = 3.0,
        delta: float = 0.1,
        k: float = 1.0,
        q: float = 0.01,
        kappa: float = 0.5,
        c_term: float = 1.0,
        mu0: Literal["uniform"] | torch.Tensor = "uniform",
    ) -> Environment:
        ...
```

  `m` and `el` determine the number of states and actins, respectively. To be precise, the states are `{-m, ..., m}` and the actions are `{-l, ..., l}`. `sigma` and `k` are parameters of the state dynamics (they impact the transition probabilities). `q`, `kappa`, and `c_term` are parameters of the reward function. `delta` appears in both the state dynamics and reward function.

- **Random Linear:** It is a custom environment as discussed in the previous subsection. It can be accessed by calling the classmethod `Environment.random_linear()` with the following signature:

```
    def random_linear(
        cls,
        T: int = 3,
        n: int = 5,
        m: float = 10.0,
        seed: int = 0,
        mu0: Literal["uniform"] | torch.Tensor = "uniform",
    ) -> Environment:
        ...
```

- **Rock Paper Scissors:** This game is inspired by Shapley (1964) and their generalized non-zero-sum version of Rock-Paper-Scissors, for which classical fictitious play would not converge. Each of the agents can choose between rock, paper and scissors, and obtains a reward proportional to double the number of beaten agents minus the number of agents beating the agent. See [32] for details. This environment can be accessed by calling the classmethod `Environment.rock_paper_scissors()` with the following signature:

```
    def rock_paper_scissors(
        cls, T: int = 1, mu0: tuple[float, float, float, float] =
    (1.0, 0.0, 0.0, 0.0)
    ) -> Environment:
        ...
```

- **Susceptible Infected:** In this problem, a large number of agents can choose between social distancing or going out. If a susceptible agent chooses social distancing, they may not become infected. Otherwise, an agent may become infected with a probability proportional to the number of agents being infected. If infected, an agent will recover with a fixed chance every time step. Both social distancing and being infected have an associated cost. See [32] for more details. This environment can be accessed by calling the classmethod `Environment.susceptible_infected()` with the following signature:

```
    ef susceptible_infected(
        cls, T: int = 50, mu0: tuple[float, float] = (0.4, 0.6)
    ) -> Environment:
        ...
```

The implemented environments encompass a wide variety MFGs. These environments include problems like Beach Bar, where only the reward function depends on the mean-field, as well as environments like Conservative Treasure Hunting, where both rewards and transition probabilities depend on the mean-field.

Furthermore, as discussed above, all implemented environments take initialization parameters that modify the resulting instance in terms of state and action space, underlying reward and transition probabilities, etc. This way, one can generate infinitely many environments with varying sizes to experiment with. Let's look at the Building Evacuation environment for example. We can create distinct buildings (distinct environment instances) by changing the number of floors, the size of each floor, etc. In the following, we create two distinct buildings, one with 10 floors each 20 by 20, and another one with 100 floors each 50 by 5.

```
from mfglib.env import Environment
building_evacuation_1 = Environment.building_evacuation(n_floor=10,
    floor_l=20, floor_w=20)
building_evacuation_2 = Environment.building_evacuation(n_floor=100,
    floor_l=50, floor_w=5)
```

# Chapter 3

# Escaping Saddle Points Efficiently with Occupation-Time-Adapted Perturbations

Gradient descent (GD), which dates back to [24], aims to minimize a function $f : \mathbb{R}^d \to \mathbb{R}$ via the iteration: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t), t = 0, 1, 2, \ldots$, where $\eta > 0$ is the step size and $\nabla f$ is the gradient of $f$. Due to its simple form and fine computational properties, GD and its variants (e.g., stochastic gradient descent) are essential for many machine learning tools: principle component analysis [22], phase retrieval [20], and deep neural network [120], just to name a few. In the era of data deluge, many problems are concerned with large-scale optimization in which the intrinsic dimension $d$ is large. GD turns out to be efficient in dealing with high-dimensional convex optimization, where the first-order stationary point $\nabla f(\boldsymbol{x}) = 0$ is necessarily the global minimum point. Algorithmically, it involves finding a point with small gradient $\|\nabla f(\boldsymbol{x})\| < \epsilon$. A classical result of [104] showed that the time required by GD to find such a point in a possibly non-convex problem is of order $\epsilon^{-2}$, independent of the dimension $d$.

In non-convex settings, applying GD will still lead to an approximate first-order stationary point. However, this is not sufficient: for non-convex functions, first-order stationary points can be either global minimum, local minimum, local maximum, or saddle points. As we will explain, saddle points are the main bottleneck for GD in many non-convex problems. The goal of this chapter is therefore to develop efficient algorithms to escape saddle points in high-dimensional non-convex problems, and hence overcome the curse of dimensionality.

**Escape local minima:** Inspired by annealing in metallurgy, [80] developed simulated annealing to approximate the global minimum of a given function. [48] proposed a diffusion simulated annealing and proved that it converges to the set of global minimum points. However, subsequent works [59, 98, 100, 13, 101, 131] revealed that it might take an exponentially long time (of order $\exp(d)$) for diffusion simulated annealing to get close to the global minimum. See [130] for a review. Some work, e.g., methods based on Lévy flights [108] or Cuckoo's search [148] showed empirically faster convergence to the global minimum. Yet the theory of these approaches is far-fetched. Closely related to simulated annealing are recent efforts in approximating the global minimum in non-convex problems via Langevin dynamics-based stochastic gradient descent [117, 26], where the gradient is evaluated at a randomly selected data point in each iteration. There also exist several variants of Langevin-based stochastic gradient descent using non-reversibility [60] and replica exchange [27, 37]. Typically, these algorithms take polynomial time in the dimension $d$, and thus may scale poorly when $d$ is large.

**Escape saddle points:** Fortunately, in many non-convex problems, it suffices to find a local minimum. Indeed, there has been a line of recent work arguing that local minima are less problematic, and that for many non-convex problems there are no spurious local minima. That is, all local minima are comparable in value with the global minimum. Examples include tensor decomposition [47, 44, 46, 123], semidefinite programming [7, 97], dictionary learning [128], phase retrieval [127], robust regression [96], low-rank matrix factorization [15, 43, 45, 107], and certain classes of deep neural networks [28, 38, 76, 77, 90, 106, 139, 147]. Nevertheless, as shown in [35, 39, 67], saddle points may correspond to suboptimal solutions, and it may take exponentially long time to move from saddle points to a local minimum point. Meanwhile, it has been observed in empirical studies [34, 129] that GD and its variants such as stochastic gradient descent (SGD) [49] and Adam [79] may be trapped in saddle points.

[47] took the first step to show that by adding noise at each iteration, GD can escape all saddle points in polynomial time. Additionally, [40, 87] proved that with random initialization, GD converges to a local minimizer. Moreover, [70] proposed the perturbed gradient descent (PGD) algorithm, which [69] further improved to the perturbed accelerated gradient descent (PAGD) algorithm. They showed that PGD and PAGD are efficient – the time complexity is almost independent of the dimension $d$. See also [71] for a summary of results in this direction.

**Our idea.** Motivated by the "fast exploration" of self-repelling random walk, this chapter develops a new perturbation mechanism by adapting the perturbations to the history of states. Recall that [70, 71] used the following perturbation update when perturbation conditions hold:

$$\boldsymbol{x}_t' = \boldsymbol{x}_t + \mathrm{Unif}(B^d(\mathbf{0}, r)), \quad \boldsymbol{x}_{t+1} = \boldsymbol{x}_t' - \eta \nabla f(\boldsymbol{x}_t'),$$

where $\mathrm{Unif}(B^d(\mathbf{0}, r))$ is a point picked uniformly in the ball of radius $r$. On the empirical side, [103, 151] applied this idea of GD with noise to train deep neural networks. Our idea is to replace $\mathrm{Unif}(B^d(\mathbf{0}, r))$ with non-uniform perturbations, whose mechanism depends on the current state $\boldsymbol{x}_t$ and the history of states $\{\boldsymbol{x}_s;\, s \le t\}$. There are conceivably many ways to add non-uniform perturbation based on the current and previous states; here we choose to adapt perturbations to the "occupation time".

The intuition is illustrated by the one-dimensional function $f(x) = x^3$ (see Figure 3.0.1). There is a saddle point at 0, and imagine GD approaches 0 from the right. It can be shown that GD converges monotonically to a stationary point (see Appendix 3.A). The uniform perturbation will add noise with probability 1/2 both to the right and to the left. To the

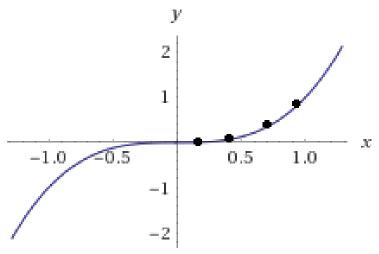


Figure 3.0.1: Illustration of occupation-time-adapted perturbation using $f(x) = x^3$.

right, GD will again get stuck at the saddle point 0. However, to the left, there is a possibility of escaping from 0 and finding a local minimum ($-\infty$ in this case). Therefore, it is reasonable to add noise with a larger probability to the left, since it has spent a long time on the right and has yet to explore the left side.

The previous intuition can be quantified via the notion of occupation times $L_t$ (the number of $\{x_s\}_{s<t}$ to the left of $x_t$) and $R_t$ (the number of $\{x_s\}_{s<t}$ to the right of $x_t$). By definition, $R_t + L_t = t$, for each $t = 0, 1, \ldots$. If $L_t$ is larger, the perturbation will push the iterate $x_t$ to the right; and if $R_t$ is larger, push to the left. More precisely,

$$x_{t+1} = \begin{cases} x_t - r\,\mathrm{Unif}(0,1) & \text{with probability } p, \\ x_t + r\,\mathrm{Unif}(0,1) & \text{with probability } 1-p, \end{cases} \tag{3.1}$$

where $p = \frac{w(R_t)}{w(L_t)+w(R_t)}$ and $w : \{0, 1, \ldots\} \to (0, \infty)$ is an increasing weight function on the nonnegative integers (e.g., $w(k) = 1 + k^\alpha$ for $\alpha > 0$).

The dynamics (3.1) is closely related to the vertex-repelling random walk defined by

$$Z_{t+1} = \begin{cases} Z_t - 1 & \text{with probability } \frac{w(\widetilde{R}_t)}{w(\widetilde{L}_t)+w(\widetilde{R}_t)}, \\ Z_t + 1 & \text{with probability } \frac{w(\widetilde{L}_t)}{w(\widetilde{L}_t)+w(\widetilde{R}_t)}, \end{cases} \tag{3.2}$$

where $\widetilde{R}_t := \{s < t : Z_s = Z_t + 1\}$ and $\widetilde{L}_t := \{s < t : Z_s = Z_t - 1\}$. This (non-Markovian) random walk model was introduced by [110] in the statistical physics literature. Based on the scaling arguments and simulations, it was conjectured that for $w(\cdot)$ with a suitable growth, the walk $(Z_t,\, t \ge 0)$ is recurrent and is further super-diffusive in the sense that $\mathbb{E}Z_t^2 \sim t^{\frac{4}{3}}$, whereas for a simple random walk $(S_t,\, t \ge 0)$ its exploration range is $\mathbb{E}S_t^2 \sim t \ll t^{\frac{4}{3}}$. These properties have only been proved rigorously for a simpler variant – the edge-repelling random walk, see [36, 134, 136, 135]. For instance, it was conjectured that for $w(k) \sim \lambda^k$ with $\lambda > 1$,

$$\left(\frac{Z_{nu}}{(\sigma n)^{\frac{2}{3}}},\, u \ge 0\right) \text{ converges in distribution to } (\mathcal{Z}_u,\, u \ge 0) \quad \text{as } n \to \infty,$$

where $\sigma > 0$ is a variance parameter depending on $w(\cdot)$, and the scaling limit $(\mathcal{Z}_u,\, u \ge 0)$ is a (universal) continuous process whose *marginal* distribution $p(u, \cdot)$ is given as follows. Let

$(|B_s^h|, s \in \mathbb{R})$ be a two-sided reflected Brownian motion with $|B_0^h| = h > 0$. Define

$$T_0^- := \sup\{s < 0 : |B_s^h| = 0\} \quad \text{and} \quad T_r^+ := \sup\{s > r : |B_s^h| = 0\} \text{ for } r > 0,$$

and

$$\mathcal{A}_r^h := \int_{T_0^-}^{T_r^+} |B_s^h| ds \quad (\text{area of } |B^h| \text{ between } T_0^- \text{ and } T_r^+).$$

See Figure 3.0.2 for an illustration.

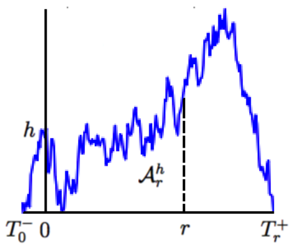


Figure 3.0.2: Illustration of $\mathcal{A}_r^h$.

Denote by $\rho(u, r; h) := \frac{\partial}{\partial u} \mathbb{P}\left(\mathcal{A}_r^h \le u\right)$ the density of $\mathcal{A}_r^h$. Then

$$p(u, x) = \int_0^\infty \rho\left(\frac{u}{2}, |x|; h\right) dh \quad \text{for } x \in \mathbb{R}. \tag{3.3}$$

Clearly, the conjecture implies that $\mathbb{E}Z_t^2$ is of order $t^{\frac{4}{3}}$. It is easy to see from (3.3) that $\mathcal{Z}_u$ has the same distribution as $r^{-\frac{2}{3}}\mathcal{Z}_{ru}$ for each $u > 0$. However, the distribution of $(\mathcal{Z}_u,\, u \ge 0)$ at the *process level* remains open to date, and we even don't know whether the process $\mathcal{Z}$ is a diffusion process. A counterpart to the vertex-repelling walk is the vertex-reinforced walk [111, 142] defined by $Z_{t+1} = Z_t - 1$ with probability $\frac{w(\widetilde{L}_t)}{w(\widetilde{L}_t)+w(\widetilde{R}_t)}$, and $Z_{t+1} = Z_t + 1$ with probability $\frac{w(\widetilde{R}_t)}{w(\widetilde{L}_t)+w(\widetilde{R}_t)}$. It is well known [132, 142] that vertex-reinforced random walk exhibits localization at a finite number of points for some choices of $w(\cdot)$, e.g., $w(k) \sim k^\alpha$ with $\alpha \ge 1$.

**Our results.** We will first show that vertex-repelling walk will never be localized or stuck at some points in contrast with vertex-reinforced walk (see Theorem 4). The non-localization and the (conjectured) super-diffusive properties of the vertex-repelling walk (3.2) facilitate exploration, and thus the corresponding perturbation scheme (3.1) makes it more likely to escape from saddle points.

We will then propose a new perturbation mechanism based on the dynamics (3.1), which can be integrated into the framework of (any) perturbation-based optimization algorithms. In particular, integrating the above-mentioned mechanism into the framework of PGD and PAGD, we propose two new algorithms: perturbed gradient descent adapted to occupation time (PGDOT, Algorithm 1) and its accelerated version, perturbed accelerated gradient descent adapted to occupation time (PAGDOT, Algorithm 2).

We will prove that Algorithm 1 (resp. Algorithm 2) converges to a second-order stationary point at least as fast as PGD (resp. PAGD). Algorithms 1 and 2 are state-dependent adaptive algorithms, perturbing GD and accelerated gradient descent (AGD) [105] non-uniformly according to the history of states.

**Algorithm 1** Perturbed Gradient Descent Adapted to Occupation Time (Meta Algorithm)

```
for $t = 0, 1, \dots$ do
    if perturbation condition holds then
        for $i = 1, \dots, d$ do
            $L_t^i \leftarrow \#\{s < t : x_s^i \le x_t^i\}$
            $R_t^i \leftarrow \#\{s < t : x_s^i > x_t^i\}$
            $x_t^i \leftarrow \begin{cases} x_t^i - \frac{r}{\sqrt{d}}\,\mathrm{Unif}(0,1) & \text{w.p. } p, \\ x_t^i + \frac{r}{\sqrt{d}}\,\mathrm{Unif}(0,1) & \text{w.p. } 1-p, \end{cases}$
            where $p = \frac{w(R_t^i)}{w(L_t^i)+w(R_t^i)}$
        end for
    end if
    $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta \nabla f(\boldsymbol{x}_t)$
end for
```

**Algorithm 2** Perturbed Accelerated Gradient Descent Adapted to Occupation Time (Meta Algorithm)

```
for $t = 0, 1, \dots,$ do
    if perturbation condition holds then
        for $i = 1, \dots, d$ do
            $L_t^i \leftarrow \#\{s < t : x_s^i \le x_t^i\}$
            $R_t^i \leftarrow \#\{s < t : x_s^i > x_t^i\}$
            $x_t^i \leftarrow \begin{cases} x_t^i - \frac{r}{\sqrt{d}}\mathrm{Unif}(0,1) & \text{w.p. } p, \\ x_t^i + \frac{r}{\sqrt{d}}\mathrm{Unif}(0,1) & \text{w.p. } 1-p, \end{cases}$   where $p = \frac{w(R_t^i)}{w(L_t^i)+w(R_t^i)}$
        end for
    end if
    $\boldsymbol{x}_{t+1} \leftarrow \mathrm{Accelerate}(\boldsymbol{x}_t, \boldsymbol{v}_t), \quad \boldsymbol{v}_{t+1} \leftarrow \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$
end for
```

We will finally corroborate our theoretical analysis by experimental results. Specifically, we will empirically demonstrate that Algorithms 1 and 2 exhibit faster escape from saddle points, surpassing not only their counterparts (PGD and PAGD) but also outperforming SGD and several adaptive gradient methods, including Adam, AMSGrad [119], AdaBelief [152], and STORM [33] in training deep learning models on popular datasets such as MNIST [86], CIFAR-10 [82], and CIFAR-100 [82].

**Notations:** Below we collect the notations that will be used throughout this chapter. For $S$ a finite set, let $\#S$ denote the number of elements in $S$. For $D$ as a domain, let $\mathrm{Unif}(D)$ be the uniform distribution on $D$, e.g., $\mathrm{Unif}(0,1)$ is the uniform distribution on $[0,1]$. For a function $f : \mathbb{R}^d \to \mathbb{R}$, let $\nabla f$ and $\nabla^2 f$ denote its gradient and Hessian, and $f^\star := \min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x})$ denote its global minimum. For $\boldsymbol{A}$ a symmetric matrix, let $\lambda_{\min}(\boldsymbol{A})$ be its minimum eigenvalue.

The notation $\|\cdot\|$ is used for both the Euclidean norm of a vector and the spectral norm of a matrix. For $\boldsymbol{x} = (x^1, \dots, x^d)$ and $r > 0$, let $B^d(\boldsymbol{x}, r) := \{\boldsymbol{y} : \|\boldsymbol{y} - \boldsymbol{x}\| \le r\}$ be the $d$-dimensional ball centered at $\boldsymbol{x}$ with radius $r$, and $C^d(\boldsymbol{x}, r) := \{\boldsymbol{y} : |y^i - x^i| \le r \text{ for } 1 \le i \le d\}$ be the $d$-dimensional hypercube centered at $\boldsymbol{x}$ with distance $r$ to each of its surfaces. We use the symbol $O(\cdot)$ to hide only absolute constants which do not depend on any problem parameter.

The rest of the chapter is organized as follows. Section 3.1 provides background on the continuous optimization and recalls some existing results. Section 3.2 presents the main results. Section 3.3 contains numerical experiments to corroborate our analysis. Section 3.4 concludes.

## 3.1 Background and Existing Results

### Results of GD

We consider non-convex optimization (convex optimization results are recalled in Appendix 3.B). In this case, it is generally difficult to find the global minima. A popular approach is to consider the first-order stationary points instead.

**Definition 2.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. We say that $(i)$ $\boldsymbol{x}$ is a first-order stationary point of $f$ if $\nabla f(x) = 0$; $(ii)$ $\boldsymbol{x}$ is an $\epsilon$-first-order stationary point of $f$ if $\|\nabla f(x)\| \le \epsilon$.*

We say that a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz if $\|\nabla f(\boldsymbol{x}_1) - \nabla f(\boldsymbol{x}_2)\| \le \ell \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$. For gradient Lipschitz functions, GD converges to the first-order stationary points, which is quantified by the following theorem from [104][Section 1.2.3].

**Theorem 1.** *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz. For any $\epsilon > 0$, if we run GD with step size $\eta = \ell^{-1}$, then the number of iterations to find an $\epsilon$-first-order stationary point is $\frac{\ell(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^2}$.*

Note that in Theorem 1, the time complexity of GD is independent of the dimension $d$. For a non-convex function, a first-order stationary point can be either a local minimum, a saddle point, or a local maximum. The following definition is taken from [70][Definition 4].

**Definition 3.** *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. We say that $(i)$ $\boldsymbol{x}$ is a local minimum if $\boldsymbol{x}$ is a first-order stationary point, and $f(\boldsymbol{x}) \le f(\boldsymbol{y})$ for all $\boldsymbol{y}$ in some neighborhood of $\boldsymbol{x}$; $(ii)$ $\boldsymbol{x}$ is a saddle point if $\boldsymbol{x}$ is a first-order stationary point but not a local minimum. Assume further that $f$ is twice differentiable. We say a saddle point $\boldsymbol{x}$ is strict if $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) < 0$.*

For a twice differentiable function $f$, note that $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \le 0$ for any saddle point $\boldsymbol{x}$. So by assuming a saddle point $\boldsymbol{x}$ to be strict, we rule out the case $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) = 0$. The next subsection will review two perturbation-based algorithms that allow jumping out of strict saddle points.

## Results of PGD and PAGD

One drawback of GD in non-convex optimization is that it may get stuck at saddle points. [70] and [69] proposed PGD and PAGD, respectively, to escape saddle points, which we review here. To proceed further, we need some vocabulary regarding the Hessian of the function $f$.

**Definition 4.** *A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is $\rho$-Hessian Lipschitz if $\|\nabla^2 f(\boldsymbol{x}_1) - \nabla^2 f(\boldsymbol{x}_2)\| \le \rho \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$. Furthermore, we say that $(i)$ $\boldsymbol{x}$ is a second-order stationary point of $f$ if $\nabla f(\boldsymbol{x}) = 0$ and $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \ge 0$; $(ii)$ $\boldsymbol{x}$ is a $\epsilon$-second-order stationary point of $f$ if $\|\nabla f(\boldsymbol{x})\| \le \epsilon$ and $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \ge -\sqrt{\rho\epsilon}$.*

To simplify the presentation, assume that all saddle points are strict (Definition 3). In this situation, all second-order stationary points are local minima. The basic idea of these two algorithms is as follows. Imagine that we are currently at an iterate $\boldsymbol{x}_t$ which is not an $\epsilon$-second-order stationary point. There are two scenarios: $(i)$ The gradient $\|\nabla f(\boldsymbol{x}_t)\|$ is large and a usual iteration of GD or AGD is enough; $(ii)$ The gradient $\|\nabla f(\boldsymbol{x}_t)\|$ is small but $\lambda_{\min}(\nabla^2 f(\boldsymbol{x}_t)) \le -\sqrt{\rho\epsilon}$ (large negative). So $\boldsymbol{x}_t$ is around a saddle point, and a perturbation $\xi$ is needed to escape from the saddle region: $\widetilde{\boldsymbol{x}}_t = \boldsymbol{x}_t + \xi$.

The main result for PGD, Theorem 3 in [70], and for PAGD, Theorem 3 in [69], are stated below showing that the time complexity of these two algorithms are almost dimension-free (with a log factor).

**Theorem 2.** *[70] Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz and $\rho$-Hessian Lipschitz. Then there exists $c_{\max} > 0$ such that for any $\delta > 0$, $\epsilon \le \ell^2/\rho$, $\Delta_f \ge f(\boldsymbol{x}_0) - f^*$, and $c \le c_{\max}$, PGD outputs an $\epsilon$-second-order stationary point with probability $1 - \delta$, terminating within the following number of iterations:*

$$O\left(\frac{\ell(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^2} \log^4\left(\frac{d\ell\Delta_f}{\epsilon^2\delta}\right)\right).$$

Compared with Theorem 1, PGD takes almost the same order of time to find a second-order stationary point as GD does to find a first-order stationary point.

**Theorem 3.** *[69] Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz and $\rho$-Hessian Lipschitz. Then there exists an absolute constant $c_{\max} > 0$ such that for any $\delta > 0$, $\epsilon \le \ell^2/\rho$, $\Delta_f \ge f(\boldsymbol{x}_0) - f^\star$, and $c \ge c_{\max}$, with probability $1 - \delta$, one of the iterates $\boldsymbol{x}_t$ of PAGD will be an $\epsilon$-second-order stationry point in the following number of iterations:*

$$O\left(\frac{\ell^{1/2}\rho^{1/4}(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^{7/4}} \log^6\left(\frac{d\ell\Delta_f}{\rho\epsilon\delta}\right)\right).$$

## 3.2 Main Results

In this section, we first prove the non-localization property of the vertex-repelling random walk. Then, we formalize the idea of perturbations adapted to occupation time and provide the full version of PGDOT and PAGDOT in Algorithms 3 and 4, respectively. Our main results show that these algorithms converge rapidly to second-order stationary points.

### Non-Localization Property of Vertex-Repelling Random Walk

The following theorem suggests that the new perturbation mechanism helps perturbation-based algorithms to avoid getting stuck at saddle points, as the dynamics of vertex-repelling random walk prescribed in (3.1) does not localize.

**Theorem 4.** *Let $\{Z_t, t = 0, 1, \ldots\}$ be the vertex-repelling random walk defined by (3.2), where $w : \{0, 1, \ldots\} \to (0, \infty)$ is an increasing function such that $w(n) \to \infty$ as $n \to \infty$. Then*

$$\mathbb{P}\left(\exists t_0 > 0,\ k \le \ell : Z_t \in \{k, \ldots, \ell\} \text{ for all } t \ge t_0\right) = 0.$$

The proof of this theorem is given in Appendix 3.C.

### Perturbed Gradient Descent Adapted to Occupation Time

PGD adds a uniform random perturbation when stuck at saddle points. From the discussion in the introduction, it is more reasonable to perturb with non-uniform noise whose distribution depends on the occupation times. Recall that $w : \{0, 1, \ldots\} \to (0, \infty)$ is an increasing weight function on the nonnegative integers. The following algorithm adapts PGD to random perturbation depending on the occupation dynamics. We follow the parameter setting as in [70]. Our algorithm performs GD with step size $\eta$ and gets a perturbation of amplitude $\frac{r}{\sqrt{d}}$ near saddle points at most once every $t_{\text{thres}}$ iterations. The threshold $t_{\text{thres}}$ ensures that the dynamics of the algorithm is mostly GD. The threshold $g_{\text{thres}}$ determines if a perturbation is needed, and the threshold $f_{\text{thres}}$ decides when the algorithm terminates.

The next theorem gives the convergence rate of Algorithm 3: PGDOT finds a second-order stationary point in the same number of iterations (up to a constant factor) as PGD does.

**Algorithm 3** Perturbed Gradient Descent Adapted to Occupation Time PGDOT($\boldsymbol{x}_0, \ell, \rho, \epsilon, c, \delta, \Delta_f, w$)

```
$\chi \leftarrow 3\max\left\{\log(\frac{d\ell\Delta_f}{c\epsilon^2\delta}), 4\right\},\ \eta \leftarrow \frac{c}{\ell},\ r \leftarrow \frac{\epsilon\sqrt{c}}{\chi^2\ell}$
$g_{\text{thres}} \leftarrow \frac{\epsilon\sqrt{c}}{\chi^2},\ f_{\text{thres}} \leftarrow \frac{c}{\chi^3}\sqrt{\frac{\epsilon^3}{\rho}},\ t_{\text{thres}} \leftarrow \frac{\chi\ell}{c^2\sqrt{\rho\epsilon}}$
$t_{\text{noise}} \leftarrow -t_{\text{thres}} - 1$
for $t = 0, 1, \dots$ do
  if $\|\nabla f(\boldsymbol{x}_t)\| \le g_{\text{thres}}$ and $t - t_{\text{noise}} > t_{\text{thres}}$ then
    $\widetilde{\boldsymbol{x}}_t \leftarrow \boldsymbol{x}_t,\ t_{\text{noise}} \leftarrow t$
    for $i = 1, \dots, d$ do
      $L_t^i \leftarrow \#\{s < t : x_s^i \le x_t^i\}$
      $R_t^i \leftarrow \#\{s < t : x_s^i > x_t^i\}$
      $x_t^i \leftarrow \begin{cases} \widetilde{x}_t^i - \frac{r}{\sqrt{d}}\text{Unif}(0,1) & \text{w.p. } p, \\ \widetilde{x}_t^i + \frac{r}{\sqrt{d}}\text{Unif}(0,1) & \text{w.p. } 1-p, \end{cases}$ where $p = \frac{w(R_t^i)}{w(L_t^i) + w(R_t^i)}$
    end for
  end if
  if $t - t_{\text{noise}} = t_{\text{thres}}$ and $f(\boldsymbol{x}_t) - f(\widetilde{\boldsymbol{x}}_{t_{\text{noise}}}) > -f_{\text{thres}}$ then
    return $\widetilde{\boldsymbol{x}}_{t_{\text{noise}}}$
  end if
  $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \eta\nabla f(\boldsymbol{x}_t)$
end for
```

**Theorem 5.** *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz and $\rho$-Hessian Lipschitz. Then there exists $c_{\max} > 0$ such that for any $\delta > 0$, $\epsilon \le \ell^2/\rho$, $\Delta_f \ge f(\boldsymbol{x}_0) - f^*$, and $c \le c_{\max}$, PGDOT (Algorithm 3) outputs an $\epsilon$-second-order stationary point with probability $1 - \delta$ terminating within the following number of iterations:*

$$O\left(\frac{\ell(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^2}\log^4\left(\frac{d\ell\Delta_f}{\epsilon^2\delta}\right)\right).$$

The proof of Theorem 5 is based on a geometric characterization of saddle points – thin pancake property [70]. In Appendix 3.D, we will discuss this property, and show how it is used to prove Theorem 5.

## Perturbed Accelerated Gradient Descent Adapted to Occupation Time

Similar to the way we combined our perturbation mechanism with PGD, we can adapt PAGD to this mechanism as well resulting in the accelerated version of PGDOT (Algorithm 4). We follow the parameter setting as in [69].

Algorithm 4, similar to PAGD, employs a feature called Negative Curvature Exploitation (NCE) which resets the momentum and decides whether to exploit the negative curvature when the function becomes “too convex". See [69][Algorithm 3].

**Algorithm 4** Perturbed Accelerated Gradient Descent Adapted to Occupation Time PAGDOT($\boldsymbol{x}_0, \eta, \theta, \gamma, s, r, \mathscr{T}, w$)

```
$\chi \leftarrow \max\left\{\log(\frac{d\ell\Delta_f}{\rho\epsilon\delta}), 1\right\}$,  $\kappa \leftarrow \frac{\ell}{\sqrt{\rho\epsilon}}$,  $\eta \leftarrow \frac{1}{4\ell}$
$\theta \leftarrow \frac{1}{4\sqrt{\kappa}}$,  $\gamma \leftarrow \frac{\theta^2}{\eta}$,  $s \leftarrow \frac{\gamma}{4\rho}$,  $r \leftarrow \frac{\eta\epsilon}{\chi^5 c^8}$,  $\mathscr{T} \leftarrow \chi c \sqrt{\kappa}$
$\boldsymbol{v}_0 \leftarrow 0$
for $t = 0, 1, \ldots,$ do
    if $\|\nabla f(\boldsymbol{x}_t)\| \le \epsilon$ and no perturbation in last $\mathscr{T}$ steps then
        $\widetilde{\boldsymbol{x}}_t \leftarrow \boldsymbol{x}_t$
        for $i = 1, \ldots, d$ do
            $L_t^i \leftarrow \#\{s < t : x_s^i \le x_t^i\}$
            $R_t^i \leftarrow \#\{s < t : x_s^i > x_t^i\}$
            $x_t^i \leftarrow \begin{cases} \widetilde{x}_t^i - \frac{r}{\sqrt{d}}\mathrm{Unif}(0,1) & \text{w.p. } p, \\ \widetilde{x}_t^i + \frac{r}{\sqrt{d}}\mathrm{Unif}(0,1) & \text{w.p. } 1-p, \end{cases}$  where $p = \frac{w(R_t^i)}{w(L_t^i)+w(R_t^i)}$
        end for
    end if
    $\boldsymbol{y}_t \leftarrow \boldsymbol{x}_t + (1-\theta)\boldsymbol{v}_t$
    $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{y}_t - \eta\nabla f(\boldsymbol{y}_t)$
    $\boldsymbol{v}_{t+1} \leftarrow \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$
    if $f(\boldsymbol{x}_t) \le f(\boldsymbol{y}_t) + \langle \nabla f(\boldsymbol{y}_t), \boldsymbol{x}_t - \boldsymbol{y}_t \rangle - \frac{\gamma}{2}\|\boldsymbol{x}_t - \boldsymbol{y}_t\|^2$ then
        $(\boldsymbol{x}_{t+1}, \boldsymbol{v}_{t+1}) \leftarrow \mathrm{NCE}(\boldsymbol{x}_t, \boldsymbol{v}_t, s)$
    end if
end for
```

The next theorem gives the convergence rate of Algorithm 4: PAGDOT finds a second-order stationary point in the same number of iterations (up to a constant factor) as PAGD does, and therefore achieves a faster convergence rate than PGD and PGDOT. The proof of Theorem 6 is similar to that of Theorem 5.

**Theorem 6.** *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz and $\rho$-Hessian Lipschitz. Then there exists an absolute constant $c_{\max} > 0$ such that for any $\delta > 0$, $\epsilon \le \ell^2/\rho$, $\Delta_f \ge f(\boldsymbol{x}_0) - f^*$, and $c \ge c_{\max}$, one of the iterates $\boldsymbol{x}_t$ of PAGDOT (Algorithm 4) will be an $\epsilon$-second-order stationry point in the following number of iterations, with probability $1 - \delta$:*

$$O\left(\frac{\ell^{1/2}\rho^{1/4}(f(\boldsymbol{x}_0) - f^\star)}{\epsilon^{7/4}} \log^6\left(\frac{d\ell\Delta_f}{\rho\epsilon\delta}\right)\right).$$

It is worth mentioning that the new algorithms can be regarded as generalizations of PGD and PAGD. This is exemplified by the fact that when the weight function $w$ is a constant function (e.g. $w(k) = 1, \forall k \geq 0$), the perturbations become uniform. Additionally, Algorithms 3 and 4 share some spirit with GD with momentum methods such as the heavy ball method [115]. In the heavy ball method, a momentum term, which is a function of the current and previous states, is explicitly added to control the oscillations and accelerate in low curvatures along the direction close to momentum. In Algorithms 3 and 4, however, no explicit momentum term is added. Instead, the perturbation is adapted to the history of states providing the current state with an explicit direction.

We also remark that Theorem 5 (Theorem 6) has been proven using the exact same steps and lemmas as the ones employed in proving Theorem 2 (Theorem 3). As a result, the convergence rate of PGDOT (PAGDOT) is identical to that of PGD (PAGD). This equivalence extends to the hidden constants as well. However, we believe that by exploiting the intrinsic properties of the embedded perturbation mechanism, such as the super-diffusivity of the corresponding random walk, there is a possibility to enhance the theoretical results. Nonetheless, formally establishing this improvement remains an open question.

## 3.3 Empirical Results

This section presents empirical results to corroborate the theoretical analysis presented in the previous section. The experiments showcase the effectiveness of our new perturbation-based algorithms, not only in escaping saddle points but also in doing so efficiently and rapidly. To comprehensively evaluate their performance, we consider both small-scale and large-scale problems to demonstrate the practicality and scalability of the new algorithms. Additionally, we explore an MFG example to establish a link with the preceding chapter and demonstrate the potential of our proposed perturbation mechanism in assisting optimization-based MFG algorithms to efficiently find NE solutions.

The small-scale problems include a synthetic problem, a nonlinear regression problem, a regularized quadratic problem, and a phase retrieval problem, wherein we compare the new algorithms with their counterparts, PGD and PAGD, and vanilla GD. In the large-scale problems, we focus on image classification tasks using popular datasets such as MNIST [86], CIFAR-10 [82], and CIFAR-100 [82]. We compare the performance of the new algorithms against their counterparts, as well as SGD [49] and several state-of-the-art adaptive gradient algorithms including Adam [79], AMSGrad [119], AdaBelief [152], and STORM [33].

In these experiments, we use $L_t^i(h) := \#\{t - t_{\text{count}} \leq s < t : x_t^i - h \leq x_s^i \leq x_t^i\}$ and $R_t^i(h) := \#\{t - t_{\text{count}} \leq s < t : x_t^i < x_s^i \leq x_t^i + h\}$ instead of $L_t^i$ and $R_t^i$ in Algorithms 3 and 4. Here $h$ is a hyperparameter characterizing the occupation time over a small interval. $t_{\text{count}}$ is another hyperparameter prescribing how long one should keep track of the history of $\boldsymbol{x}_t$ in order to approximate the occupation time with a constant memory cost. All the hyperparameter settings used in the experiments and their tuning process are reported in Appendix 3.E. The small-scale problems are run on a commodity machine with Intel® Core™ i7-7500U CPU. The image classification tasks are run on Google Colab, utilizing NVIDIA A100 GPU in the High-RAM setting.

**Stair Function Problem.** Given $N \in \mathbb{Z}^+, L \in \mathbb{R}^+$, define a function $\tilde{f} : \mathbb{R}^+ \to \mathbb{R}^+$ as

$$\tilde{f}(r) = \begin{cases} r^3, & r \in [0, \frac{1}{2}L), \\ (r - nL)^3 + \frac{1}{4}nL^3, & r \in [a(n), b(n)), 1 \le n \le N, \\ (r - NL)^3 + \frac{1}{4}NL^3, & r \in [NL + \frac{1}{2}L, \infty), \end{cases}$$

where $a(n) = nL - \frac{1}{2}L$ and $b(n) = nL + \frac{1}{2}L$. For $\boldsymbol{x} = (x_1, \dots, x_d) \in \mathbb{R}^d$, we define $f(\boldsymbol{x}) = \tilde{f}\left(\frac{1}{d}\sum_{i=1}^d x_i^2\right)$.

Figure 3.3.1 presents the visualization of the case $N = 4, L = 1$ as well as the training curves of $f$ given by 5 different algorithms when $d = 4$. The initial values are all the same, and all the algorithms except for GD are run 3 times considering the randomness of perturbations. We observe that while GD becomes trapped at saddle points, all other algorithms successfully navigate away from these points. Furthermore, the newly introduced algorithms, PGDOT and PAGDOT, outperform their respective counterparts.

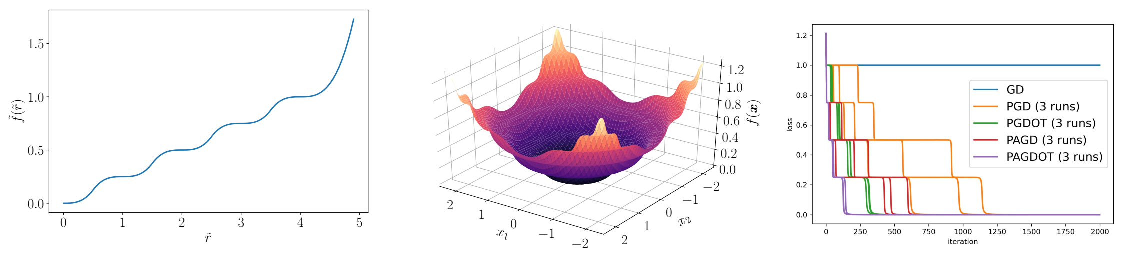


Figure 3.3.1: Graph of $\tilde{f}$ (left) and the landscape of $f(\boldsymbol{x})$ (middle) with $\boldsymbol{x} \in \mathbb{R}^2$ in the case of $N = 4, L = 1$. The figure on the right shows the performance of different algorithms in the stair function problem when $N = 4, L = 1, d = 4$. In this figure, the y-axis and x-axis represent the loss and number of iterations, respectively.

**Nonlinear Regression Problem.** We consider a nonlinear regression problem, adapted from learning time series data with a continuous dynamical system [89]. The loss function is defined as $f(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^N(\hat{y}(s_i;\boldsymbol{x}) - y^*(s_i))^2$, where $\{s_i\}_{i=1}^N$ are $N$ sample points, $y^*(s)$ is the target function, and $\hat{y}(s)$ is the function to fit with the form $\hat{y}(s;\boldsymbol{x}) = \sum_{m=1}^M(a_m\cos(\lambda_m s) + b_m\sin(\lambda_m s))e^{w_m s}$. Here $\boldsymbol{x} = \{a_m, b_m, \lambda_m, w_m\}_{m=1}^M$ and the optimization problem is non-convex. We assume $y^*(s) = \mathrm{Ai}(\omega[s - s_0])$, where $\omega = 3.2, s_0 = 3.0$, and $\mathrm{Ai}(s)$ is the Airy function of the first kind, given by the improper integral $\mathrm{Ai}(s) = \frac{1}{\pi}\int_0^\infty \cos\left(\frac{u^3}{3} + su\right)du$.

For the specific regression model, we assume $M = 4$ and use $N = 50$ data points with $s_i = i/10, i = 0, \dots, 49$. $\{a_m, b_m, \lambda_m\}_{m=1}^4$ are initialized via $\mathcal{N}(0,1)$ and $\{w_m\}_{m=1}^4$ are initialized via $\mathrm{Unif}(-2, -0.2)$. Figure 3.3.2 shows the target function and the fitted function obtained by PGDOT. Also, the learning curves of 5 different algorithms are plotted. We observe that PGDOT and PAGDOT escape the saddle point faster than GD and outperform PGD and PAGD, respectively.

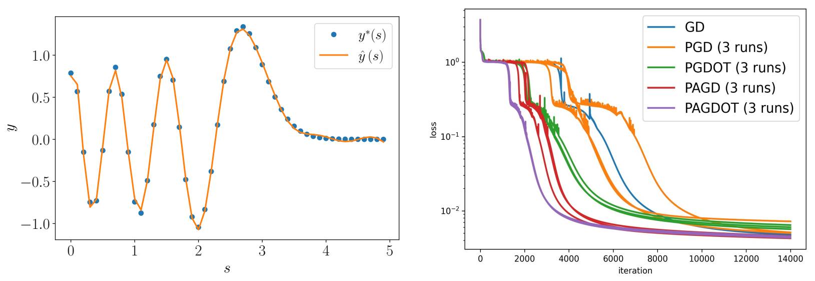


Figure 3.3.2: The target function $y^*(t)$ and fitted function $\hat{y}(t)$ obtained by PGDOT (left), and the performance of different algorithms (right) in the nonlinear regression problem. In the figure on the right, the y-axis and x-axis represent the loss and number of iterations, respectively.

The next two non-convex optimization problems are taken from [143]. In both problems, all other algorithms escape saddle points faster than GD, with PGDOT and PAGDOT slightly outperforming their counterparts.

**Regularized Quadratic Problem.** The first problem is a regularized quadratic problem [118], in which the loss function is defined as

$$f_1(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T H\boldsymbol{x} + \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{b}_i^T\boldsymbol{x} + \|\boldsymbol{x}\|_{10}^{10},$$

where we take $N = 10$, $H = \text{diag}([1, -0.1])$, and $\boldsymbol{b}_i$'s instances of $\mathcal{N}(0, \text{diag}([0.1, 0.001]))$. We initialize this problem at $\boldsymbol{x}_0 = 0$. Different algorithms' performance are shown in Figure 3.3.3.

**Phase Retrieval Problem.** The second problem is the phase retrieval problem [21] with loss function

$$f_2(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N}((\boldsymbol{a}_i^T\boldsymbol{x})^2 - (\boldsymbol{a}_i^T\boldsymbol{x}^*)^2)^2,$$

where we choose $N = 200$, $\boldsymbol{x}^*$ an instance of $\mathcal{N}(0, I_d/d)$, and $\boldsymbol{a}_i$'s instances of $\mathcal{N}(0, I_d)$ with $d = 10$. We initialize the problem at $\boldsymbol{x}_0$ sampled from $\mathcal{N}(0, I_d/(10000d))$. Figure 3.3.3 presents the learning curves of different algorithms.

**Image Classification Task.** [34] observed that in training multilayer perceptrons (MLPs) on MNIST and CIFAR-10, SGD might get stuck at saddle points. Additionally, [129] demonstrated that Adam gets stuck and performs poorly when MLPs with certain weight initialization are trained on the MNIST dataset. In the following, we conduct image classification task on the MNIST, CIFAR-10, and CIFAR-100 datasets using various deep learning models.

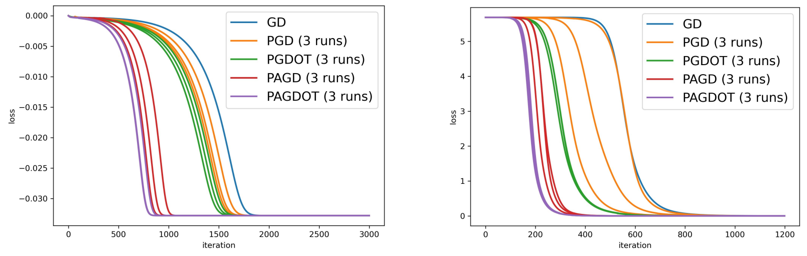


Figure 3.3.3: The performance of different algorithms in the regularized quadratic problem (left) and the phase retrieval problem (right). In both figures, the y-axis and x-axis represent the loss and number of iterations, respectively.

The purpose of these experiments is to deliberately subject the algorithms to saddle points through specific weight initialization schemes, drawing inspiration from the abovementioned works.

In each experiment, we compare our newly introduced algorithms with PGD, PAGD, SGD, Adam, AMSGrad, AdaBlief, and STORM. The batch size is set as 128 for all algorithms across all experiments. It is worth mentioning that SGD and the adaptive gradient methods exhibit poor performance across all experiments, leading to nearly identical plots. As a result, we only present the training curves of SGD and Adam in this section, while the performance results of AMSGrad, AdaBelief, and STORM can be found in Appendix 3.F.

In the first experiment, we train an MLP with one hidden layer of size 100 on the MNIST dataset. We use Rectified Linear Unit (ReLu) as the activation function of the neurons in the hidden layer. The MLP has 3 layers of size (28×28)-100-10 totalling in 79,510 parameters considering all the weights and biases. We initialize the weights and biases with $\mathcal{N}(-1, 0.1)$ and run the algorithms for 100 epochs. Figure 3.3.4 shows the train and test losses and accuracy results of different algorithms.

In the second experiment, we train SqueezeNet [64] on the CIFAR-10 dataset. We set the number of output channels in the last convolutional layer as 10 to account for the 10 different classes existing in CIFAR-10. In total, the model has 740,554 parameters. The weights and biases of the convolutional layers are initialized with $\mathcal{N}(-1, 0.1)$ and the algorithms are run for 200 epochs. Figure 3.3.5 shows the train and test losses and accuracy results of different algorithms.

In the third experiment, we train ResNet18 [57] on the CIFAR-100 dataset. We set the number of output channels in the last linear layer as 100 to account for the 100 different classes existing CIFAR-100. In total, the model has 11,227,812 parameters. This time, we initialize the weights and biases of the batch normalization modules (as denoted by $\gamma$ and $\beta$ in [66]) to 0 and run the algorithms for 50 epochs. Figure 3.3.6 shows the train and test losses and accuracy results of different algorithms.

From the results obtained, it is evident that both SGD and the adaptive gradient methods

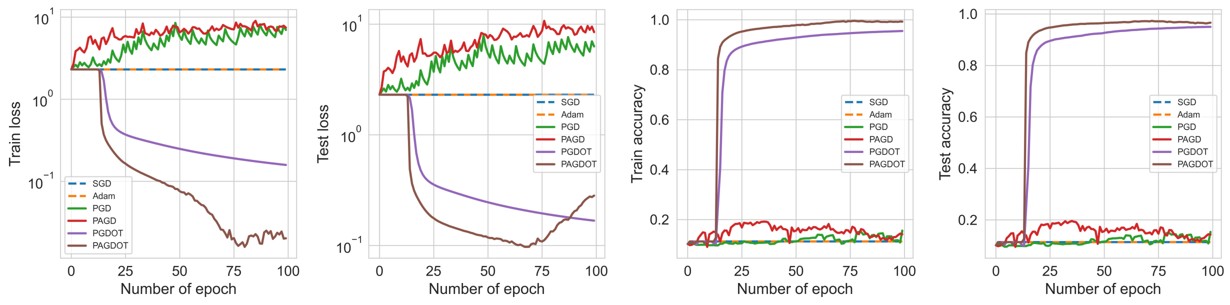


Figure 3.3.4: Experimental results of training the MLP model on MNIST using various algorithms. The dashed blue line corresponds to SGD; the dashed orange line corresponds to Adam; the solid green line corresponds to PGD; the solid red line corresponds to PAGD, the solid purple line corresponds to PGDOT; and the solid brown line corresponds to PAGDOT.

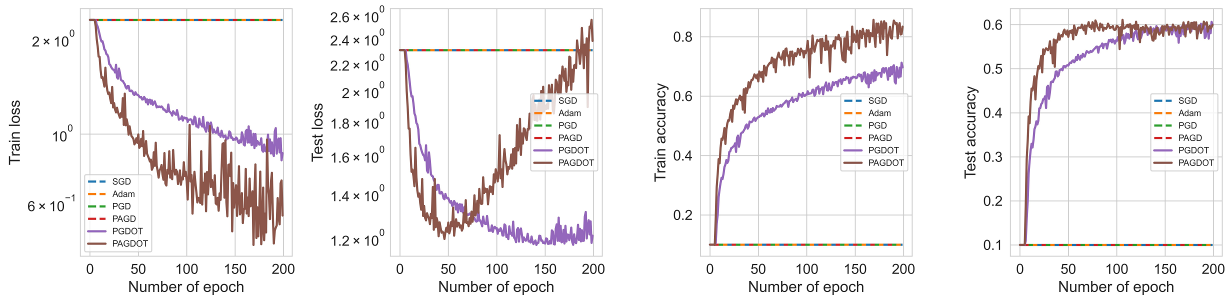


Figure 3.3.5: Experimental results of training SqueezeNet on CIFAR-10 using various algorithms. The dashed blue line corresponds to SGD; the dashed orange line corresponds to Adam; the dashed green line corresponds to PGD; the dashed red line corresponds to PAGD, the solid purple line corresponds to PGDOT; and the solid brown line corresponds to PAGDOT.

exhibit poor performance across all experiments. Despite the presence of inherent noise in the mini-batch gradient, these methods fail to navigate away from saddle points effectively. Conversely, the new algorithms demonstrate the ability to successfully escape saddle points and achieve significant reductions in loss. Moreover, when comparing the new algorithms with their counterparts, it becomes apparent that the newly introduced perturbation mechanism holds a distinct advantage over the uniform perturbation utilized in PGD and PAGD. It is important to highlight that the observed generalization gap in CIFAR-10 and CIFAR-100 can be mainly attributed to the specific initialization scheme employed.

**MFG Example.** We consider the problem of solving the Susceptible Infected environment [32] using the MFOMO algorithm. [50] For details on the terminology used, please refer to

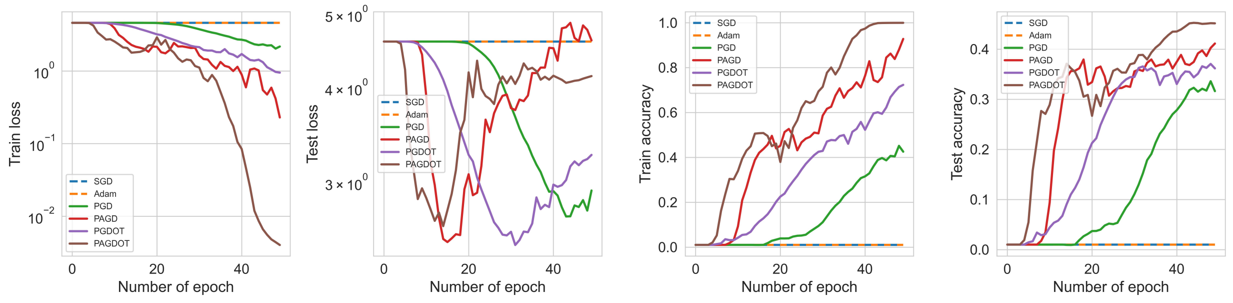


Figure 3.3.6: Experimental results of training ResNet18 on CIFAR-100 using various algorithms. The dashed blue line corresponds to SGD; the dashed orange line corresponds to Adam; the solid green line corresponds to PGD; the solid red line corresponds to PAGD, the solid purple line corresponds to PGDOT; and the solid brown line corresponds to PAGDOT.

appendices 2.A and 2.B in the previous chapter.

In this environment, a large number of agents can choose between social distancing (D) or going out (U). If a susceptible (S) agent chooses social distancing, they may not become infected (I). Otherwise, an agent may become infected with a probability proportional to the number of agents being infected. If infected, an agent will recover with a fixed chance every time step. Both social distancing and being infected have an associated cost. Let $\mathcal{S} = \{S, I\}$, $\mathcal{A} = \{U, D\}$, $\mu_0(I) = 0.6$, $r(s, a, \mu_t) = -\mathbf{1}_{\{I\}}(s) - 0.5 \;.\; \mathbf{1}_{\{D\}}(a)$, and $\mathcal{T} = \{0, 1\}$. The transition probabilities are given as

$$P(s_{t+1} = S|s_t = I) = 0.3,$$
$$P(s_{t+1} = I|s_t = S, a_t = U) = 0.9^2\mu_t(I),$$
$$P(s_{t+1} = I|s_t = S, a_t = D) = 0.$$

Recall that MFOMO reformulates the problem of finding NE solutions as a constrained optimization problem represented by equation (2.11). While this formulation restricts us to using constrained optimization algorithms, the simplicity of the constraints allows for parameterizing the variables. Consequently, an equivalent unconstrained optimization problem is formed, granting flexibility in selecting the optimizer. Initially, the last two constraints can be relaxed since they primarily serve to bound the constraints. Subsequently, $L$ and $z$ can be parameterized as follows:

$$L_{s,a,t} = \frac{\exp(u_{s,a,t})}{\sum_{s',a'} \exp(u_{s',a',t})}, \quad z_{s,a,t} = q_{s,a,t}^2,$$

for some $u = [u_{s,a,t}]_{s\in\mathcal{S},a\in\mathcal{A},t\in\mathcal{T}}$ and $q = [q_{s,a,t}]_{s\in\mathcal{S},a\in\mathcal{A},t\in\mathcal{T}}$ in $\mathbb{R}^{SA(T+1)}$. The equivalent unconstrained formulation is as follows:

$$\min_{u,q,y} \quad \|A_{\sigma(u)}\sigma(u) - b\|_2^2 + \|A_{\sigma(u)}^T y + \mathrm{diag}(q)q - c_{\sigma(u)}\|_2^2 + q^T\mathrm{diag}(q)\sigma(u), \tag{3.4}$$

where $\sigma(u)_{s,a,t} = \frac{\exp(u_{s,a,t})}{\sum_{s',a'} \exp(u_{s',a',t})}$, for $s \in \mathcal{S}, a \in \mathcal{A}, t \in \mathcal{T}$.

In general, equation (3.4) represents a non-convex optimization problem. As discussed earlier, the presence of saddle points poses a significant challenge for various optimization algorithms in non-convex settings. To demonstrate the existence of saddle points in the Susceptible Infected problem, we examine the objective function of (3.4) in the vicinity of MFOMO's initial point—we initialize the elements of $u$ with $\log(1/4)$ to ensure a uniform mean-field $L$, while setting $q$ and $y$ to zero.

Due to the high dimensionality of the problem (20 dimensions involving $u$ and $q$ in $\mathbb{R}^8$ and $y$ in $\mathbb{R}^4$), directly visualizing the objective function is not feasible. Instead, we generate a plot showing the norm of the gradient versus the minimum eigenvalue of the Hessian matrix at randomly selected points in the vicinity of MFOMO's initial point. See Figure 3.3.7. The depicted figure provides evidence supporting the presence of a saddle point near the initial point. This conclusion is drawn from the observation of multiple instances where the norm of the gradient is significantly small while the magnitude of the minimum eigenvalue of the Hessian matrix is comparatively large.

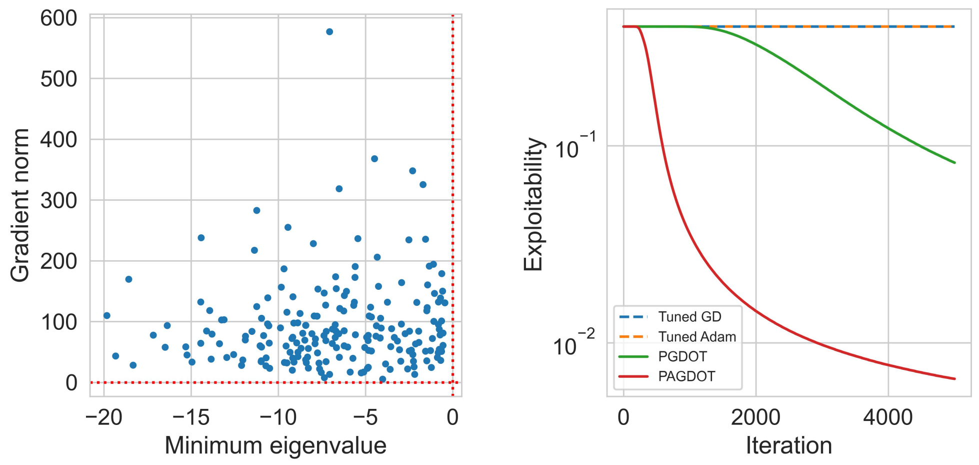


Figure 3.3.7: Scatter plot of gradient norm vs. minimum eigenvalue of Hessian in the vicinity of MFOMO's initial point (left) and Training curves of MFOMO equipped with different optimizers (right) for the Susceptible Infected environment.

Considering the existence of saddle points, it is anticipated that MFOMO equipped with GD or adaptive gradient methods such as Adam may become trapped and fail to converge to an NE solution. The training curves of MFOMO with four different optimizers, namely GD, Adam, PGDOT, and PAGDOT, are illustrated in Figure 3.3.7. It is worth noting that GD and Adam struggle to escape the saddle point and reduce the exploitability score, while both PGDOT and PAGDOT successfully do so. GD and Adam are tuned using a grid search, and their learning rate is set to 0.01 and 0.1, respectively. In PGDOT, learning rate is 0.01,

$g_{\text{thresh}} = 0.1$, $t_{\text{thresh}} = 100$, $t_{\text{count}} = 100$, $h = 10^{12}$, $\alpha = 5$, and $r = 0.01$. PAGDOT has similar parameters with momentum being 0.9.

## 3.4 Conclusion

In this chapter, we developed a new perturbation mechanism in which the perturbations are adapted to the history of states via the notion of occupation time. This mechanism is integrated into the framework of PGD and PAGD resulting in two new algorithms: PGDOT and PAGDOT. We prove that PGDOT and PAGDOT converge rapidly to second-order stationary points, which is corroborated by empirical studies ranging from time series analysis and the phase retrieval problem to deep learning.

# Appendix

## 3.A Monotone Convergence of Gradient Descent

Here we prove a property of gradient descent applied to a function $f : \mathbb{R} \to \mathbb{R}$, as mentioned in the introduction. This property of gradient descent supports the use of our proposed perturbation mechanism.

**Proposition 2.** *Let $f \in \mathcal{C}^2(\mathbb{R})$. Assume that we start gradient descent at some arbitrary point $x_0$, and the corresponding iterates $\{x_n\}_{n\geq 0}$ converge to the point $x_s$ with $f''(x_s) \neq 0$. Then, if $f$ is $\ell$-gradient Lipschitz and the step size is less than $\frac{1}{\ell}$, the sequence $\{x_n\}_{n\geq 0}$ converges monotonically to $x_s$.*

In order to prove this proposition, we break it down into two lemmas.

**Lemma 1.** *Let $f \in \mathcal{C}^2(\mathbb{R})$. Assume that we start gradient descent at some arbitrary point $x_0$, and the corresponding iterates $\{x_n\}_{n\geq 0}$ converge to the point $x_s$ with $f''(x_s) \neq 0$. Then, if $f$ is $\ell$-gradient Lipschitz and the step size is less than $\frac{1}{\ell}$, there exists $M > 0$ such that the sequence $\{x_n\}_{n\geq M}$ converges monotonically to $x_s$.*

*Proof.* Note that for $n \geq 0$, $x_{n+1} = x_n - \eta f'(x_n)$, where $0 < \eta < \frac{1}{\ell}$ is the step size. Also, it is easy to show that $f'(x_s) = 0$. Assume that at some point $x_n \geq x_s$. Then, since $|f'(x_n) - f'(x_s)| = |f'(x_n)| \leq \ell |x_n - x_s|$, we have

$$\begin{aligned} x_s \leq x_n - \frac{1}{\ell}|f'(x_n)| &\leq x_n - \eta |f'(x_n)| \\ &\leq x_n - \eta f'(x_n) = x_{n+1}. \end{aligned}$$

Similarly, if $x_n \leq x_s$, then we get $x_{n+1} \leq x_s$. This implies that the sequence $\{x_n\}_{n\geq 0}$ is entirely either on the left hand side of $x_s$ or on its right hand side (including $x_s$).

Without loss of generality, assume that the entire sequence of iterations lies on the right hand side of $x_s$. If at some iteration, $x_m = x_s$, then since $f'(x_s) = 0$, $x_n = x_s$ for $n \geq m$, which yields the desired result. So we can assume that $x_n \neq x_s$ for all $n \geq 0$. Using a similar argument, we can also assume that $f'(x_n) \neq 0$ for all $n \geq 0$. Suppose by contradiction that there is no such $M$ as described in the lemma. Then there exist infinitely many $n$ such that $x_n < x_{n+1}$ implying that for infinitely many $n$, $f'(x_n) < 0$. Since $\lim_{n\to\infty} x_n = x_s$ and the entire sequence is on the right hands side of $x_s$, we also have infinitely many $n$ such that $f'(x_n) > 0$. Combining these results, one can construct a strictly decreasing sub-sequence $\{y_n\}_{n\geq 0}$ of the iterations such that $\lim_{n\to\infty} y_n = x_s$, $f'(y_{2m}) > 0$, and $f'(y_{2m+1}) < 0$ for all $m \geq 0$. Since $f'$ is continuous, there exists $y_{2m+1} < z_m < y_{2m}$ such that $f'(z_m) = 0$, for each $m \geq 0$. It is easy to see that $\{z_n\}_{n\geq 0}$ is also strictly decreasing and $\lim_{n\to\infty} z_n = x_s$. Note that since $f''$ is continuous, by the mean value theorem, one can find a sequence $\{t_n\}_{n\geq 0}$ such that for each $n \geq 0$, $z_{n+1} < t_n < z_n$ and $f''(t_n) = 0$. Since $\{z_n\}_{n\geq 0}$ converges to $x_s$, then so does $\{t_n\}_{n\geq 0}$. But this implies that $f''(x_s) = \lim_{n\to\infty} f''(t_n) = 0$ contradicting with the fact that $f''(x_s) \neq 0$. ☐

**Lemma 2.** *Given the setting in Lemma 1, $\{x_n\}_{n\geq 0}$ converges monotonically to $x_s$.*

*Proof.* Without loss of generality, assume that $x_0 \geq x_s$, then using what we obtained during the proof of Lemma 1, we know that the entire sequence $\{x_n\}_{n\geq 0}$ lies on the right hand side of $x_s$ ((including $x_s$). Let $M$ be the minimum index that satisfies the condition in Lemma 1. Suppose by contradiction that $M > 0$. So $x_s < x_{M-1} < x_M$, which implies $f'(x_{M-1}) < 0$ considering $x_M = x_{M-1} - \eta f'(x_{M-1})$. Since the sequence converges to $x_s$, there should be a $k \geq 0$ such that $x_{M+k+1} < x_{M-1} < x_{M+k}$. Note that $x_{M+k+1} = x_{M+k} - \eta f'(x_{M+k})$, so

$$\eta f'(x_{M+k}) = x_{M+k} - x_{M+k+1} > x_{M+k} - x_{M-1}.$$

Since $f'(x_{M-1}) < 0$, we have $\eta\big(f'(x_{M+k}) - f'(x_{M-1})\big) > \eta f'(x_{M+k}) > x_{M+k} - x_{M-1}$. This contradicts the fact that $\eta(f'(x_{M+k}) - f'(x_{M-1})) \leq \eta\ell(x_{M+k} - x_{M-1}) < x_{M+k} - x_{M-1}$. ☐

## 3.B Background on Convex Optimization

We provide some context of gradient descent applied to convex functions.

**Definition 5.**

1. *A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz if $\|\nabla f(\boldsymbol{x}_1) - \nabla f(\boldsymbol{x}_2)\| \leq \ell\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ for all $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$.*
2. *A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is $\alpha$-strongly convex if $\lambda_{\min}(\nabla^2 f(\boldsymbol{x})) \geq \alpha$ for all $\boldsymbol{x} \in \mathbb{R}^d$.*

The gradient Lipschitz condition controls the amount of decay in each iteration, and the strong convexity condition guarantees that the unique stationary point is the global minimum. The ratio $\ell/\alpha$ is often called the condition number of the function $f$. The following theorem shows the linear convergence of gradient descent to the global minimum $\boldsymbol{x}^\star$, see [18][Theorem 3.10] and [104][Theorem 2.1.15].

**Theorem 7.** *[18, 104] Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz and $\alpha$-strongly convex. For any $\epsilon > 0$, if we run gradient descent with step size $\eta = \ell^{-1}$, then the number of iterations to be $\epsilon$-close to $\boldsymbol{x}^\star$ is $\frac{2\ell}{\alpha}\log\left(\frac{\|\boldsymbol{x}_0 - \boldsymbol{x}^\star\|}{\epsilon}\right)$.*

## 3.C Proof of Theorem 4

Suppose by contradiction that with positive probability, the walk is localized at some points $\{k, \dots, \ell\}$. We focus on the left end $k$. Let $\tau_n^k$ be the time at which the point $k$ is visited $n$ times. For $n$ sufficiently large, the point $k+1$ is visited approximately at least $n$ times by $\tau_n^k$. So at time $\tau_n^k$, the walk moves from $k$ to $k+1$ with probability bounded from above by $C/w(n)$ for some constant $C > 0$. Consequently, the probability that the walk is localized at $\{k, \dots, \ell\}$ is less than $\prod_{n>0} \frac{C}{w(n)}$. By standard analysis, $\prod_{n>0} \frac{C}{w(n)} = 0$ if $w(n) \to \infty$ as $n \to \infty$. This leads to the desired result.

## 3.D Proof of Theorem 5

We show how the thin-pancake property of saddle points is used to prove Theorem 5. Recall that an $\epsilon$-second-order stationary point is a point with a small gradient, and where the Hessian does not have a large negative eigenvalue. Let us put down the basic idea in Section 3.1 with the parameters in Algorithm 3 (PGDOT). If we are currently at an iterate $\boldsymbol{x}_t$ which is not an $\epsilon$-second-order stationary point, there are two cases: (1) The gradient is large: $\|\nabla f(\boldsymbol{x_t})\| \geq g_{\text{thres}}$; (2) $\boldsymbol{x}_t$ is close to a saddle point: $\|\nabla f(\boldsymbol{x_t})\| \leq g_{\text{thres}}$ and $\lambda_{\min}(\nabla^2 f(\boldsymbol{x_t})) \leq -\sqrt{\rho\epsilon}$. The case (1) is easy to deal with by the following elementary lemma.

**Lemma 3.** *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz. Then for GD with step size $\eta < \ell^{-1}$, we have $f(\boldsymbol{x}_{t+1}) - f(\boldsymbol{x}_t) \leq -\frac{\eta}{2}\|\nabla f(\boldsymbol{x}_t)\|^2$.*

The case (2) is more subtle, and the following lemma gives the decay of the function value after a random perturbation described in Algorithm 3 (PGDOT).

**Lemma 4.** *Assume that $f : \mathbb{R}^d \to \mathbb{R}$ is $\ell$-gradient Lipschitz and $\rho$-Hessian Lipschitz. If $\|\nabla f(\boldsymbol{x_t})\| \leq g_{thres}$ and $\lambda_{\min}(\nabla^2 f(\boldsymbol{x_t})) \leq -\sqrt{\rho\epsilon}$, then adding one perturbation step as in Algorithm 3 followed by $t_{thres}$ steps of GD with step size $\eta$, we have $f(\boldsymbol{x}_{t+t_{thres}}) - f(\boldsymbol{x}_t) \leq -f_{thres}$ with probability at least $1 - \frac{d\ell}{\sqrt{\rho\epsilon}} e^{-\chi}$.*

[70] proved Lemma 4 for PGD, and used it together with Lemma 3 to prove Theorem 2. We will use the same argument, with Lemmas 3 and 4, leading to Theorem 5 for PGDOT.

Now, let us explain how to prove Lemma 4 via a purely geometric property of saddle points. Consider a point $\widetilde{\boldsymbol{x}}$ satisfying the condition $\|\nabla f(\widetilde{\boldsymbol{x}})\| \leq g_{\text{thres}}$ and $\lambda_{\min}(\nabla^2 f(\widetilde{\boldsymbol{x}})) \leq -\sqrt{\rho\epsilon}$. After adding the perturbation in Algorithm 3, the resulting vector can be viewed as a distribution over the cube $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$. Similar as in [70], we call $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$ the perturbation cube which is divided into two regions: (1) escape region $\chi_{\text{escape}}$ which consists of all points $\boldsymbol{x} \in C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$ whose function value decreases by at least $f_{\text{thres}}$ after $t_{\text{thres}}$ steps; (2) stuck region $\chi_{\text{stuck}}$ which is the complement of $\chi_{\text{escape}}$ in $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$. The key idea is that the stuck region $\chi_{\text{stuck}}$ looks like a non-flat thin pancake, which has a very small volume compared to that of $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$. This claim can be formalized by the following lemma, which is a direct corollary of [70][Lemma 11] as $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \subseteq B^d(\widetilde{\boldsymbol{x}}, r)$:

**Lemma 5.** *Assume that $\widetilde{\boldsymbol{x}}$ satisfies $\|\nabla f(\widetilde{\boldsymbol{x}})\| \le g_{thres}$ and $\lambda_{\min}(\nabla^2 f(\widetilde{\boldsymbol{x}})) \le -\sqrt{\rho\epsilon}$. Let $\boldsymbol{e}_1$ be the smallest eigendirction of $\nabla^2 f(\widetilde{\boldsymbol{x}})$. For any $\delta < 1/3$ and any $\boldsymbol{u}, \boldsymbol{v} \in C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d})$, if $\boldsymbol{u} - \boldsymbol{v} = \mu r \boldsymbol{e}_1$ and $\mu \ge \delta/(2\sqrt{d})$, then at least one of $\boldsymbol{u}$ and $\boldsymbol{v}$ is not in the stuck region $\chi_{stuck}$.*

To prove Lemma 4, it suffices to check that $\mathbb{P}(\chi_{\text{stuck}}) \le C\delta$ for some $C > 0$. This criterion is general for any (random) perturbation. Let $\mathcal{O}_1, \dots, \mathcal{O}_{2^d}$ be the orthants centered at $\widetilde{\boldsymbol{x}}$; that is, the space $\mathbb{R}^d$ is divided into $2^d$ subspaces according to the coordinate signs of $\cdot - \widetilde{\boldsymbol{x}}$. The symbol $\operatorname{sgn}(\mathcal{O}_i) \in \{-1, 1\}^d$ denotes the coordinate signs of $\boldsymbol{y} - \widetilde{\boldsymbol{x}}$ for any $\boldsymbol{y} \in \mathcal{O}_i$. For $1 \le i \le 2^d$, let

$$p_i = \prod_{\operatorname{sgn}(\mathcal{O}_i)_k = -1} \frac{w(R_t^k)}{w(L_t^k) + w(R_t^k)} \prod_{\operatorname{sgn}(\mathcal{O}_i)_k = 1} \frac{w(L_t^k)}{w(L_t^k) + w(R_t^k)}$$

be the probability that the random perturbation drives $\widetilde{\boldsymbol{x}}$ into $C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \cap \mathcal{O}_i$. Consequently, $\mathbb{P}(\chi_{\text{stuck}}) = \sum_{i=1}^{2^d} p_i \frac{\text{Vol}(\chi_{\text{stuck}} \cap \mathcal{O}_i)}{\text{Vol}(C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \cap \mathcal{O}_i)}$, where $\text{Vol}(\cdot)$ denotes the volume of a domain. It is easy to see that $\text{Vol}(C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \cap \mathcal{O}_i) = (r/\sqrt{d})^d$. By Lemma 5 and the slicing volume bound [6], $\text{Vol}(\chi_{\text{stuck}} \cap \mathcal{O}_i) \le \sqrt{2}(r/\sqrt{d})^{d-1} \frac{\delta r}{\sqrt{d}}$. Therefore, $\frac{\text{Vol}(\chi_{\text{stuck}} \cap \mathcal{O}_i)}{\text{Vol}(C^{(d)}(\widetilde{\boldsymbol{x}}, r/\sqrt{d}) \cap \mathcal{O}_i)} \le \sqrt{2}\delta$ implying that $\mathbb{P}(\chi_{\text{stuck}}) \le \sqrt{2}\delta$.

Note that this proof does not rely on the full history of states for $L_t$ and $R_t$. Thus, one can restrict the number of previous iterations as is done in Section 3.3 using the hyperparameter $t_{\text{count}}$.

## 3.E Hyperparameter Settings in the Numerical Examples

To ensure a fair comparison between algorithms in the small-scale problems, we opted to select a consistent set of hyperparameters. Specifically, we employed the same learning rate as GD for all other algorithms. Additionally, we set the common hyperparameters between the new algorithms and their counterparts to be equal. Table 3.E.1 provides an overview of the hyperparameters utilized in the small-scale problems.

For the image classification tasks (the large-scale problems), we do a grid search and report the best hyperparameters for SGD and the adaptive gradient methods. For SGD, Adam, AMSGrad, and AdaBelief, the only hyperparameter is the learning rate. For STORM, we tune both $k$ and $c$ while setting $w = 0.1$. Table 3.E.2 provides an overview of the hyperparameters utilized by SGD and the adaptive gradient methods in the image classification tasks. It is important to note that the significant disparity observed in the learning rates employed by various algorithms in the second image classification task stems from the fact that regardless of the learning rate value, none of the algorithms manage to escape the saddle points, leading to no improvement in the training loss.

To ensure fair comparison between PGDOT and PAGDOT and their counterparts, we select a consistent set of hyperparameters for them. The only difference is in the perturbation

Table 3.E.1: Hyperparameters utilized in the small-scale problems.

| | d | $h$ | $t_{\text{count}}$ | $\eta$ | $t_{\text{thres}}$ | $g_{\text{thres}}$ | $r$ | momentum | steps |
|---|---|---|---|---|---|---|---|---|---|
| Stair function | 4 | 0.04 | 200 | 0.1 | 10 | 0.01 | 0.04 | 0.5 | 2000 |
| Nonlinear regression | 16 | 0.04 | 200 | 0.1 | 50 | 0.1 | 0.1 | 0.5 | 14000 |
| Regularized quadratic | 2 | 1 | 200 | 0.01 | 50 | 0.01 | 0.01 | 0.5 | 3000 |
| Phase retrieval | 10 | 1 | 200 | 0.001 | 50 | 1 | 0.01 | 0.5 | 1200 |

Table 3.E.2: Hyperparameters utilized by SGD and the adaptive gradient methods in the image classification tasks.

| | SGD | Adam | AMSGrad | AdaBelief | STORM |
|---|---|---|---|---|---|
| MLP on MNIST | lr: 0.01 | lr: 0.001 | lr: 0.001 | lr: 0.001 | k: 0.01, c: 100 |
| SqueezeNet on CIFAR-10 | lr: 1.0 | lr: 0.0001 | lr: 1.0 | lr: 0.1 | k: 0.01, c: 10 |
| ResNet18 on CIFAR-100 | lr: 0.1 | lr: 0.001 | lr: 0.001 | lr: 0.001 | k: 0.1, c: 100 |

norm $r$. Note that norm of the actual perturbation applied in PGDOT and PAGDOT is $r/\sqrt{d}$. Since $d$ is a large number in the large-scale problems, $r/\sqrt{d}$ would be significantly different than $r$. Therefore, we adjust the perturbation norm in PGD and PAGD accordingly. Additionally, in all the image classification tasks, we set $t_{\text{count}} = 100$ and $h = 10^{12}$. Table 3.E.3 provides an overview of the hyperparameters utilized by PGD, PAGD, PGDOT, and PAGDOT.

Table 3.E.3: Hyperparameters utilized by PGD, PAGD, PGDOT, and PAGDOT in the image classification tasks.

| | $\eta$ | $t_{\text{thres}}$ | $g_{\text{thres}}$ | $r_{\text{PGD/PAGD}}$ | $r_{\text{PGDOT/PAGDOT}}$ | momentum |
|---|---|---|---|---|---|---|
| MLP on MNIST | 0.01 | 10 | 0.1 | $\frac{1}{\sqrt{79{,}510}}$ | 1 | 0.9 |
| SqueezeNet on CIFAR-10 | 0.001 | 10 | 1 | $\frac{10}{\sqrt{740{,}554}}$ | 10 | 0.1 |
| ResNet18 on CIFAR-100 | 0.01 | 200 | 1 | $\frac{1}{\sqrt{11{,}227{,}812}}$ | 1 | 0.9 |

We also remark that the weight function in Algorithms 3 and 4 is set as $w(k) = 1 + k^5$ for all the small-scale and large-scale problems.

## 3.F Image Classification Task Results of Adaptive Gradient Methods

Here, we present the training outcomes of AMSGrad, AdaBelief, and STORM, in addition to Adam, across the three distinct image classification tasks. It is worth noting that as Adam exhibited unsatisfactory performance in all experiments, any training curves that closely resemble or are identical to Adam's indicate poor performance for the other algorithms as well.

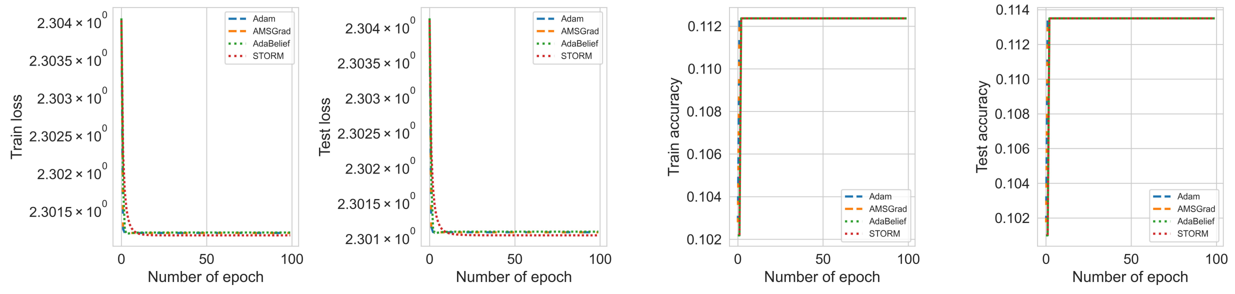


Figure 3.F.1: Experimental results of training the MLP model on MNIST using several adaptive gradient methods. The dashed blue line corresponds to Adam; the dashed orange line corresponds to AMSGrad; the dotted green line corresponds to AdaBelief; and the dotted red line corresponds to STORM.

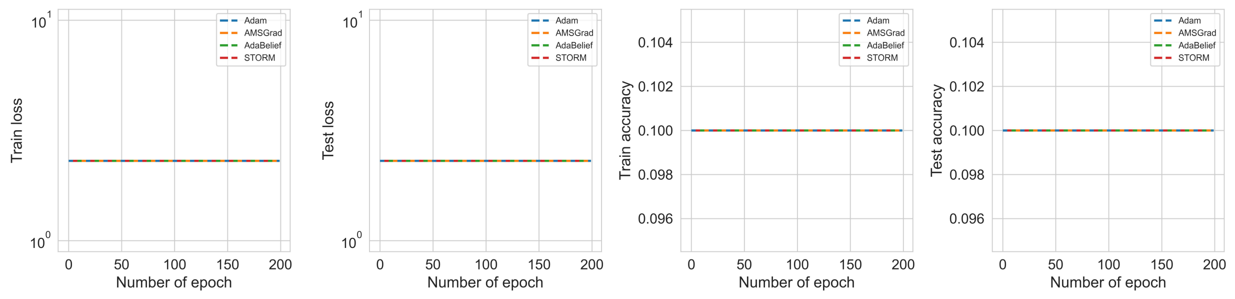


Figure 3.F.2: Experimental results of training SqueezeNet on CIFAR-10 using several adaptive gradient methods. The dashed blue line corresponds to Adam; the dashed orange line corresponds to AMSGrad; the dashed green line corresponds to AdaBelief; and the dashed red line corresponds to STORM.

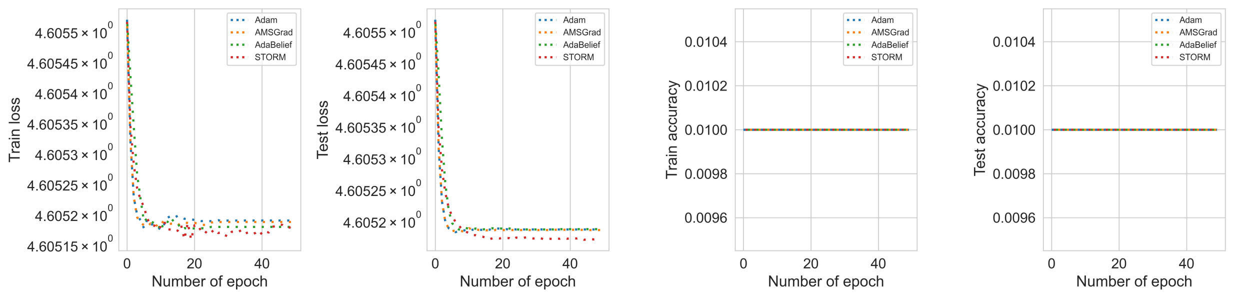


Figure 3.F.3: Experimental results of training ResNet18 on CIFAR-100 using several adaptive gradient methods. The dotted blue line corresponds to Adam; the dotted orange line corresponds to AMSGrad; the dotted green line corresponds to AdaBelief; and the dotted red line corresponds to STORM.

# Chapter 4

# Leveraging Stacked Generalization to Effectively Detect Overutilization in Medicare

It is estimated that over \$100 billion is lost each year in the United States due to healthcare fraud and abuse accounting for 3-10% of all healthcare expenditures.[102, 137] One major target for overutilization and fraud is Medicare, for which utilization data is publicly available. Over 63 million Americans are currently enrolled in Medicare and the total cost of the program in 2021 was \$900.8 billion, accounting for 21% of total national healthcare expenditures.[94] Given a rapidly aging American population, 75 million Americans will be enrolled in Medicare by 2027, and Medicare spending is projected to jump to nearly \$1.6 trillion by 2028.[63, 78] Despite the increase in federal healthcare spending over past decades, patients are increasingly being forced to shoulder more of the financial burden. Between 2002 and 2022, Medicare Part B deductibles have risen 133% and monthly premiums have risen 215%.[31, 93] Yet, even with increased spending from both the federal government and patients, the Medicare trust fund is expected to become insolvent in the near future.[41] The substantial financial losses from fraud and abuse call for new methods of fraud detection, involving both physicians and data scientists.

A small fraction of medical providers are known to have engaged in various forms of overutilization or fraud to exploit the Medicare system. One prevalent form is phantom billing, where unnecessary procedures are performed and billed to Medicare. Another form is upcoding, where providers bill for higher reimbursing procedures while actually performing lower-reimbursing ones. Additionally, cloned documentation occurs when providers copy information from previous patient records or even from different patients, leading to false or inaccurate billing. These practices not only waste healthcare resources but also compromise patient care and can put lives at risk with unnecessary procedures. Currently, there is no reliable method to detect overutilization or fraud in healthcare whereas financial systems (i.e., banking, stock markets, etc.) have far more sophisticated methods for early detection and prevention. Fraud detection within healthcare is primarily done through a manual effort by auditors and investigators searching through billing records in a time-consuming and often imprecise manner. It is often dependent on whistleblowers or insiders to report employers, and therefore the vast majority of overutilization and fraud goes undetected. In 2019, only $2.6 billion of fraudulent claims were recovered by the United States Department of Justice. [74] Given the lack of domain expertise and the subtlety of Medicare abuse by healthcare providers, it can be difficult for auditors to determine what test or procedure is medically necessary in a particular office population.

Detection of fraud using machine learning (ML) has gained significant attention in recent years due to its potential for improving the efficiency and effectiveness of fraud identification and prevention. Since the Centers for Medicare and Medicaid Services (CMS) released public data files in 2014, numerous studies have explored the application of various ML techniques in Medicare fraud detection. For instance, Bauder and Khoshgoftaar [11] conducted a comparative study with supervised, unsupervised, and hybrid ML approaches. They found that the supervised methods tend to perform better than unsupervised or hybrid methods, but the results could vary depending on the class imbalance sampling technique and provider type. Herland et al. [58] focused on the detection of Medicare fraud using various CMS public datasets. They provided detailed discussions on Medicare data processing and exploratory analyses in order to show the best learners and datasets for the detection of fraudulent claims. Several studies had a narrower focus on the performance and applications of supervised ML algorithms such as gradient boosted decision trees (Hancock and Khoshgoftaar [56]), CatBoost (Hancock and Khoshgoftaar [54]), bagging (Yao et al. [149]), and deep learning (Johnson and Khoshgoftaar [73]; Mayaki and Riveill [92]) in identifying Medicare fraud. Additionally, there have been works that deployed unsupervised approaches to identify fraudulent outliers and Medicare anomaly. (Bauder and Khoshgoftaar [10]; Branting et al. [16]; Bauder et al. [9]; Sadiq and Shyu [121])

While ML techniques have shown promise in detecting Medicare fraud, there are several issues that can limit their effectiveness. A common issue with many studies is the heavy reliance on the List of Excluded Individuals and Entities (LEIE) [65] for model training and evaluation. LEIE is a database compiled by the Office of the Inspector General that reports individuals and entities that have been excluded from receiving federally-funded healthcare programs due to fraud. A significant hurdle in using known fraud labels from the LEIE dataset is the issue of class imbalance with a fraud rate between 0.038% and 0.074%. Data-level techniques such as random sampling and varying class distribution are shown to help mitigate the issue of class imbalance. (Brauder and Khoshgoftar [8, 12]; Johnson and Khoshgoftaar [73]; Hancock et al. [55]) Another notable problem associated with using the LEIE dataset is that many of the providers listed in LEIE were prosecuted due to overt and deliberate fraudulent billing—their convictions were a diverse array of felonies such as billing under the National Provider Identifier (NPI) of another doctor, etc. The vast majority of healthcare abuse is attributed to overutilization. The subtle variations in billing patterns, which may show higher utilization of certain procedures and services than medically necessary is often undetectable by the non-physician. Using LEIE providers and their Medicare billing data as a training set may develop a model that can detect brazen, outlandish billing patterns, but would be unable to pinpoint instances of more subtle fraud from which a majority of the financial loss and waste occurs.

Another common issue observed in many Medicare fraud detection studies is the tendency to focus on aggregate information such as the total number of procedures performed, total Medicare reimbursement amount, etc. to detect fraudulent activity. (Herland et al. [58]; Yao et al. [149]; Mayaki and Riveill [92]) While these aggregate features can provide a high-level overview, they may overlook crucial details about the specific procedures and services performed by the providers. By neglecting the granular information associated with each individual procedure, the model may miss out on identifying specific patterns or anomalies that could be indicative of fraudulent behavior. It is worth mentioning that one approach used to get around the data aggregation challenge is to assign overutilization labels to provider-procedure pairs instead of just the providers. This approach reflects the fact that fraudulent providers do not necessarily engage in fraudulent activities for every single procedure they perform. Bauder and Khoshgoftaar [8], Johnson and Khoshgoftaar [72], and Hancock and Khoshgoftaar [54] use the LEIE dataset to label the provider-procedure pairs. The issue with using the LEIE dataset is that it does not contain data on which procedures the provider submitted fraudulent claims for, and therefore all the provider-procedure pairs for the providers listed on the LEIE dataset should be labeled as fraud. This clearly could result in incorrect labels and introducing noise and bias into the training data, which in turn could significantly compromise the performance of the ML models.

In this chapter, our primary goals are twofold. First, we aim to address the limitations of the LEIE dataset by incorporating the experience of seasoned physicians and medical billers to create a labeled dataset that overcomes the issues of class imbalance and the exclusive focus on overtly fraudulent providers. Unlike previous approaches our methodology involves capturing the nuances and intricacies of fraudulent behavior by incorporating the specific details of each procedure and service. By focusing our efforts on the field of ophthalmology, we leverage our access to domain knowledge and concentrate on the unique characteristics and specific Healthcare Common Procedure Coding System (HCPCS) codes relevant to this medical specialty. This targeted approach enables us to build a specialized dataset that reflects the intricacies and complexities of Medicare overutilization within ophthalmology.

Second, we conduct a comparative study of various machine learning models for the task of predicting Medicare overutilization within ophthalmology. To this end, we compare the performance of several supervised ML models including k-nearest neighbor, logistic regression, support vector machines, extreme gradient boosting, multilayer perceptron, as well as an ensemble model based on the Stacked Generalization (stacking) method [145] on the newly created dataset. The comparative study not only enables us to identify the best-performing model but also aims to demonstrate the effectiveness of stacking ensemble model in surpassing individual models in Medicare overutilization detection. Once the best-performing model is determined, we extend our analysis beyond model performance to estimate essential overutilization statistics within the field of ophthalmology. These statistics include the overall overutilization and fraud rate within the specialty and the financial losses incurred as a result of these activities.

## 4.1 Data Source, Preprocessing, and Labeling

### Data Source

The unprocessed data, without labels, is sourced from the CMS website [95], which currently encompasses data for 2013 to 2021 calendar years. Our analysis mainly relies on the Medicare Provider Utilization and Payment Data (MPUPD). [124] MPUPD comprises publicly available data files summarizing the information on services and procedures provided to Medicare beneficiaries by physicians and other healthcare professionals. Within MPUPD, we utilize two specific data files:

- Medicare Physician and Other Practitioners - by Provider and Service [126]: This data, referred to as MPOP_PS, provides detailed information on use, payments, and submitted charges organized by NPI, HCPCS code, and place of service.
- Medicare Physician and Other Practitioners - by Provider [125]: This data, referred to as MPOP_P, provides summary information on use, payments, submitted charges, and beneficiary demographic and health characteristics organized by NPI.

Note that both MPOP_PS and MPOP_P comprise individual datasets corresponding to each calendar year from 2013 to 2021. It is also worth mentioning that the data extraction process is specifically tailored to ophthalmologists, as our study focuses exclusively on this medical specialty. Table 4.1.1 summarizes the number of ophthalmology records found in MPOP_PS and MPOP_P in different calendar years. Appendix A provides further details on the data source.

Table 4.1.1: Number of ophthalmology records

| Year | 2013 | 2014 | 2015 | 2016 | 2017 | 2018 | 2019 | 2020 | 2021 |
|---|---|---|---|---|---|---|---|---|---|
| MPOP_PS | 237,630 | 238,242 | 240,279 | 242,748 | 240,830 | 239,707 | 239,906 | 219,717 | 230,653 |
| MPOP_P | 17,550 | 17,664 | 17,698 | 17,788 | 17,817 | 17,804 | 17,856 | 17,631 | 17,489 |

### Data Preprocessing

In our study, specific data preprocessing steps are taken to transform MPOP_PS and MPOP_P into a structured feature representation that captures the details of procedures and services for each provider in a given calendar year. The flowchart in Figure 4.1.1 illustrates a concise representation of the various stages involved in the data preprocessing step. In particular,

- Pivoting MPOP_PS: We pivot the records based on the providers' NPI to create a feature vector for each provider. This way, for each provider, we have a unique record detailing the procedures and services they have performed.

- Feature extraction from MPOP_PS: In order to ensure a unified feature representation across all providers, we adopt a standardized set of 513 HCPCS codes pertaining to ophthalmology. For each HCPCS code, we extract two essential billing statistics from the pivoted MPOP_PS—the number of services (NoS) and the number of Medicare beneficiaries (NoMB). This results in a feature set of size 1026 for each provider. More details on the HCPCS codes used can be found in Appendix B.

- Feature extraction from MPOP_P: We extract only 3 features for each provider from the MPOP_P data—total count of Medicare beneficiaries served by each provider (TotPatient), total payment made by Medicare to each provider (TotPayment), and the total payment made by Medicare to each provider for drug services (DrugPayment).

- Feature transformation: To enhance the informativeness of the features, we introduce two new features per HCPCS code—the ratio of NoS to NoMB and the ratio of NoMB to TotPatient. Additionaly, we introduce two supplementary features to further enrich the feature set—the ratio of TotPayment to TotPatient and the ratio of DrugPayment to TotPayment.

By combining the transformed features, we obtain a feature set of size 1028 for each provider.

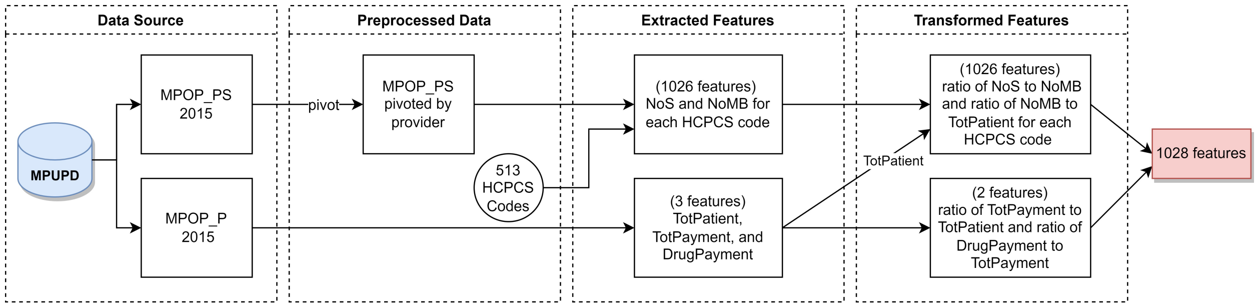

Data Source
Preprocessed Data
Extracted Features
Transformed Features
MPUPD
MPOP_PS 2015
MPOP_P 2015
pivot
MPOP_PS pivoted by provider
513 HCPCS Codes
(1026 features) NoS and NoMB for each HCPCS code
(3 features) TotPatient, TotPayment, and DrugPayment
TotPatient
(1026 features) ratio of NoS to NoMB and ratio of NoMB to TotPatient for each HCPCS code
(2 features) ratio of TotPayment to TotPatient and ratio of DrugPayment to TotPayment
1028 features

Figure 4.1.1: Data preprocessing flowchart. The presented flowchart outlines the sequential steps undertaken in the data preprocessing phase, specifically focusing on obtaining the feature set for a particular provider using their billing information from the year 2015.

## Dataset Labeling

Through collaboration with experienced physicians with a minimum of 10 years in practice and seasoned medical billers, we curated a labeled dataset consisting of 663 ophthalmologists based on their 2015 Medicare billing data. The assigned labels are either "non-fraudulent" or "overutilizer". The labeling process was conducted with the assumption that Medicare restrictions are consistent nationwide, ensuring equal opportunities for providers across the country to engage in overutilization activities. Each provider in the dataset was labeled by no less than 3 individuals to ensure consistency and accuracy. When there was any disagreement, an experienced fourth member of the team served as a tiebreaker to discuss and obtain consensus among the four. To ensure accuracy of labeling, further validation was made by a survey and test administered to practicing ophthalmologists. Baseline characteristics of the labeled providers can be found in Table 4.1.2. The labeling process is detailed in Appendix C.

Table 4.1.2: Characteristics of labeled providers

| Characteristic | Overutilizer | Non-fraudulent |
|---|---|---|
| Number | 200 | 463 |
| Location | | |
| California | 185 | 269 |
| New York | 38 | 171 |
| Average number of patients | 733.8 | 760.7 |
| Average number of services | 6904.3 | 4144.2 |
| Average Medicare payment per patient ($) | 917.6 | 520.8 |

**Preliminary statistical analysis.** An initial examination of the generated features for the labeled providers highlights the significant sparsity (a large number of zero features) present in the feature set. Each provider is associated with 1028 features, out of which 1026 features are derived from 513 distinct HCPCS codes (2 features per code). However, not all the HCPCS codes are utilized by every provider, and on average, ophthalmologists bill for fewer than 50 HCPCS codes annually. For example, in 2015, ophthalmologists in California billed an average of just 33 HCPCS codes. Figure 4.1.2 illustrates the sparsity pattern observed in the features associated with the providers in the labeled dataset. The observed sparsity necessitates the use of special techniques during the training process.

## 4.2 Overutilization Analysis via Machine Learning Techniques

Providers' billing behaviors, both the normal or the abnormal ones, can evolve over time due to various factors such as the introduction of new procedures, shifts in the medically

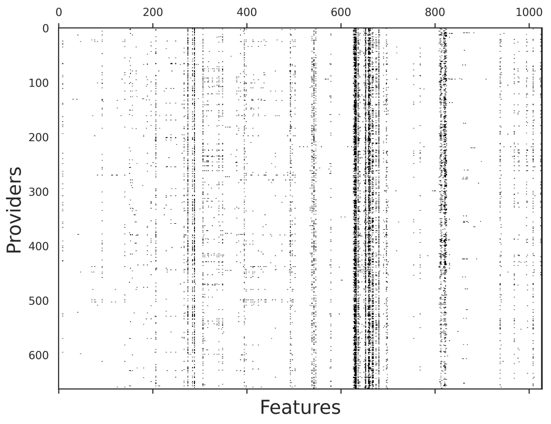


Figure 4.1.2: Sparsity map of the features associated with the providers in the labeled dataset. The plot shows the sparsity pattern of the feature space, indicating the utilization of HCPCS codes by labeled providers. Each cell represents the presence (non-zero value) or absence (zero value) of a specific HCPCS code for a provider.

necessary requirements of their patients, or efforts to avoid detection by auditors through normalized billing practices. To account for this dynamic nature, we employ models that aim to classify providers as either overutilizer (positive) or non-fraudulent (negative) based on their billing information within a specific calendar year. By focusing on yearly billing data, we can capture the evolving patterns and behaviors of providers, allowing for more accurate and timely detection of abnormal billing practice.

**ML models.** In our analysis, we experiment with a diverse set of both linear and nonlinear predictive models including K-nearest neighbor (KNN), logistic regression (LR), support vector machines (SVM), extreme gradient boosting (XGB), and multilayer perceptron (MLP). To improve the model performance, we also explore an ensemble technique called stacked generalization (stacking) [145] to develop an ensemble model by combining the predictions of the five aforementioned ML models. The diverse capabilities of the individual models are shown to provide improved predictive accuracy. More details on the deployed ML models are provided in Appendix D.

**Performance evaluation.** To ensure an unbiased evaluation of the models' performance, a repeated nested cross-validation (nested CV) approach is employed. See Appendix E for detailed technical description. The models' performance was evaluated using two metrics. The primary measure of performance is the area under the receiver operating characteristic curve (AUROC score), which quantifies the models' ability to discriminate between positive and negative instances across different classification thresholds. Additionally, we report the

Brier score assessing the models' confidence by evaluating the accuracy of their probabilistic predictions.

We also plot receiver operating characteristic (ROC) curves, which provide a graphical representation of the models' discrimination ability across different classification thresholds. These curves allow us to compare the models' performance in terms of true positive rate and false positive rate. Furthermore, we generate calibration plots to assess the calibration of the models. Calibration plots illustrate the agreement between the predicted probabilities of the models and the observed frequencies of the target variable. By visually examining the calibration plots, we can determine the models' reliability and assess if they are underconfident or overconfident in their predictions.

## Results

**Model performance.** The results of the model performance are presented in Table 4.2.1, which provides an overview of the mean performance scores along with their corresponding 95% confidence intervals (CIs) obtained through the repeated nested CV procedure. Additionally, Figure 4.2.1a displays the average ROC curves, showcasing the discriminative ability of each model.

It is evident that all models exhibit a certain albeit varying level of discriminative power, as their mean AUROC scores surpassed that of a completely random classifier (AUROC score of 0.50). Among the models, the KNN model demonstrated the lowest discrimination with an AUROC score of 0.740 (95% CI: 0.716-0.765), indicating its limitations in accurately distinguishing between overutilizer and non-fraudulent instances. Meanwhile, the LR, SVM, and MLP models exhibited almost similar performance across all evaluation metrics, with strong discriminatory capabilities, achieving AUROC scores higher than 0.85. The XGB model displayed even better discriminative power, with an AUROC score of 0.898 (95% CI: 0.887-0.909). However, the higher AUROC score was accompanied by lower predictive accuracy, as indicated by the Brier score. In contrast, the stacking ensemble model demonstrated the strongest discriminative power, with an AUROC score of 0.907 (95% CI: 0.896-0.918), and the highest predictive accuracy and confidence.

The calibration plots in Figure 4.2.1b illustrate the consistency between the observed and predicted probabilities of overutilization for the XGB and stacking ensemble models. See Appendix F for the rest of the models' calibration plots. Although the XGB model and the stacking ensemble model exhibit almost similar discriminative power (in terms of the AUROC score), a comparison of their calibration plots revealed a notable difference. The XGB model demonstrates lower consistency in its overutilization detection compared to the stacking ensemble model. Specifically, the XGB model is overconfident in its prediction, underestimating overutilization when the predicted probabilities are below 0.5 and overestimating overutilization when the predicted probabilities exceed 0.5.

Table 4.2.1: Performance measures for various ML models

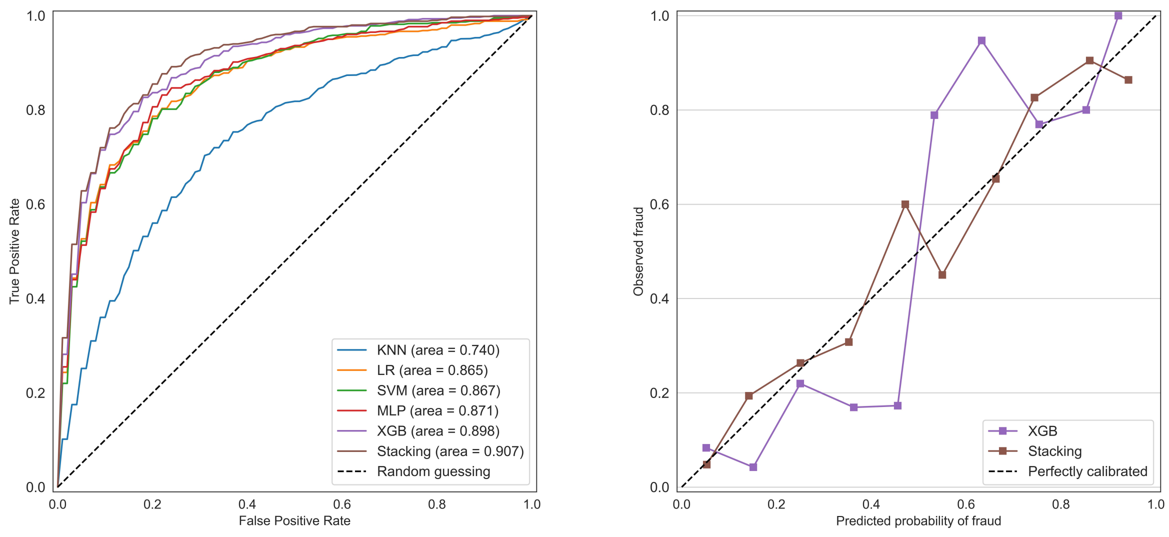
| Model | AUROC score (95% CI) | Brier score (95% CI) |
|---|---|---|
| KNN | 0.740 (0.716-0.765) | 0.182 (0.175-0.189) |
| LR | 0.865 (0.854-0.876) | 0.132 (0.127-0.138) |
| SVM | 0.867 (0.855-0.879) | 0.130 (0.123-0.137) |
| MLP | 0.871 (0.857-0.885) | 0.134 (0.125-0.143) |
| XGB | 0.898 (0.887-0.909) | 0.152 (0.136-0.168) |
| Stacking | 0.907 (0.896-0.918) | 0.109 (0.102-0.117) |

## Predicting Overutilization Statistics

Based on the model performance results, the stacking ensemble model is chosen and then trained on the entire labeled dataset to predict the labels for all ophthalmologists across the United States based on their Medicare billing data from 2021. Consequently, various overutilization statistics, including the Medicare overutilization rate and the monetary losses attributed to Medicare overutilization within the field of ophthalmology are obtained. We first conduct these calculations across the entire nation, and then within individual Medicare jurisdictions. Furthermore, we present a heatmap depicting the predicted overutilization rates across different states in the United States.

**Nationwide overutilization rate.** Among the 17,013 ophthalmologists enrolled in Medicare in 2021, our analysis predicts that 1,457 of them are likely to engage in overutilization activities, resulting in a predicted overutilization rate of approximately 8.6%.

**Nationwide monetary loss.** We compute the average Medicare payment per patient (MPPP) for both the predicted overutilizer and non-fraudulent providers. The 1,457 overutilizer ophthalmologists (mean=\$908.0, std=\$1021.2) compared to 15,556 non-fraudulent ophthalmologists (mean=\$542.0, std=\$841.3) had significantly higher MPPP (t(17011)=13.3, p << 0.01). The t-test is conducted using Scipy v1.8.1 [141]. This suggests that the overutilizer ophthalmologists tend to receive higher Medicare payments per patient compared to their non-fraudulent counterparts. Additionally, these overutilizer providers had a combined patient count of 932,520 and the total Medicare payment to them amounted to \$942.5 million. To put these numbers into perspective, if all the overutilizer physicians had similar MPPP to the average non-fraudulent physician (\$542.0), their total Medicare payment would have been \$505.4 million ($932,520 \times 542.0$). This indicates that there is a potential loss of \$437.1 million due to overutilization activities for just one year and just the field of ophthalmology.



(a) ROC curves (b) Calibration plots

Figure 4.2.1: ROC curves and calibration plots. Corresponding values of the area under the curve (AUROC score) for each are presented in Table 4.2.1. KNN (blue line) indicates k-nearest neighbor; LR (orange line), logistic regression; SVM (greed line), support vector machines; MLP (red line), multilayer perceptron; XGB (purple line), extreme gradient boosting; and Stacking (brown line), stacking ensemble.

We further provide a breakdown of the calculated statistics across different Medicare jurisdictions and states. Table 4.2.2 provides an overview of the 12 Medicare jurisdictions, including the states and territories they cover. The calculated statistics within each Medicare jurisdiction are shown in Figures 4.2.2 and 4.2.3. Figure 4.2.4 depicts the heat map of overutilization rates across different states in the US. More details on the Medicare jurisdictions' overutilization statistics are provided in Appendix G.

We remark that the significant variation in MPPP, as indicated by the high standard deviation calculated earlier, is primarily attributed to the inclusion of drug payments. If we exclude drug payments from the total payments, we anticipate that the average payment per patient for non-drug services among overutilizer providers will still be considerably higher than that of non-fraudulent providers. See Appendix G for a comparison.

It is also worth mentioning that in order to obtain precise labels for overutilization detection, we employed a specific probability threshold to convert the predicted probabilities into suspected overutilizer versus non-fraudulent labels. In order to strike a balance between precision and sensitivity (recall), we select a probability threshold of 0.353. With this threshold, the stacking ensemble model achieves an accuracy of 0.850 (95% CI: 0.836-0.862),

Table 4.2.2: Medicare jurisdictions and coverage areas

| Jurisdiction | States and territories | Number of ophthalmologists in 2021 |
|---|---|---|
| JE | CA, NV, HI, AS, GU, MP | 2149 |
| JF | WA, OR, ID, MT, WY, ND, SD, UT, AZ, AK | 1243 |
| JH | CO, NM, TX, OK, AR, MS, LA | 2143 |
| JL | PA, MD, DE, NJ, DC | 1855 |
| J5 | NE, KS, IA, MO | 662 |
| J8 | MI, IN | 825 |
| JJ | TN, AL, GA | 926 |
| JM | WV, VA, NC, SC | 1264 |
| JK | NY, VT, MA, CT, RI, NH, ME | 2502 |
| J6 | MN, WI, IL | 1248 |
| J15 | OH, KY | 779 |
| JN | FL, PR, VI | 1417 |

a specificity of 0.886 (95% CI: 0.867-0.903), a precision of 0.750 (95% CI: 0.720-0.779), and a sensitivity of 0.763 (95% CI: 0.733-0.793). More information on selecting the probability threshold is provided in Appendix H.

## Feature Importance Analysis

Feature importance analysis is a valuable technique in ML that enables understanding the relative importance of different features in making predictions. By assessing the impact of features on the stacking ensemble model's output, we can gain insights into which factors play a significant role in influencing the model's decisions in the detection of overutilization. One popular approach to feature importance analysis is using SHAP (SHapley Additive exPlanations).[91] SHAP quantifies the contribution of each feature in the prediction process. We utilize the Python package SHAP v0.41.0 [91] in our study.

Figure 4.2.5 shows the SHAP summary plot to visualize the feature importance values obtained from the stacking ensemble model analysis. In this plot, the features are listed along the y-axis of the plot, with the most important features at the top. Secondly, each feature is represented by a horizontal bar in the plot depicting the corresponding SHAP values. A positive SHAP value means positive impact on prediction (in our case, overutilization/fraud prediction), whereas a negative SHAP value corresponds to a negative impact. A higher positive value indicates a higher positive impact, and a lower negative value indicates a higher negative impact. Lastly, the color gradient within each bar represents the value of the

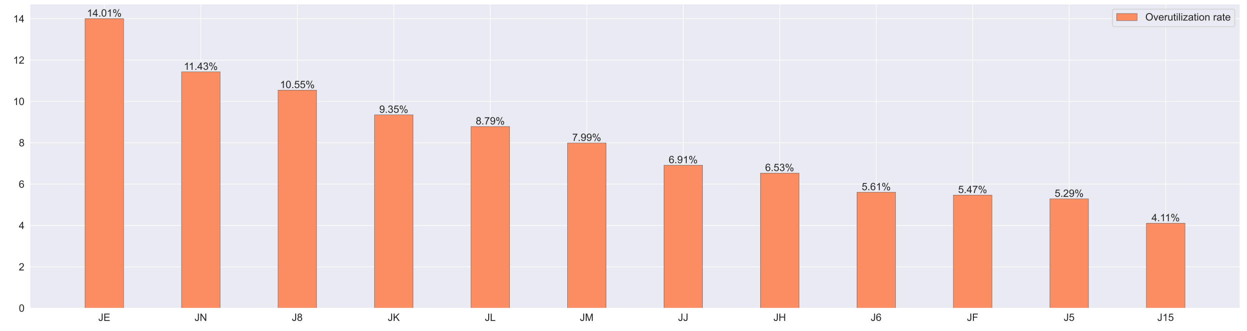


Figure 4.2.2: Predicted Overutilization Rates within Medicare Jurisdictions in 2021. The figure displays the overutilization rates, sorted in descending order, for each Medicare jurisdiction. The leftmost jurisdiction represents the highest overutilization rate, while the rightmost jurisdiction indicates the lowest fraud rate. The data provides insights into the distribution of predicted overutilization across different jurisdictions.

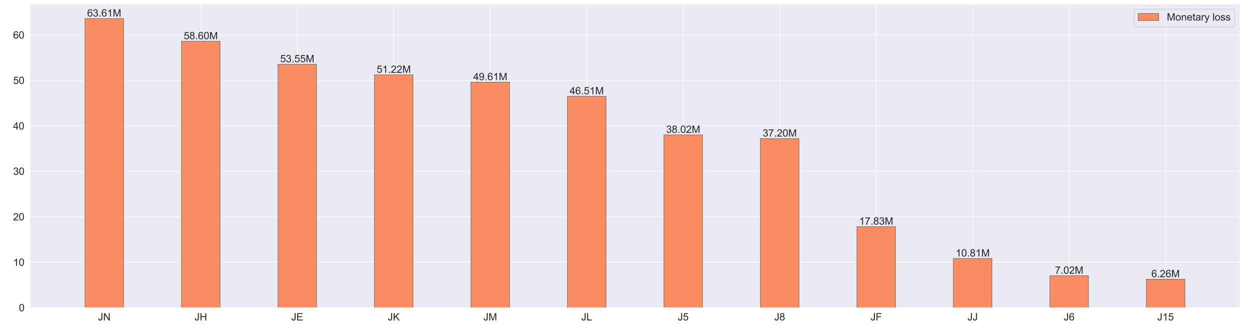


Figure 4.2.3: Monetary Losses on Medicare overutilization within Medicare Jurisdictions in 2021. The figure illustrates the amount of money lost due to overutilization, sorted in descending order, for each Medicare jurisdiction. The leftmost jurisdiction represents the highest monetary losses, while the rightmost jurisdiction indicates the lowest losses. The losses are calculated in the same way nationwide loss is calculated.

corresponding feature.

The first feature displayed at the top is the ratio of total amounts of Medicare payments to the total number of patients. The corresponding horizontal bar suggests that higher values of this ratio consistently have a positive impact on the model's predictions of overutilization. Other features listed in the plot present the HCPCS codes that could be more influential in determining overutilization.

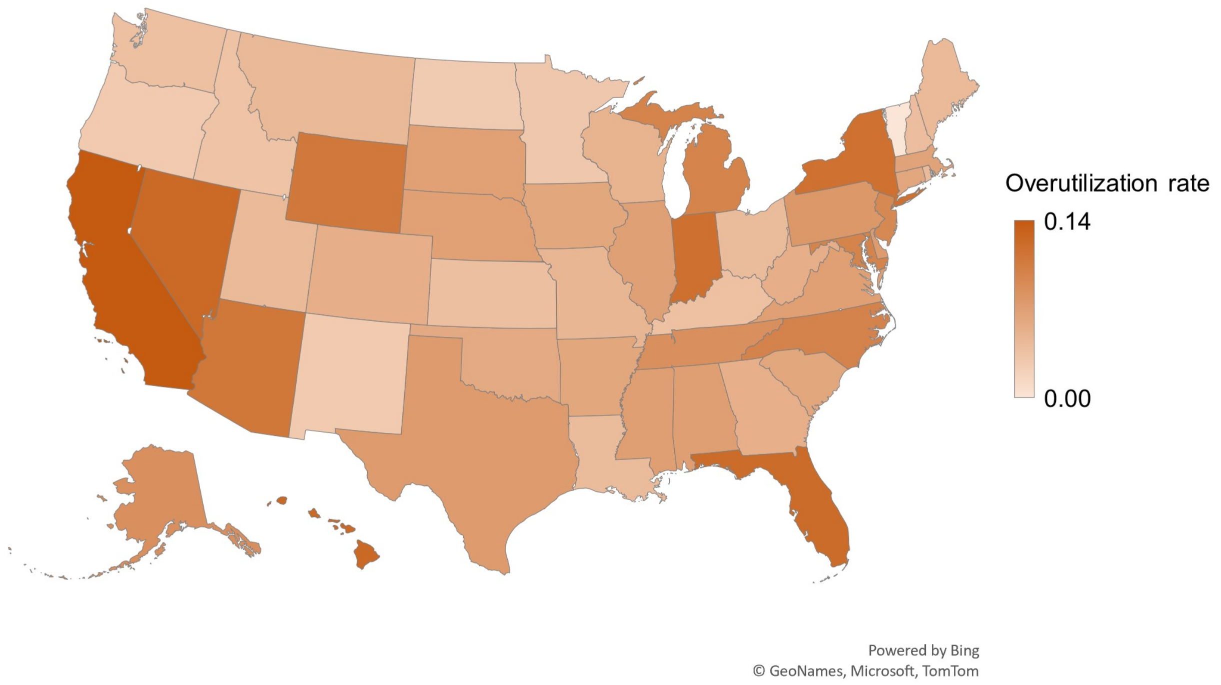


Figure 4.2.4: The heat map showcases the predicted overutilization rates across states in the United States, providing a visual representation of the variations in overutilization patterns across different regions. The darker colors represent higher rates of overutilization.

## 4.3 Discussion

The goal of our study was to test various ML models to identify the most effective screening approach for Medicare overutilization detection within the field of ophthalmology. This was uniquely accomplished by curating a comprehensive labeled dataset of ophthalmologists. Our medical team ensured the labeled dataset addressed the limitations of the LEIE dataset by tackling the issue of class imbalance and capturing nuanced fraudulent patterns and overutilization. Using our created dataset, we found that the stacking ensemble model enhanced overutilization detection by harnessing the strengths and diverse perspectives of individual ML models. In what follows, we further discuss the different components of our analysis and the key findings.

The engineered features are designed in very labor-intensive collaboration with physicians and medical experts, ensuring that they align with their domain expertise and thought process during the labeling of the data. First, the ratio of NoS to NoMB provides insights into the frequency of services rendered per beneficiary, allowing for a more nuanced understanding of

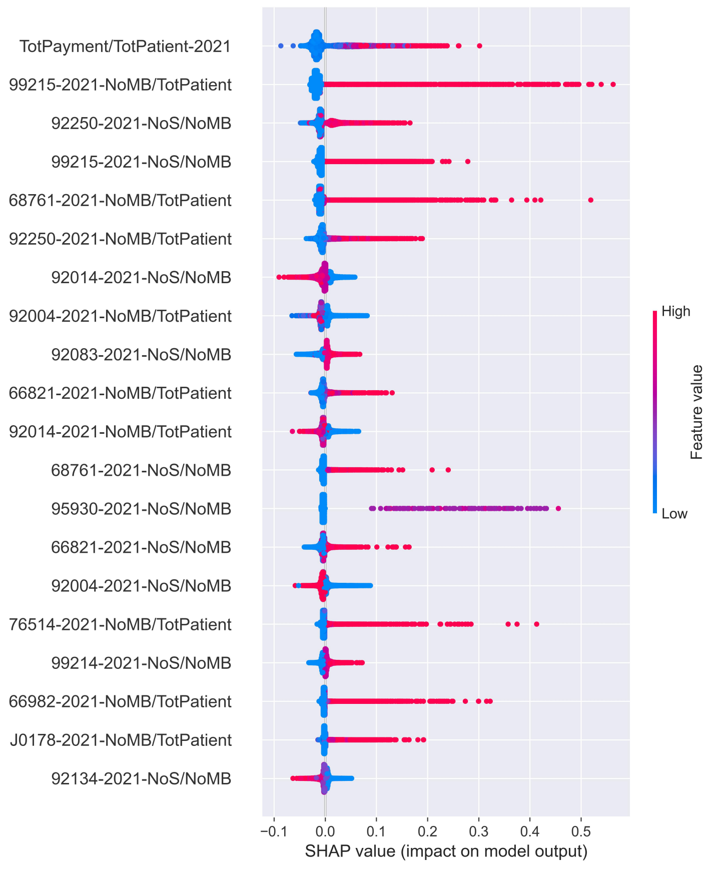


Figure 4.2.5: Shap Summary Plot.

utilization patterns. Second, the ratio of NoMB to TotPatient provides valuable information about the proportion of Medicare beneficiaries among all patients served, which can shed light on the demographics and patient population of the provider. Additionally, the ratio of TotPayment to TotPatient reflects the average Medicare payment per patient, offering insights into the financial aspects of the services provided. Lastly, the ratio of DrugPayment to TotPayment captures the proportion of drug-related payments within the total Medicare payment amount.

To evaluate the effectiveness of these features in overutilization detection, consider the following scenarios. Take the HCPCS code 66984, which represents extracapsular cataract removal. If the ratio of the NoS to NoMB ratio exceeds two in a given year, it would be highly suspicious. This is because, on average, each patient has only two eyes, and the average number of cataract surgeries per patient should not exceed two. (Note that the revision surgery has a separate code and can be distinguished from initial surgery.) Similarly, for the HCPCS code 95004, which pertains to allergy testing of the skin, if the NoMB to TotPatient ratio is 1 or higher, it indicates that allergy testing has been performed on every patient encounter. In the case of ophthalmologists, this suggests excessive and unwarranted billing since not every patient requires skin allergy testing every encounter. Additionally, the ratio of DrugPayment to TotPatient is particularly useful in distinguishing retina specialists from other subspecialties. The aforementioned ratio is often high among retina specialists due to the expensive nature of the anti-VEGF drugs they utilize.

Our labeled dataset provides a more balanced distribution between overutilizer and non-fraudulent providers compared to the LEIE dataset, with the minority class representing 30.2% of the dataset. In the specific context of ophthalmology, the LEIE dataset includes only 63 fraudulent ophthalmologists, while in 2021, there were over 17,000 ophthalmologists participating in Medicare. It is important to note that the LEIE dataset primarily focuses on extreme cases of fraud and may not capture the full range of overutilizing activities in the field. In contrast, our collaborative approach with physicians and medical billers allowed us to thoroughly examine the billing information of selected ophthalmologists, ensuring that our dataset represents a wider range of overutilizer and non-fraudulent providers. The detailed examination of billing patterns can only be accomplished when physicians or medical billers with years of experience are analyzing the data. Unfortunately this has never been done before because of the labor intensive nature of creating the dataset as well as industry wide lack of domain expertise, i.e., claim analysis is rarely done by physicians and those who understand the patterns of clinical care.

The comparative evaluation of different ML models revealed varying levels of discrimination performance in detecting Medicare overutilization within the field of ophthalmology. The KNN model exhibited the lowest discrimination ability, while the LR, SVM, and MLP models demonstrated similar strong performance. The XGB model displayed even better discrimination power, although with lower predictive accuracy. However, it was the stacking ensemble model that consistently showcased the highest performance. The superior discrimination performance as well as the high predictive confidence of the stacking ensemble model indicates that combining the predictions of multiple models using the stacked generalization technique can lead to improved overutilization detection outcomes. Therefore, more attention should be directed towards ensemble approaches, as they have the capability to leverage the strengths of individual models and produce improved overutilization detection outcomes.

Using our stacking ensemble model trained on the entire labeled dataset, we were able to estimate the nationwide overutilization rate as 8.6%, which falls within the range of existing fraud estimates of 3-10%.[137] Additionally, we estimated the annual monetary loss on overutilization and fraud in Medicare for the field of ophthalmology to be approximately $437 million. By extrapolating this estimate to the entire Medicare system, considering ophthalmology's share of 0.89% of the Medicare budget, we estimate the total money lost on overutilization and fraud in Medicare to be around $49.1 billion per year. Considering the $900.8 billion total cost of Medicare program in 2021, this implies that approximately 5.5% of the Medicare budget is lost due to overutilization and possible fraud. These findings highlight the dire need for effective prevention strategies.

Current prevention and detection strategies prove to be very insufficient. CMS often outsources and relies on overly simplistic ratios to identify potentially irregular behavior. For example, one analysis is the ratio of complex cataract surgery to total surgeries done by a physician. This would not necessarily account for those physicians expert in complex cataract surgery who are referred these particularly difficult cases. Additionally, these analyses lack the consideration of subspecialties and fail to capture the intricacies and complexities of physician billing practices. Incorporating real-time machine learning for claims analysis would be a huge step forward in both the deterrence and post claim payment recoupment of overutilization. This way, flagged claims could undergo further scrutiny without the need to manually review thousands of appropriately billed and paid claims

We additionally conducted estimations of overutilization rates and associated monetary losses within each Medicare jurisdiction. The predicted rates exhibited variations across different regions of the United States. Notably, Jurisdiction JE, which includes California, displayed the highest predicted overutilization rate at 14.01%, followed by JN with 11.43%. Conversely, the jurisdiction J15 encompassing Ohio and Kentucky had the lowest overutilization rate at 4.11%. Surprisingly, although jurisdiction JE had the highest overutilization rate, it did not have the highest monetary loss. Instead, JN incurred the highest monetary loss at $63.6 million, followed by JH with $58.6 million. Several factors could contribute to this distribution, including demographic disparities, variations in healthcare provider density, regional differences in fraud awareness and enforcement, cultural and behavioral norms, and variances in healthcare practices. These findings underscore the need for targeted measures for detecting and preventing overutilization that are customized to specific regions.

The feature importance analysis revealed that the payment per patient ratio was a strong indicator of overutilization, with higher values indicating a higher likelihood of overutilization activity. This association can be attributed to the potential incentive of using certain highly reimbursed HCPCS codes frequently, even when not medically necessary. Moreover, the excessive utilization of specific HCPCS codes has been identified as a significant indicator of overutilization. This heightened utilization is observed through either a high ratio of beneficiaries to total patients or a high ratio of services to beneficiaries. The HCPCS code 95930 (visual evoked potential) is an example to demonstrate how excessive use serves as a red flag for fraudulent behavior. In ophthalmology, this code is narrowly applicable, primarily employed by neuroophthalmologists to diagnose optic neuropathies or malingering through measurements of the visual cortex's response to visual stimuli. However, when a general ophthalmologist utilizes this code for nearly every patient and every visit, it deviates from the standard of care and signifies excessive utilization. It is apparent that there may be a financial motivation behind this utilization. It is noteworthy that the Medicare fee schedule allowable for HCPCS code 95930 is $134.35 in certain geographic locations, and when multiplied by all the patients in a given practice can be financially very lucrative.

It is crucial to mention that the high utilization of an HCPCS codes is not necessarily indicative of fraud. For instance, an ophthalmologist may have a very high usage of codes such as 99215 (highest reimbursing patient examination code) as well as many esoteric codes such as the aforementioned 95930 or 92275 (retinal electrography). Individual code analysis may pinpoint this physician with overutilization, but an analysis of all claims of their practice may reveal that he or she has a neuro-ophthalmology subspecialty, with very complex patients and few total encounters over the course of a year compared to a high volume general ophthalmologist. This suggests that overutilization detection cannot solely rely on individual code analysis but should consider patterns and relationships among multiple codes to gain a comprehensive understanding of potential fraudulent behavior.

Never before has domain expertise of ophthalmologists and medical billers been combined with machine learning to analyze the medicare claims database. The implications of this study are that medicare could save $50 billlion dollars per year by incorporating a properly trained machine learning model under each specialty in its claims analysis. We focused on the most egregious and blatant forms of overutilization. If applied to more subtle forms of overutilization, then the cost savings would be even greater. The additional funds generated could be applied to lowering the eligibility age of medicare recipients, expanding clinical and drug coverage, or raising reimbursement rates for the vast majority of honest physicians providing clinically and epidemiologically appropriate care. In this modern era where machine learning has the power to analyze individual provider behavior, and to compare it with the totality of the medicare data set, it is long overdue that we leverage the computational power of big data, to use medicare dollars efficiently and appropriately.

## 4.4 Limitations

Despite the aforementioned strengths and critical findings, our study has a number of limitations. The first limitation of our work is the limited number of data points in our created dataset. The meticulous and labor intensive process of analyzing and evaluating billing information for each provider resulted in a relatively small dataset. This limited sample size may not capture the full variations and distributions of overutilizer and non-fraudulent data, particularly given its high-dimensional nature. To enhance the accuracy and generalizability of the models, further data acquisition efforts are necessary to expand the dataset.

Secondly, it is important to acknowledge that the reported overutilization statistics are based on model predictions, which are not infallible. As such, it is crucial to interpret the model's predictions as estimations rather than absolute truths. Validation and verification through additional means, such as manual audits or investigations, are necessary to ascertain the true extent of overutilization and possibly fraudulent activities, and the associated financial losses. While the findings can guide decision-making and resource allocation, ongoing refinement and improvement of the models and methodologies are essential to ensure their accuracy and reliability in detecting aberrant behavior.

Further, it is often difficult to distinguish high or overutilization from blatant fraud, which has intent behind it. Utilization ultimately is a broad continuum and often the medical record is needed to justify clinical behavior. This paper addresses clinician behavior by yearly patterns, but Medicare ascertains fraud by individual patient records to determine if a procedure was done and if clinically appropriate. Often on an individual basis, actions can be defensible, even if occurring on a repeated basis.

Lastly, it is worth noting that although some components of our study, such as data preprocessing and feature engineering, can be applied to detect Medicare overutilization in other subspecialties, the availability of a labeled dataset specific to each subspecialty remains a requirement. This underscores the need for domain expertise and collaboration with healthcare professionals in that subspecialty to create labeled datasets that accurately capture overutilization patterns within specific subspecialties. This can be very labor intensive and costly and represents the largest barrier to widespread incorporation of this methodology.

# Appendix

## 4.A Data Source Details

**MPOP_PS data:** This data is structured in comma-separated values (CSV) format and comprises individual files for each calendar year. Every record in the data identifies a provider, primarily by the provider's National Provider Identifier (NPI), and a procedure that the provider has submitted a claim to Medicare for, which is represented with a Healthcare Common Procedure Coding System (HCPCS) code. In addition to the NPI, each record contains several other elements that offer detailed information about the provider including their name, demographics, and the provider's type (e.g. ophthalmology). Furthermore, each record includes essential aggregate statistics pertinent to the listed procedure. These statistics encompass the number of times the provider performed the procedure within the given year, the average billing amount for that procedure, and other relevant information. Table 4.A.1 lists the subset of features from the MPOP_PS data used in our analysis.

Table 4.A.1: Features used from the MPOP_PS data

| Name | Description | Type |
|---|---|---|
| Rndrng_NPI | National Provider Identifier (NPI) for the rendering provider on the claim | Categorical |
| HCPCS_Cd | HCPCS code used to identify the specific medical service furnished by the provider | Categorical |
| Tot_Srvcs | Number of services provided | Numerical |
| Tot_Benes | Number of distinct Medicare beneficiaries receiving the service | Numerical |

**MPOP_P data:** This data is structured in comma-separated values (CSV) format and comprises individual files for each calendar year. Each record in the data represents a unique provider, primarily identified by their National Provider Identifier (NPI). Additional information provided by each record includes demographic details, provider's type, and various aggregate statistics related to their services and procedures such as total number of unique HCPCS codes, total number of Medicare beneficiaries receiving services from the provider, etc. Table 4.A.2 lists the subset of features from the MPOP_P data used in our analysis.

Table 4.A.2: Features used from the MPOP_P data

| Name | Description | Type |
|---|---|---|
| Rndrng_NPI | National Provider Identifier (NPI) for the rendering provider on the claim | Categorical |
| Tot_Benes | Total Medicare beneficiaries receiving services from the provider. | Numerical |
| Tot_Mdcr_Pymt_Amt | Total amount that Medicare paid after deductible and coinsurance amounts have been deducted for all the provider's line item services | Numerical |
| Drug_Mdcr_Pymt_Amt | Total amount that Medicare paid after deductible and coinsurance amounts have been deducted for all the provider's line item drug services | Numerical |

## 4.B HCPCS Codes

The set of 513 HCPCS codes used to construct the feature vectors is derived from aggregating the services and procedures performed by ophthalmologists located in California over the period from 2013 to 2019. This timeframe allows us to capture a substantial amount of data and encompass the diversity of services rendered by ophthalmologists during those years. “hcpcs_codes.xltx" lists all the 513 HCPCS codes used.

California, being home to the highest number of ophthalmologists compared to other states (in 2015, 1,975 out of 17,698 ophthalmologists in the United States were located in

California), provides a rich source of information for constructing a representative set of HCPCS codes. By leveraging the data from California, we can ensure that the selected 513 HCPCS codes encompass a wide range of procedures and services commonly performed within the field of ophthalmology.

## 4.C Labeling Process

The initial set of labels was generated manually by utilizing the Wall Street Journal's Medicare Unmasked graphic [146], which provided visual representations of Medicare reimbursements received by providers in a given year. This program offered insights into various aspects, including total payments, patient counts, payments per patient, and a breakdown of reimbursements by service category. It also provided information on HCPCS codes used, the number of unique patients receiving each procedure, and the total Medicare reimbursement for specific HCPCS codes in the year under consideration (2015 in our case). The labeling team thoroughly examined this information to determine the epidemiological accuracy of the billed HCPCS codes. This involved considering factors such as the ratio of HCPCS codes to the number of Medicare beneficiaries, the ratio of higher-reimbursing procedures to lower-reimbursing procedures, and the provider's reimbursement percentile at the state and national levels.

The labeling process involved multiple iterations of training the SVM model on the already labeled dataset, using the trained model to predict the likelihood of unlabeled providers being fraudulent, and selecting a random sample of 50 providers who were on the borderline (with a probability close to 0.5) for further labeling by the team. Sampling providers who were on the borderline helped improve the model's ability to distinguish between fraudulent and non-fraudulent cases. While it was relatively easier to identify blatantly fraudulent or non-fraudulent doctors, borderline cases posed greater challenges. Obtaining labels for this borderline sample contributed to enhancing the model's accuracy in detecting fraud.

However, due to the difficulty in definitively determining the fraudulent or non-fraudulent nature of borderline doctors, the process of generating these labels was time-consuming. Manual analysis encompassing various dimensions, such as the number of services performed per code, the total number of patients for each physician, and the physician's subspecialty, was conducted to create a label for each billing pattern. Given the meticulous nature of this analysis, generating an adequate number of labels to train the model required a significant amount of time

## 4.D ML Models Details

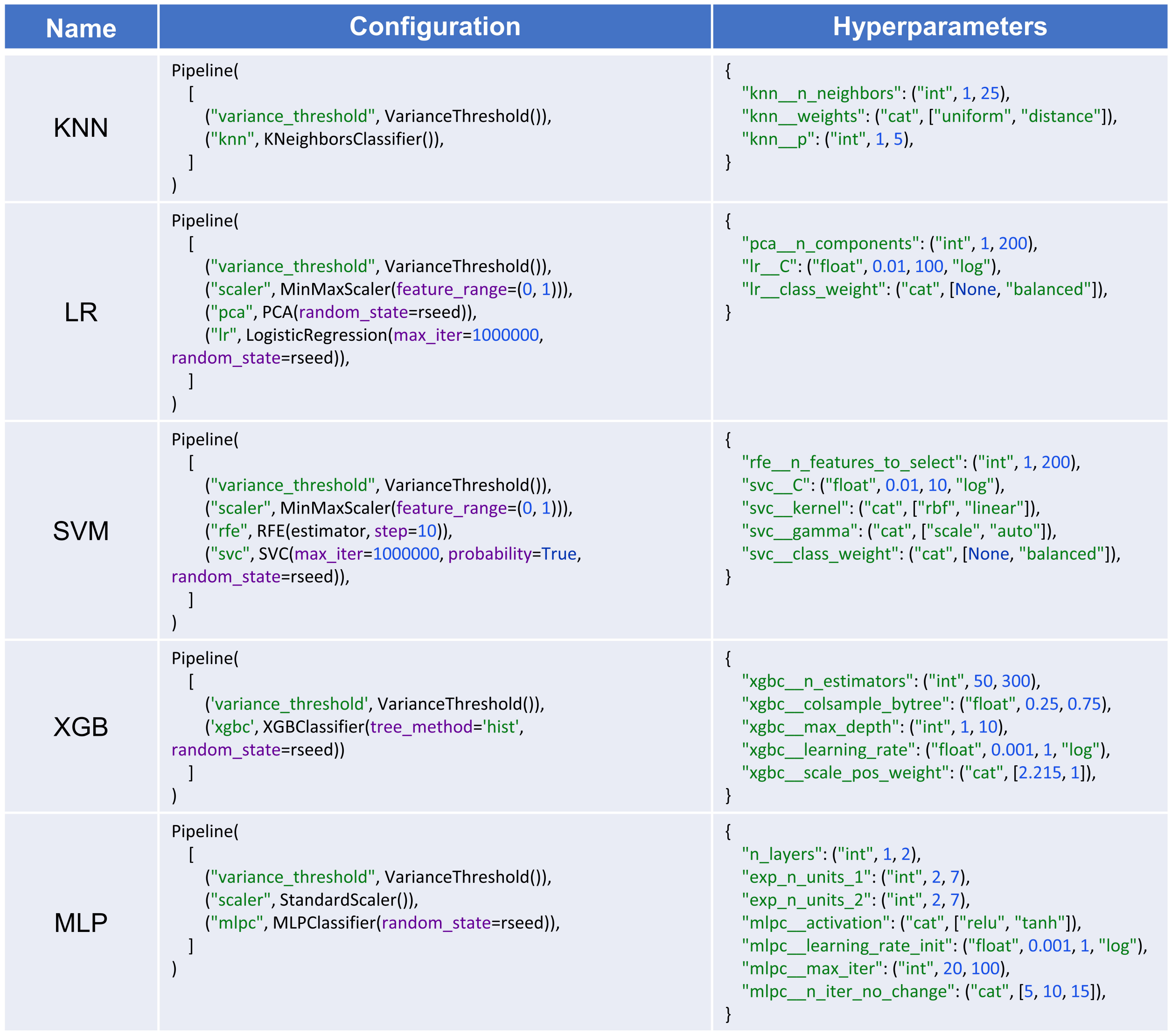
### Models' Configuration and Hyperparameters

In our analysis, we develop a diverse set of both linear and nonlinear predictive models including k-nearest neighbor (KNN), logistic regression (LR), support vector machines (SVM), extreme gradient boosting (XGB), and multilayer perceptron (MLP). Each model is embodied in a specific pipeline that incorporates various preprocessing techniques, such as feature scaling (e.g., min-max normalization, standardization) and dimensionality reduction methods (e.g., principal component analysis, recursive feature elimination, feature subsampling) to enhance their performance. It is important to note that when we refer to the models, we are specifically referring to the complete pipelines they are part of. Figure 4.D.1 shows the exact configuration and hyperparameters associated with the deployed pipelines.

The hyperparameter settings are used during the nested CV procedure to fine-tune the models. For each model, the hyperparameter settings are defined as a Python dictionary in which each key defines a hyperparameter and its corresponding value determines the type and search domain for that hyperparameter. The type and range of a hyperparameter is formatted as a tuple in which the first element defines its type and could be either `"int"` (integer), `"float"`, or `"cat"` (categorical). The rest of elements determine the search domain. For instance, `("int", 1, 25)` defines an integer between 1-25, or `("float", 0.01, 10, ‘‘log")` defines a float in the range 0.01-10 sampled in the logarithmic domain. Additionally, `(" cat", [None, ‘‘balanced"])` determines a categorical variable which could be either `None` or `"balanced"`.

Further information regarding the pipelines' configurations and hyperparameters are listed below:

- All analyses are done using Python v3.8. The KNN, LR, SVM,and MLP models were implemented using scikit-learn v1.1.2 [109]. The XGB model is implemented using XGBoost v1.6.2 [25].
- To expedite the training process, preprocessing methods like feature scaling are employed. The primary technique utilized for feature scaling is min-max normalization, which preserves the non-negativity and sparsity of the features. Standardization, on the other hand, is exclusively applied to MLP due to improved performance compared to min-max normalization, as observed in our experiments.
- In our analysis, we employ multiple techniques for dimensionality reduction. The Sklearn library's VarianceThreshold module is utilized to eliminate constant features, which are essentially features that consist entirely of zeros. Additionally, PCA (Principal Component Analysis) and RFE (Recursive Feature Elimination) are specifically employed in the LR and SVM pipelines, respectively. Furthermore, in the XGB pipeline, we restrict the model from utilizing the entire feature set by setting the `colsample_bytree` argument to a value less than one.
- Certain models including LR, SVM, and XGB offer the ability to address class imbalance by resampling the training data. For instance, by setting `"lr__class_weight"` to `"balanced "`, the LR model oversamples the minory (fraudulent) class. This helps mitigate the impact of class imbalance.



| Name | Configuration | Hyperparameters |
|---|---|---|
| KNN | Pipeline(<br>[<br>("variance_threshold", VarianceThreshold()),<br>("knn", KNeighborsClassifier()),<br>]<br>) | {<br>"knn__n_neighbors": ("int", 1, 25),<br>"knn__weights": ("cat", ["uniform", "distance"]),<br>"knn__p": ("int", 1, 5),<br>} |
| LR | Pipeline(<br>[<br>("variance_threshold", VarianceThreshold()),<br>("scaler", MinMaxScaler(feature_range=(0, 1))),<br>("pca", PCA(random_state=rseed)),<br>("lr", LogisticRegression(max_iter=1000000,<br>random_state=rseed)),<br>]<br>) | {<br>"pca__n_components": ("int", 1, 200),<br>"lr__C": ("float", 0.01, 100, "log"),<br>"lr__class_weight": ("cat", [None, "balanced"]),<br>} |
| SVM | Pipeline(<br>[<br>("variance_threshold", VarianceThreshold()),<br>("scaler", MinMaxScaler(feature_range=(0, 1))),<br>("rfe", RFE(estimator, step=10)),<br>("svc", SVC(max_iter=1000000, probability=True,<br>random_state=rseed)),<br>]<br>) | {<br>"rfe__n_features_to_select": ("int", 1, 200),<br>"svc__C": ("float", 0.01, 10, "log"),<br>"svc__kernel": ("cat", ["rbf", "linear"]),<br>"svc__gamma": ("cat", ["scale", "auto"]),<br>"svc__class_weight": ("cat", [None, "balanced"]),<br>} |
| XGB | Pipeline(<br>[<br>('variance_threshold', VarianceThreshold()),<br>('xgbc', XGBClassifier(tree_method='hist',<br>random_state=rseed))<br>]<br>) | {<br>"xgbc__n_estimators": ("int", 50, 300),<br>"xgbc__colsample_bytree": ("float", 0.25, 0.75),<br>"xgbc__max_depth": ("int", 1, 10),<br>"xgbc__learning_rate": ("float", 0.001, 1, "log"),<br>"xgbc__scale_pos_weight": ("cat", [2.215, 1]),<br>} |
| MLP | Pipeline(<br>[<br>("variance_threshold", VarianceThreshold()),<br>("scaler", StandardScaler()),<br>("mlpc", MLPClassifier(random_state=rseed)),<br>]<br>) | {<br>"n_layers": ("int", 1, 2),<br>"exp_n_units_1": ("int", 2, 7),<br>"exp_n_units_2": ("int", 2, 7),<br>"mlpc__activation": ("cat", ["relu", "tanh"]),<br>"mlpc__learning_rate_init": ("float", 0.001, 1, "log"),<br>"mlpc__max_iter": ("int", 20, 100),<br>"mlpc__n_iter_no_change": ("cat", [5, 10, 15]),<br>} |

Figure 4.D.1: Configuration and hyperparameters of deployed pipelines. The figure illustrates the detailed configuration and hyperparameter settings for the pipelines utilized in the training process.

## Stacking

In addition to the individual models, we also explore the benefits of ensemble techniques in improving overall performance. We leverage stacked generalization (stacking) [145] to develop an ensemble model, which combines the predictions of the five aforementioned ML models. Stacking involves the use of an additional model called the "meta-learner" to learn the optimal way of combining the predictions from the base models. In our ensemble model, the base models, KNN, LR, SVM, XGB, and MLP, generate predictions that are then fused together using an LR-based meta-learner. This ensemble approach allows for leveraging the diverse capabilities of the individual models and can lead to improved predictive accuracy. The exact meta-learner used in our analysis is `LogisticRegression(penalty="none", max_iter =1000, random_state=0)`.

In the following explanation, we outline the training and testing process for the stacking ensemble model using a given training set. Initially, the 5 base models undergo fine-tuning through k-fold cross-validation on the training set. During this procedure, the predictions made by each tuned base model are stacked together. By merging the stacked predictions from all the tuned base models, we construct the training set for the meta learner. Subsequently, all the tuned models are trained using the entire training set, while the meta learner is trained using its corresponding training set. For testing and making predictions on unseen data, we first utilize the trained base models to generate predictions, and then these predictions are combined using the trained meta learner.

Please note that to evaluate the performance scores of the stacking ensemble model using nested CV, the described procedure is repeated multiple times, depending on the number of outer loops in nested CV.

## 4.E Nested CV

The nested CV procedure consists of an outer loop for model evaluation while an inner loop tunes the hyperparameters. The performance estimates are calculated by averaging the test scores obtained across the different dataset splits in the outer loop. Varma and Simon [138] demonstrated that this approach, which avoids pooling the training and test data, produces nearly unbiased performance estimates. To further reduce bias and obtain more accurate estimates, the nested CV procedure could be repeated multiple times using different random shuffles of the labeled data. In our analysis, we utilized three random shuffles of nested CV with 10 outer loops and 5 inner loops, ensuring a robust assessment of the models' performance. We remark that in order to conduct the hyperparameter tuning procedure in the inner loop of the nested CV, we used the Optuna framework [2]. Optuna is a powerful optimization library that efficiently searches for the optimal hyperparameters of the models. For more details on the nested cross-validation procedure, including information on determining the number of outer and inner loops and the computational implications, please refer to the online article by Jason Brownlee [17].

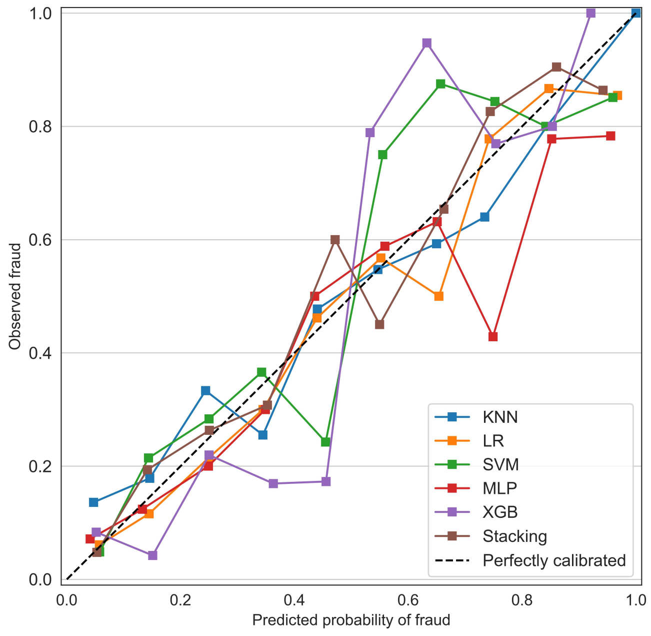


Figure 4.F.1: Calibration plots. KNN (blue line) indicates k-nearest neighbor; LR (orange line), logistic regression; SVM (greed line), support vector machines; MLP (red line), multilayer perceptron; XGB (purple line), extreme gradient boosting; and Stacking (brown line), stacking ensemble.

## 4.F Calibration Plots

## 4.G Further Overutilization Statistics

### Overutilization Statistics for Medicare Jurisdictions

Below, we present the details of overutilization statistics computed for each Medicare jurisdiction. (Note that 0.0 as p-value means a value significantly smaller than 0.01.)

### Non-Drug Medicare Payment Per Patient

The significant variation in Medicare payment per patient, as indicated by the high standard deviation calculated earlier, is primarily attributed to the inclusion of drug payments. If we exclude drug payments from the total payments, we anticipate that the average payment per patient for non-drug services among overutilizing providers will still be considerably higher

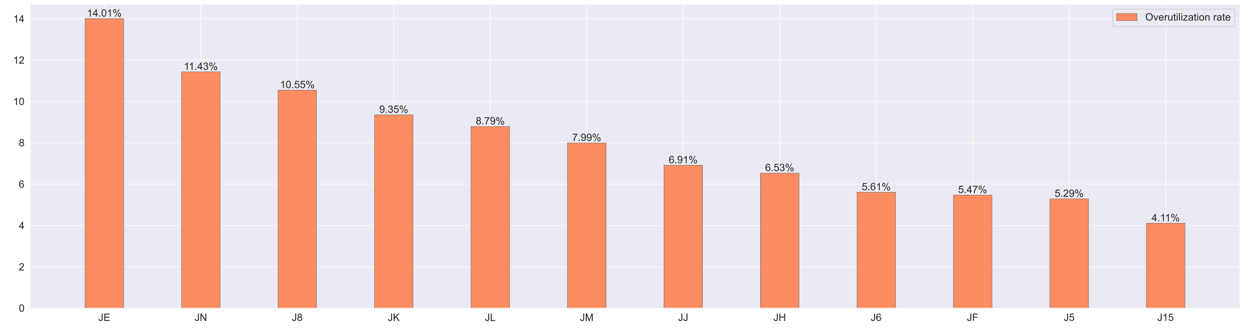


Figure 4.G.1: Predicted overutilization rate within Medicare jurisdictions in 2021.

Table 4.G.1: Overutilization rate within Medicare jurisdictions details

| | JE | JF | JH | JL | J5 | J8 | JJ | JM | JK | J6 | J15 | JN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Overutilizer # | 301 | 68 | 140 | 163 | 35 | 87 | 64 | 101 | 234 | 70 | 32 | 162 |
| Non-fraud # | 1848 | 1175 | 2003 | 1692 | 627 | 738 | 862 | 1163 | 2268 | 1178 | 747 | 1255 |
| overutilization rate (%) | 14.01 | 5.47 | 6.53 | 8.79 | 5.29 | 10.54 | 6.91 | 7.99 | 9.35 | 5.61 | 4.11 | 11.4 |

than that of non-fraudulent providers. Below, we detail this statistic across the nation and within Medicare jurisdictions (See Figure 4.G.4 and Table 4.G.4.)

Across the nation, the 1,457 overutilizing/fraudulent ophthalmologists (mean=\$545.4, std=\$388.9) compared to 15,556 non-fraudulent ophthalmologists (mean=\$302.9, std=\$157.1) had significantly higher non-drug Medicare payment per patient (t(17011)=21.3, $p << 0.01$).

## 4.H Probability Threshold

To compute performance metrics such as accuracy, specificity, precision, and sensitivity (recall), a specific probability threshold is required. Typically, these metrics are computed using a default threshold of 0.5, but this may not be optimal for imbalanced data situations.[116] Additionally, depending on the application, the emphasis may be placed on either precision or sensitivity, which would warrant a higher or lower threshold, respectively.

In the case of Medicare overutilization/fraud, if the primary goal is to avoid missing instances of overutilization/fraud, a higher sensitivity is desired, leading to a lower probability threshold. This approach would detect more providers as overutilizer/fraudulent, but it carries the risk of mislabeling non-fraudulent providers, potentially damaging their reputation

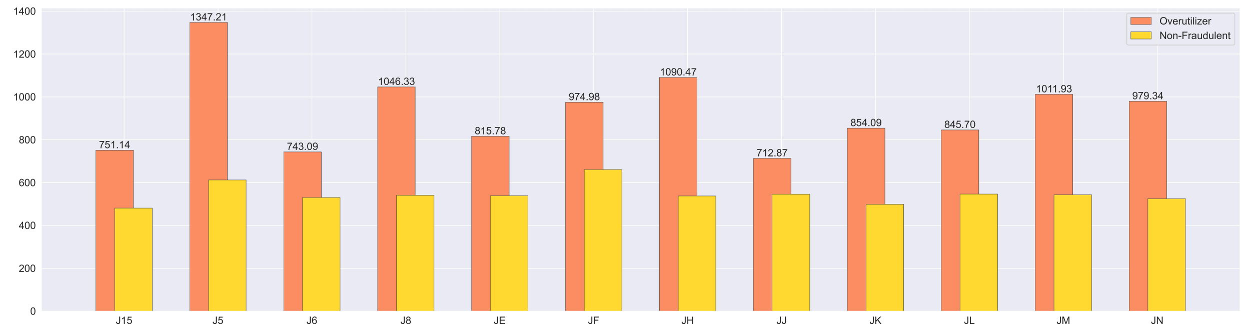


Figure 4.G.2: Comparing the average payment per patient between overutilizer and non-fraudulent ophthalmologists within Medicare Jurisdictions 2021.

Table 4.G.2: Average payment per patient within Medicare jurisdictions details

| | JE | JF | JH | JL | J5 | J8 | JJ | JM | JK | J6 | J15 | JN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Overutilizer # | 301 | 68 | 140 | 163 | 35 | 87 | 64 | 101 | 234 | 70 | 32 | 162 |
| Overutilizers mean ($) | 815.8 | 975.0 | 1090.5 | 845.67 | 1347.2 | 1046.3 | 712.9 | 1011.9 | 854.1 | 743.1 | 751.1 | 979.3 |
| Overutilizers std ($) | 819.8 | 1108.5 | 1287.9 | 1000.1 | 1456.4 | 1193.5 | 952.5 | 1208.3 | 963.6 | 757.9 | 909.3 | 925.9 |
| Non-fraud # | 1848 | 1175 | 2003 | 1692 | 627 | 738 | 862 | 1163 | 2268 | 1178 | 747 | 1255 |
| Non-frauds mean ($) | 538.7 | 661.0 | 537.3 | 546.3 | 612.0 | 540.7 | 545.8 | 543.1 | 498.6 | 530.5 | 480.8 | 524.7 |
| Non-frauds std ($) | 1423.9 | 850.6 | 694.2 | 754.7 | 831.4 | 683.1 | 747.2 | 788.4 | 657.6 | 720.2 | 603.1 | 701.8 |
| t-value | 4.8 | 2.3 | 5.0 | 3.7 | 3.0 | 3.9 | 1.4 | 3.8 | 5.5 | 2.3 | 1.7 | 6.0 |
| p-value | 0.0 | 0.01 | 0.0 | 0.0 | 0.003 | 0.0 | 0.09 | 0.0 | 0.0 | 0.01 | 0.05 | 0.0 |

if publicly announced. However, if the goal is to minimize mislabeling, a higher precision is preferred, requiring a higher probability threshold. While this is beneficial for public disclosure of overutilizing/fraudulent providers, it poses a financial burden on Medicare, as more instances of abuse go undetected.

In this study, we aim to strike a balance between precision and sensitivity. Thus, we seek a probability threshold that achieves a trade-off between these metrics. The F-score is optimized to determine such a balance, and the threshold is adjusted accordingly to maximize the average F-score obtained through the repeated nested CV procedure. This results in a probability threshold of 0.353. Accordingly, providers with a predicted overutilization/fraud probability exceeding 35.3% are labeled as overutilizer/fraudulent, while those below this

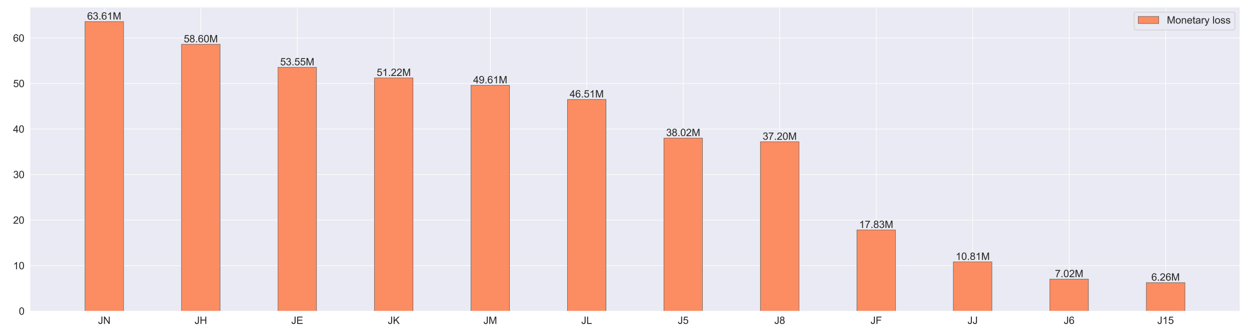


Figure 4.G.3: Monetary loss within Medicare jurisdictions in 2021.

Table 4.G.3: Monetary loss within Medicare jurisdictions details

| | JE | JF | JH | JL | J5 | J8 | JJ | JM | JK | J6 | J15 | JN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Overutilizers total payment ($m) | 150.5 | 43.5 | 104.0 | 108.6 | 54.2 | 66.1 | 35.8 | 91.8 | 117.0 | 27.0 | 13.6 | 130 |
| Monetary loss ($m) | 53.5 | 17.8 | 58.6 | 46.5 | 38.0 | 37.2 | 10.8 | 49.6 | 51.2 | 7.0 | 6.2 | 63.6 |

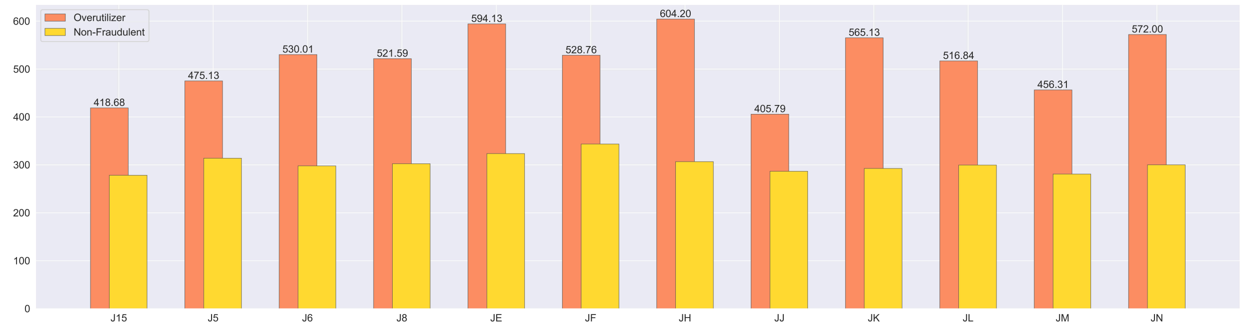


Figure 4.G.4: Comparing the average non-drug Medicare payment per patient between overutilizer and non-fraudulent ophthalmologists within Medicare Jurisdictions 2021.

threshold are considered non-fraudulent. The accuracy, specificity, precision, and sensitivity of the stacking ensemble model obtained using the probability threshold of 0.353 are presented in Table 4.H.1. The metrics are calculated via the repeated nested CV procedure.

Table 4.G.4: Average non-drug Medicare payment per patient within Medicare jurisdictions details

| | JE | JF | JH | JL | J5 | J8 | JJ | JM | JK | J6 | J15 | JN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Overutilizer # | 301 | 68 | 140 | 163 | 35 | 87 | 64 | 101 | 234 | 70 | 32 | 162 |
| Overutilizers mean ($) | 594.1 | 528.8 | 604.2 | 516.8 | 475.1 | 521.6 | 405.8 | 456.3 | 565.1 | 530.0 | 418.7 | 572 |
| Overutilizers std ($) | 349.6 | 219.0 | 766.6 | 235.9 | 220.0 | 203.4 | 171.4 | 178.1 | 404.8 | 370.7 | 191.0 | 386 |
| Non-fraud # | 1848 | 1175 | 2003 | 1692 | 627 | 738 | 862 | 1163 | 2268 | 1178 | 747 | 1255 |
| Non-frauds mean ($) | 323.4 | 343.4 | 306.7 | 299.5 | 313.8 | 302.4 | 286.6 | 280.7 | 292.4 | 297.9 | 278.3 | 300 |
| Non-frauds std ($) | 174.1 | 175.9 | 157.9 | 149.6 | 149.5 | 150.3 | 133.5 | 140.4 | 148.3 | 174.4 | 134.6 | 159 |
| t-value | 11.9 | 6.2 | 4.2 | 10.4 | 4.0 | 8.7 | 4.7 | 8.2 | 9.3 | 5.0 | 3.8 | 7.9 |
| p-value | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

Table 4.H.1: Additional performance metrics for the stacking ensemble model utilizing a probability threshold of 0.353

| Accuracy (95% CI) | Specificity (95% CI) | Precision (95% CI) | Sensitivity (95% CI) |
|---|---|---|---|
| 0.850 (0.836-0.862) | 0.886 (0.867-0.903) | 0.750 (0.720-0.779) | 0.763 (0.733-0.793) |

## Reducing Mislabeling

Modifying the probability threshold not only would change the performance metrics, but also would lead to alterations in predicted overutilization statistics. In what follows, we adjust the probability threshold so as to reduce mislabeling. Subsequently, we recalculate the overutilization statistics and compare them with the results mentioned in the article.

To reduce mislabeling, we adjust the probability threshold to a higher value, resulting in an increased specificity (or a decreased false positive rate). Suppose our target is to achieve a specificity score of 95% or higher (or a false positive rate of 5% or lower). According to Figure 4.H.1, we can identify a probability threshold that maintains the same accuracy level while achieving a very low false positive rate. Specifically, we choose the probability threshold of 0.646. The corresponding performance metrics are presented in Table 4.H.2. The results demonstrate that the new probability threshold leads to higher specificity and precision, albeit at the expense of lower sensitivity. This choice of probability threshold is particularly suitable in situations where the main objective is to minimize false positives.

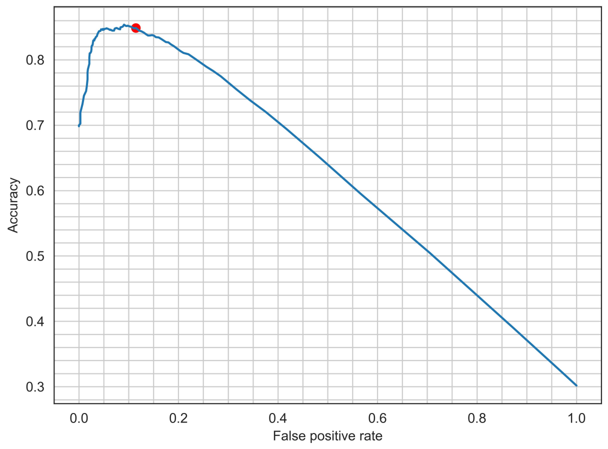


Figure 4.H.1: The Relation between the accuracy and false positive rate. The red dot corresponds to the probability threshold of 0.353.

Table 4.H.2: Performance metrics for the stacking ensemble model utilizing a probability threshold of 0.646

| Accuracy (95% CI) | Specificity (95% CI) | Precision (95% CI) | Sensitivity (95% CI) |
|---|---|---|---|
| 0.847 (0.835-0.860) | 0.952 (0.939-0.965) | 0.854 (0.819- 0.889) | 0.605 (0.5685-0.642) |

With this higher probability threshold, we expect to detect less overutilizer physicians, and therefore the estimated overutilization rate should decrease. Indeed, the estimated nationwide overutilization rate is decreased to 4.2% (from 8.6%), which still falls between the 3-10% fraud rate estimation [137], and the estimated monetary loss is decreased to $168.5 million (from $437.1 million). Additionally, Figure 4.H.2 shows the updated heat map of overutilization rate across the US.

It is intriguing to find that a state like CA, which previously had the highest overutilization rate (14.2%), does not have the highest rate in the new estimations. Instead, its overutilization rate is estimated to be 8.6% whereas the highest rate is 11.1%. The shift in the estimated overutilization rates for CA and many other states indicates that many physicians in these states fall into the category of borderline physicians. This means that the probability of them being overutilizers lies above 35% (previous probability threshold) but below 64% (the new

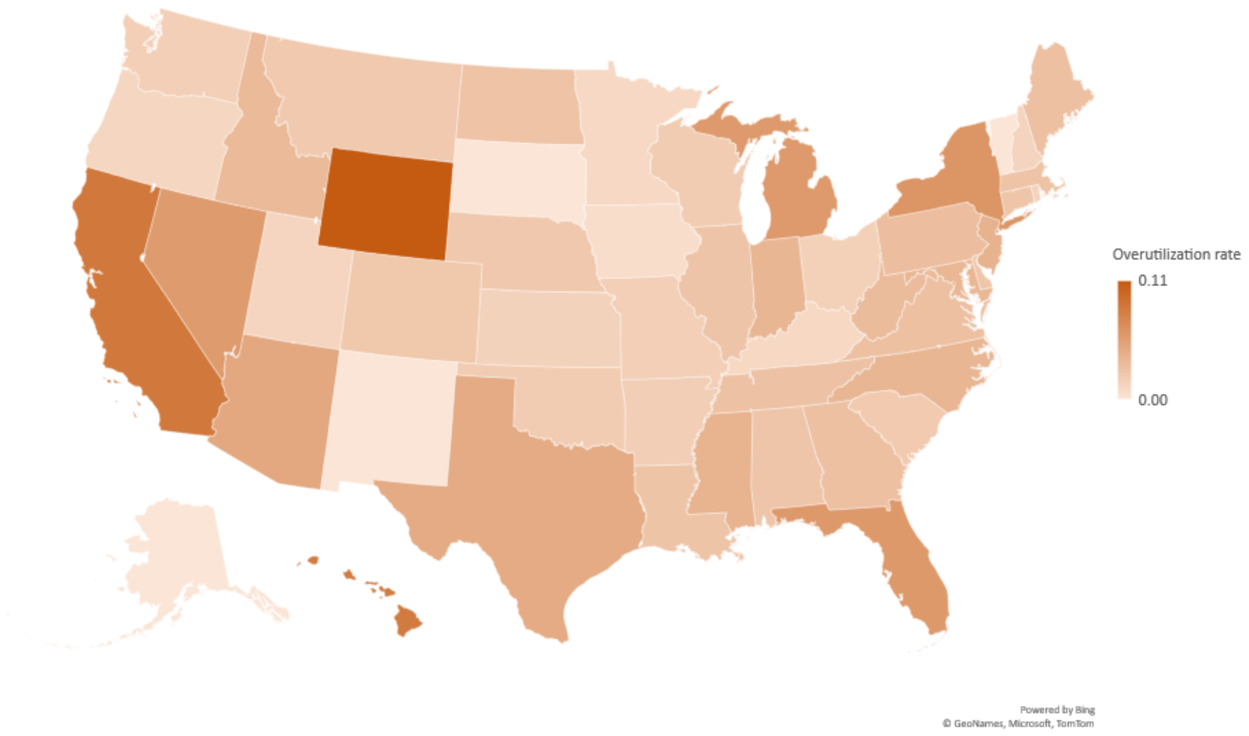


Figure 4.H.2: The heat map showcases the predicted overutilization rates across states in the United States. The rates are estimated using the probability threshold of 0.646.

probability threshold). The presence of a significant number of borderline physicians in these states highlights the need for further investigation into their billing practices.

# Bibliography

[1] Yves Achdou and Jean-Michel Lasry. "Mean field games for modeling crowd motion". In: *Contributions to partial differential equations and applications* (2019), pp. 17–42.

[2] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[3] Takuya Akiba et al. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 2623–2631.

[4] Clémence Alasseur, Imen Ben Taher, and Anis Matoussi. "An extended mean field game for storage in smart grids". In: *Journal of Optimization Theory and Applications* 184 (2020), pp. 644–670.

[5] Donald G Anderson. "Iterative procedures for nonlinear integral equations". In: *Journal of the ACM (JACM)* 12.4 (1965), pp. 547–560.

[6] Keith Ball. "Cube slicing in $\mathbf{R}^n$". In: *Proc. Amer. Math. Soc.* 97.3 (1986), pp. 465–473.

[7] Afonso S. Bandeira, Nicolas Boumal, and Vladislav Voroninski. "On the low-rank approach for semidefinite programs arising in synchronization and community detection". In: *Conference on Learning Theory*. 2016, pp. 361–382.

[8] Richard Bauder and Taghi Khoshgoftaar. "Medicare fraud detection using random forest with class imbalanced big data". In: *2018 IEEE international conference on information reuse and integration (IRI)*. IEEE. 2018, pp. 80–87.

[9] Richard Bauder, Raquel da Rosa, and Taghi Khoshgoftaar. "Identifying medicare provider fraud with unsupervised machine learning". In: *2018 IEEE international conference on information Reuse and integration (IRI)*. IEEE. 2018, pp. 285–292.

[10] Richard A Bauder and Taghi M Khoshgoftaar. "A probabilistic programming approach for outlier detection in healthcare claims". In: *2016 15th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2016, pp. 347–354.

[11] Richard A Bauder and Taghi M Khoshgoftaar. "Medicare fraud detection using machine learning methods". In: *2017 16th IEEE international conference on machine learning and applications (ICMLA)*. IEEE. 2017, pp. 858–865.

[12] Richard A Bauder and Taghi M Khoshgoftaar. “The effects of varying class distribution on learner behavior for medicare fraud detection with imbalanced big data”. In: *Health information science and systems* 6.1 (2018), pp. 1–14.

[13] Fabrice Baudoin, Martin Hairer, and Josef Teichmann. “Ornstein-Uhlenbeck processes on Lie groups”. In: *J. Funct. Anal.* 255.4 (2008), pp. 877–890.

[14] Alain Bensoussan, Jens Frehse, Phillip Yam, et al. *Mean field games and mean field type control theory.* Vol. 101. Springer, 2013.

[15] Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. “Global optimality of local search for low rank matrix recovery”. In: *Advances in Neural Information Processing Systems.* 2016, pp. 3873–3881.

[16] L Karl Branting et al. “Graph analytics for healthcare fraud risk estimation”. In: *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM).* IEEE. 2016, pp. 845–851.

[17] Jason Brownlee. “Nested Cross-Validation for Machine Learning with Python”. In: *https://machinelearningmastery.com/nested-cross-validation-for-machine-learning-with-python* (2020).

[18] Sébastien Bubeck. “Convex optimization: Algorithms and complexity”. In: *Foundations and Trends® in Machine Learning* 8.3-4 (2015), pp. 231–357.

[19] Theophile Cabannes et al. “Solving N-player dynamic routing games with congestion: a mean field approach”. In: *arXiv preprint arXiv:2110.11943* (2021).

[20] Emmanuel J. Candès, Xiaodong Li, and Mahdi Soltanolkotabi. “Phase retrieval via Wirtinger flow: theory and algorithms”. In: *IEEE Trans. Inform. Theory* 61.4 (2015), pp. 1985–2007.

[21] Emmanuel J. Candès et al. “Phase retrieval via matrix completion”. In: *SIAM J. Imaging Sci.* 6.1 (2013), pp. 199–225.

[22] Emmanuel J. Candès et al. “Robust principal component analysis?” In: *J. ACM* 58.3 (2011), Art. 11, 37.

[23] René Carmona, François Delarue, et al. *Probabilistic theory of mean field games with applications I-II.* Springer, 2018.

[24] Augustin Cauchy. “Méthode générale pour la résolution des systemes d’équations simultanées”. In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538.

[25] Tianqi Chen and Carlos Guestrin. “Xgboost: A scalable tree boosting system”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining.* 2016, pp. 785–794.

[26] Xi Chen, Simon S. Du, and Xin T. Tong. “On stationary-point hitting time and ergodicity of stochastic gradient Langevin dynamics”. In: *J. Mach. Learn. Res.* 21 (2020), Paper No. 68, 41.

[27] Yi Chen et al. “Accelerating nonconvex learning via replica exchange Langevin diffusion”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[28] Anna Choromanska et al. “The loss surfaces of multilayer networks”. In: *Artificial Intelligence and Statistics*. 2015, pp. 192–204.

[29] Vincent Conitzer et al. “Pacing Equilibrium in First Price Auction Markets”. In: *Management Science* (2022).

[30] Areski Cousin et al. “Mean field games and applications”. In: *Paris-Princeton lectures on mathematical finance 2010* (2011), pp. 205–266.

[31] J. Cubanski. “What’s in Store for Medicare’s Part B Premiums and Deductible in 2016, and Why?” In: *https://www.kff.org/medicare/issue-brief/whats-in-store-for-medicares -part-b-premiums-and-deductible-in-2016-and-why* (2015).

[32] Kai Cui and Heinz Koeppl. “Approximately solving mean field games via entropy-regularized deep reinforcement learning”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 1909–1917.

[33] Ashok Cutkosky and Francesco Orabona. “Momentum-based variance reduction in non-convex sgd”. In: *Advances in neural information processing systems* 32 (2019).

[34] Yann Dauphin et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: (2014). arXiv:1406.2572.

[35] Yann N. Dauphin et al. “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2933–2941.

[36] Burgess Davis. “Reinforced random walk”. In: *Probab. Theory Related Fields* 84.2 (1990), pp. 203–229.

[37] Jing Dong and Xin T. Tong. “Replica exchange for non-convex optimization”. In: *J. Mach. Learn. Res.* 22 (2021), Paper No. 173, 59.

[38] Felix Draxler et al. “Essentially No Barriers in Neural Network Energy Landscape”. In: *International Conference on Machine Learning*. 2018, pp. 1309–1318.

[39] Simon S. Du et al. “Gradient descent can take exponential time to escape saddle points”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 1067–1077.

[40] Simon S. Du et al. “Gradient Descent Learns One-hidden-layer CNN: Don’t be Afraid of Spurious Local Minima”. In: *International Conference on Machine Learning*. 2018, pp. 1339–1348.

[41] Richard G Frank and Tricia Neuman. “Addressing the risk of Medicare trust fund insolvency”. In: *JAMA* 325.4 (2021), pp. 341–342.

[42] David Fridovich-Keil et al. “Efficient iterative linear-quadratic approximations for nonlinear multi-player general-sum differential games”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 1475–1481.

[43] Rong Ge, Chi Jin, and Yi Zheng. “No spurious local minima in nonconvex low rank problems: A unified geometric analysis”. In: *International Conference on Machine Learning*. 2017, pp. 1233–1242.

[44] Rong Ge, Jason D. Lee, and Tengyu Ma. “Learning One-hidden-layer Neural Networks with Landscape Design”. In: *International Conference on Learning Representations*. 2018.

[45] Rong Ge, Jason D. Lee, and Tengyu Ma. “Matrix completion has no spurious local minimum”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2973–2981.

[46] Rong Ge and Tengyu Ma. “On the optimization landscape of tensor decompositions”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3653–3663.

[47] Rong Ge et al. “Escaping from saddle points – online stochastic gradient for tensor decomposition”. In: *Conference on Learning Theory*. 2015, pp. 797–842.

[48] Stuart Geman and Chii-Ruey Hwang. “Diffusions for global optimization”. In: *SIAM J. Control Optim.* 24.5 (1986), pp. 1031–1043.

[49] Saeed Ghadimi and Guanghui Lan. “Stochastic first-and zeroth-order methods for nonconvex stochastic programming”. In: *SIAM Journal on Optimization* 23.4 (2013), pp. 2341–2368.

[50] Xin Guo, Anran Hu, and Junzi Zhang. “MF-OMO: An optimization formulation of mean-field games”. In: *arXiv preprint arXiv:2206.09608* (2022).

[51] Xin Guo et al. “A general framework for learning mean-field games”. In: *arXiv preprint arXiv:2003.06069* (2020).

[52] Xin Guo et al. “Learning mean-field games”. In: *Advances in Neural Information Processing Systems* 32 (2019).

[53] Juho Hamari, Mimmi Sjöklint, and Antti Ukkonen. “The sharing economy: Why people participate in collaborative consumption”. In: *Journal of the association for information science and technology* 67.9 (2016), pp. 2047–2059.

[54] John Hancock and Taghi M Khoshgoftaar. “Medicare fraud detection using catboost”. In: *2020 IEEE 21st international conference on information reuse and integration for data science (IRI)*. IEEE. 2020, pp. 97–103.

[55] John Hancock, Taghi M Khoshgoftaar, and Justin M Johnson. “The Effects of Random Undersampling for Big Data Medicare Fraud Detection”. In: *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE. 2022, pp. 141–146.

[56] John T Hancock and Taghi M Khoshgoftaar. “Gradient boosted decision tree algorithms for medicare fraud detection”. In: *SN Computer Science* 2.4 (2021), p. 268.

[57] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[58] Matthew Herland, Taghi M Khoshgoftaar, and Richard A Bauder. “Big data fraud detection using multiple medicare data sources”. In: *Journal of Big Data* 5.1 (2018), pp. 1–21.

[59] Richard A. Holley, Shigeo Kusuoka, and Daniel W. Stroock. “Asymptotics of the spectral gap with applications to the theory of simulated annealing”. In: *J. Funct. Anal.* 83.2 (1989), pp. 333–347.

[60] Yuanhan Hu et al. “Non-Convex Optimization via Non-Reversible Stochastic Gradient Langevin Dynamics”. In: (2020). arXiv:2004.02823.

[61] Kuang Huang et al. “A game-theoretic framework for autonomous vehicles velocity control: Bridging microscopic differential games and macroscopic mean field games”. In: *arXiv preprint arXiv:1903.06053* (2019).

[62] Minyi Huang, Roland P Malhamé, and Peter E Caines. “Large population stochastic dynamic games: closed-loop McKean-Vlasov systems and the Nash certainty equivalence principle”. In: *Communications in Information & Systems* 6.3 (2006), pp. 221–252.

[63] Katie F Huffman and Gina Upchurch. “The health of older Americans: a primer on Medicare and a local perspective”. In: *Journal of the American Geriatrics Society* 66.1 (2018), pp. 25–32.

[64] Forrest N Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size”. In: *arXiv preprint arXiv:1602.07360* (2016).

[65] Office of Inspector General. “List of Excluded Individuals Entities (LEIE) Database”. In: *https://oig.hhs.gov/exclusions/index.asp* (2023).

[66] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. pmlr. 2015, pp. 448–456.

[67] Prateek Jain et al. “Global Convergence of Non-Convex Gradient Descent for Computing Matrix Squareroot”. In: *Artificial Intelligence and Statistics*. 2017, pp. 479–488.

[68] Seong Hoon Jeong, Ah Reum Kang, and Huy Kang Kim. “Analysis of game bot’s behavioral characteristics in social interaction networks of MMORPG”. In: *ACM SIGCOMM Computer Communication Review* 45.4 (2015), pp. 99–100.

[69] Chi Jin, Praneeth Netrapalli, and Michael I Jordan. “Accelerated gradient descent escapes saddle points faster than gradient descent”. In: *Conference On Learning Theory*. 2018, pp. 1042–1085.

[70] Chi Jin et al. “How to escape saddle points efficiently”. In: *International Conference on Machine Learning*. 2017, pp. 1724–1732.

[71] Chi Jin et al. “On Nonconvex Optimization for Machine Learning: Gradients, Stochasticity, and Saddle Points”. In: *Journal of the ACM* 68.2 (2021), pp. 1–29.

[72] Justin M Johnson and Taghi M Khoshgoftaar. “Deep learning and data sampling with imbalanced big data”. In: *2019 IEEE 20th international conference on information reuse and integration for data science (IRI)*. IEEE. 2019, pp. 175–183.

[73] Justin M Johnson and Taghi M Khoshgoftaar. “Medicare fraud detection using neural networks”. In: *Journal of Big Data* 6.1 (2019), pp. 1–35.

[74] U.S. Department of Justice. “Justice Department Recovers over $3 Billion from False Claims Act.” In: *https://www.justice.gov/opa/pr/justice-department-recovers-over-3-billion-false-claims-act-cases-fiscal-year-2019* (2020).

[75] Kai Cui. “GMFG-learning: Learning graphon mean-field games”. In: *https://github.com/tudkcui/gmfg-learning* last visited 2023 March 25 (2021).

[76] Kenji Kawaguchi. “Deep learning without poor local minima”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 586–594.

[77] Abbas Kazemipour, Brett Larsen, and Shaul Druckmann. “Avoiding Spurious Local Minima in Deep Quadratic Networks”. In: (2019). arXiv:2001.00098.

[78] Sean P Keehan et al. “National Health Expenditure Projections, 2019–28: Expected Rebound In Prices Drives Rising Spending Growth: National health expenditure projections for the period 2019–2028.” In: *Health Affairs* 39.4 (2020), pp. 704–714.

[79] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations*. 2015.

[80] S. Kirkpatrick, Jr. C. D. Gelatt, and M. P. Vecchi. “Optimization by simulated annealing”. In: *Science* 220.4598 (1983), pp. 671–680.

[81] Arman C Kizilkale, Rabih Salhab, and Roland P Malhamé. “An integral control formulation of mean field game based large scale coordination of loads in smart grids”. In: *Automatica* 100 (2019), pp. 312–322.

[82] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[83] Marc Lanctot et al. “OpenSpiel: A Framework for Reinforcement Learning in Games”. In: *CoRR* abs/1908.09453 (2019). arXiv: `1908.09453 [cs.LG]`.

[84] Jean-Michel Lasry and Pierre-Louis Lions. “Mean field games”. In: *Japanese Journal of Mathematics* 2.1 (2007), pp. 229–260.

[85] Mathieu Laurière et al. “Learning mean field games: A survey”. In: *arXiv preprint arXiv:2205.12944* (2022).

[86] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[87] Jason D. Lee et al. “Gradient descent only converges to minimizers”. In: *Conference on Learning Theory*. 2016, pp. 1246–1257.

[88] Charles-Albert Lehalle and Charafeddine Mouzouni. “A mean field game of portfolio trading and its consequences on perceived correlations”. In: *arXiv preprint arXiv:1902.09606* (2019).

[89] Zhong Li et al. “On the Curse of Memory in Recurrent Neural Networks: Approximation and Optimization Analysis”. In: *International Conference on Learning Representations*. 2021.

[90] Shiyu Liang et al. “Adding one neuron can eliminate all bad local minima”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 4350–4360.

[91] Scott M Lundberg and Su-In Lee. “A unified approach to interpreting model predictions”. In: *Advances in neural information processing systems* 30 (2017).

[92] Mansour Zoubeirou A Mayaki and Michel Riveill. “Multiple Inputs Neural Networks for Medicare fraud Detection”. In: *arXiv preprint arXiv:2203.05842* (2022).

[93] Centers for Medicare and Medicaid Services. “2020 Medicare Parts A & B Premiums and Deductibles”. In: *https://www.cms.gov/newsroom/fact-sheets/2020-medicare-parts-b-premiums-and-deductibles* (2019).

[94] Centers for Medicare and Medicaid Services. “NHE fact sheet”. In: *https://www.cms.gov/research-statistics-data-and-systems/statistics-trends-and-reports/nationalhealthexpenddata/nhe-fact-sheet* (2023).

[95] Centers for Medicare & Medicaid Services. “CMS Website”. In: *https://www.cms.gov* (2023).

[96] Song Mei, Yu Bai, and Andrea Montanari. “The landscape of empirical risk for nonconvex losses”. In: *Ann. Statist.* 46.6A (2018), pp. 2747–2774.

[97] Song Mei et al. “Solving SDPs for synchronization and MaxCut problems via the Grothendieck inequality”. In: *Conference on Learning Theory*. 2017, pp. 1476–1515.

[98] Georg Menz et al. “Ergodicity of the infinite swapping algorithm at low temperature”. In: (2018). arXiv:1811.10174.

[99] François Mériaux, Vineeth Varma, and Samson Lasaulce. “Mean field energy games in wireless networks”. In: *2012 conference record of the forty sixth Asilomar conference on signals, systems and computers (ASILOMAR)*. IEEE. 2012, pp. 671–675.

[100] Laurent Miclo. “Recuit simulé sur $\mathbf{R}^n$. Étude de l’évolution de l’énergie libre”. In: *Ann. Inst. H. Poincaré Probab. Statist.* 28.2 (1992), pp. 235–266.

[101] Pierre Monmarché. “Hypocoercivity in metastable settings and kinetic simulated annealing”. In: *Probab. Theory Related Fields* 172.3-4 (2018), pp. 1215–1248.

[102] National Health Care Anti-Fraud Association. “The Challenge of Health Care Fraud”. In: *https://www.nhcaa.org/tools-insights/about-health-care-fraud/the-challenge-of-health-care-fraud* (2023).

[103] Arvind Neelakantan et al. "Adding gradient noise improves learning for very deep networks". In: (2015). arXiv:1511.06807.

[104] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course.* Vol. 87. Applied Optimization. Kluwer Academic Publishers, Boston, MA, 2004, pp. xviii+236.

[105] Yurii E Nesterov. "A method for solving the convex programming problem with convergence rate $O(1/k^2)$". In: *Dokl. Akad. Nauk SSSR.* Vol. 269. 1983, pp. 543–547.

[106] Quynh Nguyen and Matthias Hein. "The loss surface of deep and wide neural networks". In: *International Conference on Machine Learning.* 2017, pp. 2603–2612.

[107] Dohyung Park et al. "Non-square matrix sensing without spurious local minima via the Burer-Monteiro approach". In: *Artificial Intelligence and Statistics.* 2017, pp. 65–74.

[108] Ilya Pavlyukevich. "Lévy flights, non-local search and simulated annealing". In: *Journal of Computational Physics* 226.2 (2007), pp. 1830–1844.

[109] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[110] Luca Peliti and Luciano Pietronero. "Random walks with memory". In: *La Rivista del Nuovo Cimento* 10.6 (1987), pp. 1–33.

[111] Robin Pemantle. "Vertex-reinforced random walk". In: *Probab. Theory Related Fields* 92.1 (1992), pp. 117–136.

[112] Julien Perolat et al. "Scaling up mean field games with online mirror descent". In: *arXiv preprint arXiv:2103.00623* (2021).

[113] Sarah Perrin et al. "Fictitious play for mean field games: Continuous time analysis and applications". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13199–13213.

[114] Sarah Perrin et al. "Mean Field Games Flock! The Reinforcement Learning Way". In: *arXiv preprint arXiv:2105.07933* (2021).

[115] Boris T Polyak. "Some methods of speeding up the convergence of iteration methods". In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17.

[116] Foster Provost. "Machine learning from imbalanced data sets 101". In: (2008).

[117] Maxim Raginsky, Alexander Rakhlin, and Matus Telgarsky. "Non-convex learning via stochastic gradient Langevin dynamics: a nonasymptotic analysis". In: *Conference On Learning Theory.* 2017, pp. 1674–1703.

[118] Sashank Reddi et al. "A generic approach for escaping saddle points". In: *International Conference on Artificial Intelligence and Statistics.* 2018, pp. 1233–1242.

[119] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. "On the convergence of Adam and beyond". In: *International Conference on Learning Representations.* 2018.

[120] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors". In: *Nature* 323.6088 (1986), pp. 533–536.

[121] Saad Sadiq and Mei-Ling Shyu. “Cascaded propensity matched fraud miner: Detecting anomalies in medicare big data”. In: *Journal of Innovative Technology* 1.1 (2019), pp. 51–61.

[122] Naci Saldi, Tamer Basar, and Maxim Raginsky. “Markov Nash equilibria in mean-field games with discounted cost”. In: *SIAM Journal on Control and Optimization* 56.6 (2018), pp. 4256–4287.

[123] Maziar Sanjabi et al. “When Does Non-Orthogonal Tensor Decomposition Have No Spurious Local Minima?” In: (2019). arXiv:1911.09815.

[124] Centers For Medicare & Medicaid Services. “Medicare Provider Utilization and Payment Data”. In: *https://data.cms.gov/provider-summary-by-type-of-service* (2023).

[125] Centers For Medicare & Medicaid Services. “Medicare Provider Utilization and Payment Data: Medicare Physician & Other Practitioners - by Provider”. In: *https://data.cms.gov/provider-summary-by-type-of-service/medicare-physician-other-practitioners/medicare-physician-other-practitioners-by-provider* (2023).

[126] Centers For Medicare & Medicaid Services. “Medicare Provider Utilization and Payment Data: Medicare Physician & Other Practitioners - by Provider and Service”. In: *https://data.cms.gov/provider-summary-by-type-of-service/medicare-inpatient-hospitals/medicare-inpatient-hospitals-by-provider-and-service* (2023).

[127] Ju Sun, Qing Qu, and John Wright. “A geometric analysis of phase retrieval”. In: *Found. Comput. Math.* 18.5 (2018), pp. 1131–1198.

[128] Ju Sun, Qing Qu, and John Wright. “Complete dictionary recovery over the sphere I: Overview and the geometric picture”. In: *IEEE Trans. Inform. Theory* 63.2 (2017), pp. 853–884.

[129] Grzegorz Swirszcz, Wojciech Marian Czarnecki, and Razvan Pascanu. “Local minima in training of deep networks”. In: *https://openreview.net/pdf?id=Syoiqwcxx* (2016).

[130] Wenpin Tang and Xun Yu Zhou. “Simulated annealing from continuum to discretization: a convergence analysis via the Eyring–Kramers law”. In: (2021). arXiv:2102.02339.

[131] Wenpin Tang and Xun Yu Zhou. “Tail probability estimates of continuous-time simulated annealing processes”. In: *Numer. Algebra Control Optim.* 13.3-4 (2023), pp. 473–485.

[132] Pierre Tarrès. “Vertex-reinforced random walk on $\mathbb{Z}$ eventually gets stuck on five points”. In: *Ann. Probab.* 32.3B (2004), pp. 2650–2701.

[133] Thomas J. Sargent, et al. “QuantEcon.py: A high performance, open source Python code library for economics”. In: *https://github.com/QuantEcon/QuantEcon.py* last visited 2023 March 25 (2013).

[134] Bálint Tóth. ““True” self-avoiding walks with generalized bond repulsion on $\mathbf{Z}$”. In: *J. Statist. Phys.* 77.1-2 (1994), pp. 17–33.

[135] Bálint Tóth. “Generalized Ray-Knight theory and limit theorems for self-interacting random walks on $\mathbf{Z}^1$”. In: *Ann. Probab.* 24.3 (1996), pp. 1324–1367.

[136] Bálint Tóth. “The “true” self-avoiding walk with bond repulsion on $\mathbf{Z}$: limit theorems”. In: *Ann. Probab.* 23.4 (1995), pp. 1523–1556.

[137] Thad Trousdale. “Health care fraud & the FBI”. In: *Missouri medicine* 109.2 (2012), p. 102.

[138] Sudhir Varma and Richard Simon. “Bias in error estimation when using cross-validation for model selection”. In: *BMC bioinformatics* 7.1 (2006), pp. 1–8.

[139] Luca Venturi, Afonso S. Bandeira, and Joan Bruna. “Spurious valleys in one-hidden-layer neural network optimization landscapes”. In: *J. Mach. Learn. Res.* 20 (2019), Paper No. 133, 34.

[140] Vince Knight. “Nashpy: A Python library for 2 player games”. In: *https://github.com/drvinceknight/Nashpy* last visited 2023 March 25 (2016).

[141] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272.

[142] Stanislav Volkov. “Phase transition in vertex-reinforced random walks on $\mathbb{Z}$ with non-linear reinforcement”. In: *J. Theoret. Probab.* 19.3 (2006), pp. 691–700.

[143] Jun-Kun Wang, Chi-Heng Lin, and Jacob Abernethy. “Escaping Saddle Points Faster with Stochastic Momentum”. In: *International Conference on Learning Representations.* 2019.

[144] Wilson Jallet. “Entropic-MFG: Entropic variational mean-field games”. In: *https://github.com/ManifoldFR/entropic-mfg* last visited 2023 March 25 (2020).

[145] David H Wolpert. “Stacked generalization”. In: *Neural networks* 5.2 (1992), pp. 241–259.

[146] WSJ. “WSJ Medicare Unmasked reference”. In: *https://graphics.wsj.com/medicare-billing* (2023).

[147] Chenwei Wu, Jiajun Luo, and Jason D. Lee. “No Spurious Local Minima in a Two Hidden Unit ReLU Network”. In: *https://openreview.net/forum?id=B14uJzW0b* (2018).

[148] Xin-She Yang and Suash Deb. “Cuckoo search via Lévy flights”. In: *2009 World congress on nature & biologically inspired computing (NaBIC).* 2009, pp. 210–214.

[149] Jiahe Yao et al. “Medicare fraud detection using wtbagging algorithm”. In: *2021 7th International Conference on Computer and Communications (ICCC).* IEEE. 2021, pp. 1515–1519.

[150] Junzi Zhang, Brendan O’Donoghue, and Stephen Boyd. “Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations”. In: *SIAM Journal on Optimization* 30.4 (2020), pp. 3170–3197.

[151] Mo Zhou et al. "Toward Understanding the Importance of Noise in Training Neural Networks". In: *International Conference on Machine Learning*. 2019.

[152] Juntang Zhuang et al. "Adabelief optimizer: Adapting stepsizes by the belief in observed gradients". In: *Advances in neural information processing systems* 33 (2020), pp. 18795–18806.