ABSTRACT

PRIVATE AND FEDERATED DEEP LEARNING: SYSTEM, THEORY, AND APPLICATIONS FOR SOCIAL GOOD

by
Han Hu

During the past decade, drug abuse continues to accelerate towards becoming the most severe public health problem in the United States. The ability to detect drug-abuse risk behavior at a population scale, such as among the population of Twitter users, can help to monitor the trend of drug-abuse incidents. However, traditional methods do not effectively detect drug-abuse risk behavior in tweets, mainly due to the sparsity of such tweets and the noisy nature of tweets. In the first part of this dissertation work, the task of classifying tweets as containing drug-abuse risk behavior or not, is studied. Millions of public tweets were collected through the Twitter API, and a large human labeled dataset with both expert labels and crowd-sourced (through the Amazon Mechanical Turks) labels was built. Three papers on this topic were published: The first work leveraged large quantities of unlabeled tweets with self-taught deep learning (DL); In the second work, the method of mitigating the imbalance of tweets' classes through the ensemble of DL models was proposed. Results on the testing dataset showed improved performance over traditional and recent methods. Statistical analysis on the results of applying the model on 3-million tweets also yield interesting and meaningful results. Based on the detection model, a demo system was built, which allows the geographical and various statistical information of drug abuse indication tweets to be viewed on live interactive maps.

The development of the drug abuse detection models revealed the importance of privacy preservation in DL. Related works have demonstrated that the privacy of the training data of a DL model can be exploited through either reconstruction attack or membership-inference attack. Thus, due to the sensitive nature of the drug abuse detection model, the privacy of the training data has to be rigorously protected before the model can be

made public. The goal of the first work in this direction was to develop a novel mechanism for preserving differential privacy (DP), such that the privacy budget consumption is independent of the training steps and grants the ability to adaptively inject noise according to the importance of features to improve the model utility. Then, in the second work, the aim was to develop a scalable DP preserving algorithm for deep neural networks, with certified robustness to adversarial examples. The robustness bound was strengthened by a novel adversarial objective function, and by injecting noise into both input and latent space. For the first time, a novel stochastic batch training that allows the training of the DP models to be parallelized, was proposed. In the third work along this line, the goal was to preserve DP in the setting of lifelong learning (L2M), given the more challenging privacy risk that the L2M posts. A scalable and heterogeneous algorithm was proposed and implemented, which allows the efficient training and the continuous releasing of new versions of L2M models without affecting the DP protection.

Despite that DP-DL can provide provable privacy protection, another aspect of privacy protection is to protect the data itself. In the foreseeable future, more rigorous data privacy regulations will be widely implemented, which promotes the use of federated learning (FL). In the third part of the dissertation work, the FLSys, a prototype mobile-cloud federated deep learning system was designed and implemented. By utilizing modern cloud architecture, the FLSys is designed to achieve energy efficiency, tolerance failure tolerance, and scalability. To demonstrate the capability of the FLSys, the task of mobile human activity recognition, which aims at predicting human activities with smartphone sensors, was selected. For model developing purpose, two data collection campaigns were launched to collect human activity data through smartphone sensors in the wild from hundreds of volunteers. A simple yet effective way of data augmentation to combat the non-I.I.D (Independent and Identically Distributed) issue that plagues FL was proposed.

PRIVATE AND FEDERATED DEEP LEARNING: SYSTEM, THEORY, AND APPLICATIONS FOR SOCIAL GOOD

by
Han Hu

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Information Systems

Department of Informatics

December 2021

APPROVAL PAGE

PRIVATE AND FEDERATED DEEP LEARNING:
SYSTEM, THEORY, AND APPLICATIONS FOR SOCIAL GOOD

Han Hu

Dr. Nhathai Phan, Dissertation Advisor Date
Assistant Professor of Informatics, New Jersey Institute of Technology

---

Dr. Frank A. Biocca, Committee Member Date
Chair, Professor of Informatics, New Jersey Institute of Technology

---

Dr. Cristian M. Borcea, Committee Member Date
Associate Dean, Professor of Computer Science, New Jersey Institute of Technology

---

Dr. Dejing Dou, Committee Member Date
Professor of Computer and Information Science, University of Oregon in Eugene, Oregon

---

Dr. Ruoming Jin, Committee Member Date
Professor of Computer Science, Kent State University in Kent, Ohio

BIOGRAPHICAL SKETCH

Author:	Han Hu
Degree:	Doctor of Philosophy
Date:	December 2021

Undergraduate and Graduate Education:

- Doctor of Philosophy in Information Systems,
  New Jersey Institute of Technology, Newark, New Jersey, 2021
- Bachelor of Science in Computer Science,
  Zhejiang University of Technology, Hangzhou, China, 2015

Major: Information Systems

Presentations and Publications:

H. Hu, N. Phan, X. Ye, R. Jin, K. Ding, D. Dou, and H. Vo, DrugTracker: a community-focused drug abuse monitoring and supporting system using social media and geospatial data (Demo Paper). In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp 564-567, 2019.

H. Hu, N. Phan, J. Geller, S. Iezzi, H. Vo, D. Dou, and S. A. Chun, An ensemble deep learning model for drug abuse detection in sparse Twitter-Sphere. In *Proceedings of the 17th World Congress of Medical and Health Informatics*, pp 163-167, 2019.

H. Hu, N. Phan, J. Geller, S. A. Chun, R. Jin, K. Ding, D. Kenne, and D. Dou, An insight analysis and detection of drug abuse risk behavior on Twitter with self-taught deep learning. In *Computational Social Networks*, vol. 6, no. 10, 2019.

H. Hu, N. Phan, J. Geller, H. Vo, B. Manasi, X. Huang, S. Di Lorio, T. Dinh, and S. A. Chun, Deep self-taught learning for detecting drug abuse risk behavior in Tweets, in *Proceedings of the 7th International Conference on Computational Social Networks*, 2018. doi: 10.1007%2F978-3-030-04648-4_28 pp 330–342.

H. Hu, P. Moturu, K. Dharan, J. Geller, S. Di Lorio, N. Phan, H. Vo, and S. A. Chun, Deep learning model for classifying drug abuse risk behavior in Tweets, In *Proceedings of the IEEE International Conference on Healthcare Informatics*, 2018. doi: 10.1109/ICHI.2018.00066 pp 386–387.

H. Hu, Y. Fang, R. Jin, W. Xiong, X. Qian, D. Dou, and N. Phan, Recursive structure similarity: a novel algorithm for graph clustering. In *Proceedings of the 30th IEEE International Conference on Tools with Artificial Intelligence*, pp 395–400, 2018.

N. Phan, M. T. Thai, H. Hu, R. Jin, T. Sun, and D. Dou, Scalable differential privacy with certified robustness in adversarial learning. In *Proceedings of the 37th International Conference on Machine Learning*, pp 7683-7694, 2020.

P. Lai, N. Phan, H. Hu, A. Badeti, D. Newman, and D. Dou, Ontology-based interpretable machine learning for textual data. In *Proceedings of the International Joint Conference on Neural Networks*, pp 1-10, 2020.

W. Liu, X. Ye, N. Phan, and H. Hu, Scalable self-taught deep-embedded learning framework for drug abuse spatial behaviors detection. In *Proceedings of the 8th International Conference on Computational Data & Social Networks*, pp 223-228, 2019.

N. Phan, X. Wu, H. Hu, and D. Dou, Adaptive laplace mechanism: differential privacy preservation in deep learning. In *Proceedings of the IEEE International Conference on Data Mining*, pp 385-394, 2017.

P. Lia, N. Phan, D. Newman, H. Hu, A. Badeti, and D. Dou, Ontology-based interpretable machine learning with learnable anchors, In *Proceedings of the Knowledge Representation & Reasoning Meets Machine Learning (KR2ML) Workshop at NeurIPS'19*, 2019.

N. Phan, H. Hu, R. Jin, A. Chen, M. Thai, Consistently bounded differential privacy in lifelong learning, Under Review.

H. Hu*, X. Jiang*, V. D. Mayyuri, A. Chen, D. M. Shila, A. Larmuseau, R. Jin, C. Borcea, N. Phan, FLSys: toward an open ecosystem for federated learning mobile apps, *IEEE Transactions on Mobile Computing* Under Review.

至我敬爱的家人们，感谢你们的鼓励与付出。
This dissertation is dedicated to my family for the inspiring, encouragement, and support.

人生是一场长跑。
Life is a marathon, it's not a sprint.

–Phillip C. McGraw

ACKNOWLEDGMENT

First and foremost, I would like to express my sincere gratitude to my advisor Nhathai Phan for the support of my Ph.D. study without reservation. Thanks to his patience, motivation, and immense knowledge, I was able to pick up my research subject quickly. Without his vision and guidance, this dissertation would not be possible. He is an extremely kind, caring, and supportive advisor that I could not have asked for more. I would also thank Songhua Xu for inspiring me to pursue the Ph.D. degree and admitting me to this Ph.D. program.

Secondly, my gratitude also goes to Professor Frank A. Biocca, Cristian Borcea, Shaohua Wang, and Dejing Dou, for kindly serving on my dissertation committee. Their insightful inputs greatly helped both my research effort and the writing of this dissertation.

Then, I must thank Professor Y.F. Brook Wu, James Geller, Ruoming Jin, Xinyue Ye, Yixing Fang, Soon Ae Chun, and My T. Thai, who guided me when I am at lost, assisted me when I need help, and collaborated with me in various papers. I also would like to thank all my fellow Ph.D. students for sharing both the struggles and the joys of the Ph.D. life.

I would like to thanks the Department of Informatics for granting the position of Teaching Assistant that funds my Ph.D. studies. This dissertation work is also supported in part by National Science Foundation grants CNS-1935928, CNS-1850094, CNS-1747798, and CNS-1650587.

Last but not least, I want to thank my uncle's family for supporting me locally in the United States. I also want to thank my roommates for being the best roommates I could have met.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter Page

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Overview

In this dissertation, a series of work on three distinctive yet interconnected fields will be presented. In the first part of the dissertation, our aim is to provide an improved method of monitoring the on-going drug abuse epidemic through the lens of online social media, utilizing the state-of-the-art AI techniques. In the first work, we demonstrate the feasibility and the improved performance of a trained neural networks in identifying tweets, a type of short posts on the online social media platform Twitter, that contains drug abuse-related contents. The training data is constructed by manually identifying tweets, which are automatically cultivated with drug abuse-related keywords, with drug abuse contents. Then in the second work, we provide improved detection model that utilizes the large quantity of unlabeled tweets through self-training mechanism. In addition, we run the trained detection model on a large quantity of tweets (i.e., 3 million tweets) and preformed quantitative and qualitative analysis over the detection results. The analysis not only shows that the detection model is effective, but also reveals interesting properties that were not explored by previous work. Given the fact that a significant portion of tweets are geo-tagged, it makes sense to explore the analysis results in conjunction with their geographical proprieties. Aiming to extend the impact of this study, and to provide a tool for inspecting the geographical proprieties of the monitoring results in a visual and interactive way, we implement and demonstrate a web-based data visualization tool in the third work.

The great potential of monitoring the drug abuse epidemic through online social media is explored and demonstrated in the first part of this dissertation. However, a significant roadblock of privacy issue prevents the work to be wildly and openly adopted. With the modern standard of privacy, any model to be released that is trained with private or individually identifiable data, has to employ strict privacy protection methods. Given the individual identifiable nature of tweets and the sensitive nature of the drug abuse-related topic, strict and rigorous privacy protection will be required for the work in the first part. Thus, in the second part of this dissertation, we switch the course to the developing of more advanced privacy protection methods for deep learning. Our aim here is to advance both the empirical and theoretical aspects of the differential privacy (DP) mechanism, a promising privacy protection mechanism that is originated from the field of database but is quickly found its place in deep learning. Three pieces of work are presented in this part of the dissertation. In the fourth work, we propose a novel Adaptive Laplace Mechanism (AdLM) for differential privacy protection in deep learning. Two key features set the AdLM apart from existing mechanism: (1) The privacy budget consumed by AdLM is totally independent of the number of training epochs; and (2) The amount of noise that the AdLM injects to different weight depends on the importance of that weight. These features mean that the model will be able to have better utility while having the same level of privacy protection, when compared with existing methods. Then, in the fifth work, we propose a more advanced DP mechanism that: (1) Works together with adversarial training and addresses the previously unattended aspect of DP; (2) The provided DP protection is bounded with certified robustness; and (3) Allows fully parallelized training of DP deep learning models. And in the sixth work, we propose novel DP mechanism for Lifelong Learning (L2M). L2M has great potential to be used in the drug abuse detection tasks in order to capture the ever changing trends and expressions of drug abuse-related topics.

Here is yet another privacy protection method that has great potential, the Federated Learning (FL). FL means to train the models at where the data is collected. In the mobile computing setting as an example, it means to use the data collected on each smartphone, and the ever growing mobile computation power, to jointly train a model, instead of uploading the collected data and conducting centralized training. Sharing gradients or model parameters that can be protected by DP further protects privacy of each user. When the epidemic of COVID-19 hits, we were presented an unique opportunity to conduct a study that collects sensor data from smartphones for studying human behaviors and mental health. We also see this study an opportunity to pursue better privacy protection techniques by working on FL, in conjunction with DP. Thus, we step into the track of FL and make our first work in the field. In this work, we propose and implemented a prototype FL framework, which is a first working prototype in the literature that employs application-system co-design approach. Differ from existing proprietary FL system of the Gboard (Google's keyboard app), our design propose an architecture with the following three aims: (1) To provide easy integration of FL with a wide variety of apps, by sharing the FL framework among different apps that want to use FL; (2) To provide low overhead and high efficiency for having many FL models co-exist on one smartphone; and (3) To provide users a place to see and to control the behaviors of different FL models, with unified user experiences.

Thus, this dissertation consists of the aforementioned three tracks. The following introductions will give detailed introduction, background, related works, and contribution of each individual work.

1.2 Drug Abuse Detection and Analysis in Online Social Media

1.2.1 Background

Misuse and abuse of prescription drugs and of illicit drugs have been major public health problems in the United States for decades. A "Public Health Emergency" declared in 2017 [1] and several official surveys [2] all show that the problem has been getting worse in recent years. For example, the most recent reports from the National Survey on Drug Use and Health (NSDUH) [2] estimate that 10.6% of the total population of people ages 12 years and older (i.e., about 28.6 million people) have misused illicit drugs in 2016, which represents an increase of 0.5% over 2015. According to the Centers for Disease Control and Prevention (CDC), opioid drugs were involved in 42,249 known deaths in 2016 nationwide

[3]. In addition, the number of heroin-related deaths has been increasing sharply over five years and has surpassed the number of firearm homicides in 2015 [4]. The emerging new problems, such as the epidemic of illicitly manufactured fentanyl (IMF) [5], marijuana-related traffic accidents [6], and marijuana use among adolescents [7] are posing further increasing threats to public health.

In April 2017, the Department of Health and Human Services announced their "Opioid Strategy" to battle the country's drug abuse crisis [1]. In the Opioid Strategy, one of the major aims is to strengthen public health data collection, in order to inform a timeliness public health response as the epidemic evolves. Given its 100 million daily active users and 500 million daily *tweets* [8] (messages posted by Twitter users), Twitter has been used as a sufficient and reliable data source for many detection tasks, including epidemiology [9] and public health [10, 11, 12, 13, 14, 15], at the population scale, in a real-time manner. Motivated by these facts and the urgent needs, our goal in this dissertation is to develop a large-scale computational system to detect drug abuse risk behaviors via Twitter sphere. Twitter is a popular social media platform that has 100 million daily active users and 500 million daily tweets [8] (messages posted by Twitter users), most of which are publicly accessible, on a wide range of topics.

Several studies [12, 16, 17, 18, 15, 19] have explored the detection of prescription drug abuse on Twitter. Still, the current state-of-the-art approaches and systems are limited in terms of scales and accuracy. They typically applied keyword-based approaches to collect tweets explicitly mentioning specific drug names, such as Adderall, Oxycodone, Quetiapine, Metformin, Cocaine, marijuana, weed, meth, tranquilizer, etc. [12, 17, 15, 19]. However, that may not reflect the actual distribution of drug abuse risk behaviors on online social media, since: (1) The expressions of drug abuse risk behaviors are often vague, in comparison to common topics, i.e., a lot of slang is used; and (2) Relying on only keyword-based approaches is susceptible to lexical ambiguity in natural language [14]. In addition, the drug abuse risk behavior Twitter data is very imbalanced, i.e., dominated by non-drug abuse risk behavior tweets, such as drug-related news, social discussions, reports, advertisements, etc. The limited availability of annotated tweets makes it even more challenging to distinguish drug abuse risk behaviors from drug-related tweets. Yet, existing approaches [12, 17, 15, 19] have not been designed to address these challenging issues for drug abuse risk behavior detection on online social media.

1.2.2 Traditional Drug Abuse Monitoring Systems

Traditionally, drug abuse activities and trends are monitored through both large-scale surveys, such as NSDUH [2] and the Monitoring the Future project [20]; and reporting systems, such as the FDA MedWatch program [21], the National Poisoning Data System [22], the Drug Abuse Warning Network (DAWN) [23]. The results derived from these surveys [24], clearly show that there is an epidemic of drug abuse across the United States. Generally these systems and surveys are considered as trustworthy sources for getting the general picture of the drug abuse epidemic. Nonetheless, a recent report [25] states that the estimated number of deaths due to prescription drugs could be inflated due to the difficulties in determining whether a drug is obtained by prescription or not. We assert that the ambiguities highlighted in this new report raise questions about the reliability of the earlier surveys, and thus, such a report illustrates the potential value of social media-based studies.

1.2.3 Social Media-Based Analytical Studies

In recent years, increasing number of studies utilize online social media data to find trends, preform estimations, and conduct mass monitoring. Many of the existing studies were focusing on the quantitative analysis utilizing data from online social media. Several studies found positive correlations between Twitter data and real world data. Chary et al. [26] performed semantic analysis on 3.6 million tweets with 5% labeled and found significant agreement with the NSDUH data. Hanson et al. [17] conducted a quantitative analysis on 213,633 tweets discussing Adderall, and found positive geo-temporal correlations. Another study from their team [16] focused on how possible drug-abusers interact with and influence others in online social circles. The results showed that strong correlation could be found: (1) Between the amount of interaction about prescription drugs and a level of abusiveness shown by the network; and (2) Between the types of drugs mentioned by the index user and his or her network. Shutler et al. [19] performed a qualitative analysis of six prescription opioids, i.e., Percocet, Percs, OxyContin, Oxys, Vicodin, and Hydros. Tweets were collected with exact word matching and manual classification. Their primary goal was to identify the key terms used in tweets that likely indicate drug abuse. They found that the use of Oxys, Percs and OxyContin was common among the tweets where there were positive indications of abuse. Meng et al. [27] used traditional text and sentiment analysis methods to investigate substance use patterns and underage use of substance, and the association between demographic data and these patterns. Ding et al. [28] investigated the correlation between substance (tobacco, alcohol, and drug) use disorders and words in Facebook users' "Status Updates" and "Likes". Their results showing word patterns are different between users who have substance us disorder and users who do not have. McNaughton et al. [18] measured online endorsement of prescription opioid abuse by developing an integrative metric through the lens of Internet communities. Simpson et al. [29] demonstrated an attempt to identify emerging drug terms using NLP techniques. Furthermore, Twitter and social media have been shown to be reliable sources in analyzing drug abuse and public health-related topics, such as cigarette smoking [10, 14], alcohol use [13], and even cardiac arrest [11].

1.2.4 Studies Utilizing Social Media Data for Classification Tasks

There also have been studies that focused on designing machine learning models to preform classification on online social media posts. Katsuki et al. [30] trained SVM on a dataset of 1,000 tweets for classification of tweets for relevance and favorability of online drug sales. Coloma et al. [31] illustrated the potential of social media in drug safety surveillance with two case studies multiple online social media platforms. Sarker et al. [15] proposed a supervised classification model, in which different features such as n-grams, abuse-indicating terms, slang terms, synonyms, etc., were extracted from manually annotated tweets. Then, these features were used to train traditional machine learning models to classify drug abuse tweets and non-abuse tweets. Recently, many work, including one of our work [32], explored the use of more advanced deep learning models for drug-related classification tasks on online social media. Following our work, Kong et al. [33] proposed deep learning model that utilizes geographical prior information as input features. Chary et al. [12] discussed how to use AI models to extract content useful for purposes of toxicovigilance from social media, such as Facebook, Twitter, and Google+. Weissenbacher et al. [34] proposed deep neural network based model to detect drug name mentions in tweets. Mahata et al. [35] proposed an ensemble CNN model to classify tweets from three classes, i.e., personal medication intakes, possible personal medication intake, and non-intake. Work have also been done in perspectives other than content-based analysis and classification. Zhang et al. [36] proposed a complex schema, which models all possible interactions between users and posts, for automatic detection of drug abusers on Twitter. Li et al. [37] evaluated deep learning models against traditional machine learning models on the task of detecting illicit drug dealers on Instagram.

1.2.5 In Attempt to Scale the Models Up

Although existing studies have shown promising approaches toward the detecting of drug-related posts and information on popular online social media platforms, such as Twitter and Instagram, their limitations can be identified as: (1) Limited in scale, as the methods proposed in many studies do not scale well, or rely on larger manually annotated training dataset for higher performance; (2) Limited in scope, as most studies focus on a small group of drugs; and (3) Limited in performance, as many methods use traditional machine learning models. In this dissertation, we propose a novel deep self-taught learning system to leverage a huge number of unlabeled tweets. Self-taught learning [38] is a method that integrated the concepts of semi-supervised and multi-task learning, in which the model can exploit examples that are unlabeled and possibly come from a distribution different from the target distribution. It has already been shown that deep neural networks can take advantage of unsupervised learning and unlabeled examples [39, 40]. Different from other approaches mainly designed for image processing and object detection [41, 42, 43, 44], our deep self-learning model shows the ability to detect drug abuse risk behavior given noisy and sparse Twitter data with a limited availability of annotated tweets.

1.2.6 In Need of an Real-time User Interface for Monitoring Online Social Media Drug Abuse Data

Even though drug abuse has reached epidemic proportions [3], there still lacks tools and means to prevent drug abuse epidemics effectively, especially for local communities and organizations, who are at the front and center of the fight. Several well-known resources have been developed for toxicovigilance monitoring include the National Poisoning Data System [22], the U.S. Food and Drug Administration, the Drug Abuse Warning Network [23] and the MedWatch program [21]. These existing systems usually provide statistical data in the typical yearly fashion, which does not offer adequate information about drug abuse activities in a timely manner. This leads to difficulty in managing available resources and efforts (e.g., antidotes, recovery education, etc.), and to challenges in policy-making towards achieving best practices in prevention and recovery.

In addition, the prevalent usage of social network sites, mobile apps, forums, and the internet marketplace, has increasingly been recognized as a major factor in the spread of drug abuse epidemics [45]. Social media apps and the internet also make the purchase of illegal drugs more convenient; access to drugs can be just a few keystrokes away [46]. As both the exchange of information and the obtaining of drugs become easier and faster, the drug use trends become more volatile, diversified, and potentially lethal. Increasingly, one drug can result in damage, even loss of lives in a short time window (a few hours/days) [46]. It is an urgent demand to detect and monitor drug abuse activities on online social media.

1.2.7 Contributions in this Dissertation

In this chapter, we present work from three papers along the topic of drug abuse detection through online social media.

In the first part, we propose an ensemble of two types of deep learning-based methods as better options, among classifiers, for situations in which the collected data is inevitably imbalanced, because they are more robust than traditional machine learning models. Our ensemble deep learning model combines word-level CNN models and character-level CNN models to perform classification. We compare our models with baseline models on a dataset we collected, where we can configure the class distribution of positive versus negative tweets in the training data and test data. By changing the percentage of positively and negatively labeled data in the dataset, we can simulate the imbalanced datasets that were collected by different means. We validate the performance of different models in a variety of settings to get a clearer picture of how imbalanced data affect classification performance.

In the second part, to address the aforementioned challenges of the noisiness of the Tweet data and the lack of labeled Tweet data, our main contributions are to propose: **(1)** A large-scale drug abuse risk behavior tweets collection mechanism based on supervised machine learning and data crowd-sourcing techniques; and **(2)** A deep self-taught learning algorithm for drug abuse risk behavior detection. We first collect tweets through a filter, in which a variety of drug names, colloquialisms and slang terms, and abuse-indicating terms (e.g., *overdose, addiction, high, abuse*, and even *death*) are combined together. We manually annotate a small number of tweets as seed tweets, which are used to train machine learning classifiers. Then, the classifiers are applied to large number of unlabeled tweets to produce machine-labeled tweets. The machine-labeled tweets are verified again by humans on Mechanical Turk, i.e., a crowd-sourcing platform, with good accuracy but at a much lower cost. The new labeled tweets and the seed tweets are combined to form a sufficient and reliable labeled data set for drug abuse risk behavior detection by applying deep learning models, i.e., convolution neural networks (CNN) [47] and long-short term memory (LSTM) models [48]. Then, we propose a self-taught learning algorithm, in which the training data of our deep self-taught learning models will be recursively augmented with a set of new machine-labeled tweets. These machine-labeled tweets are generated by applying the previously trained deep learning models to a random sample of a huge number of unlabeled tweets. Based on our pervious work [49], we extended the analysis of the classification results from our three million tweets dataset. We apply the proposed model to our geolocation-tagged dataset to acquire classification results for analysis. Results from the analysis show that the drug abuse risk behavior-positive tweets have distinctive patterns of words, hashtags, drug name-behavior co-occurrence, time-of-day distribution and spatial distribution, compared with other tweets. These results show that our approach is highly effective in detecting drug abuse risk behaviors.

Then in the third part of this chapter, we develop a *community-focused drug abuse monitoring and supporting system*, called **DrugTracker**, using social media and geospatial data to provide local communities and organizations with the tools and capabilities to identify and understand the specific needs of drug misusers and/or abusers in near real-time. A such system will have crucial benefits to connect local communities and organizations with individuals and families, who are struggling with drug abuse, towards a better prevention and recovery outcome. In our system, well-trained deep learning models are integrated into the data collection process to detect tweets that contain drug abuse risk behaviors. Then end-users can operate the web-based interactive monitoring interface to browse the collected data in a spatial-temporal context in order to acquire insightful patterns about drug abuse risk behaviors. *Our system is available on GitHub: `https://github.com/hu7han73/DrugVis`.*

1.3 Differential Privacy in Deep Learning with Certified Robustness Bounds

1.3.1 Background

Today, deep learning has become the tool of choice in many areas of engineering, such as autonomous systems, signal and information processing, and data analytics. Deep learning systems are, therefore, not only applied in classic settings, such as speech and handwriting recognition, but also progressively operate at the core of security and privacy critical applications.

For instance, self-driving cars make use of deep learning for recognizing objects and street signs [50]. Detection systems for email spam integrate learning methods for analyzing data more effectively [51]. Furthermore, deep learning has applications in a number of healthcare areas, e.g., phenotype extraction and health risk prediction [52], prediction of the development of various diseases, including schizophrenia, cancers, diabetes, heart failure, etc. [53], and many more. This presents an obvious threat to privacy in new deep learning systems which are being deployed. Yet, there are only a few scientific studies in preserving privacy in deep learning.

Lifelong learning (**L2M**) is crucial for machine learning to acquire new skills through continual learning, pushing machine learning toward a more human learning in reality. Given a stream of different tasks and data, a deep neural network (**DNN**) can quickly learn a new task, by leveraging the acquired knowledge after learning previous tasks, under constraints in terms of the amount of computing and memory required [54]. As a result, it is quite challenging to train an L2M model with a high utility.

1.3.2 Privacy Protection in Deep Learning

In the past few decades, a subject of significant interest has been how to release the sensitive results of statistical analyses and data mining, while still protecting privacy. One state-of-the-art privacy model is ϵ-differential privacy [55], which ensures that the adversary cannot infer any information about any specific record with high confidence (controlled by a privacy budget) from the released learning models, even if all the remaining tuples of the sensitive data are possessed by the adversary. The privacy budget controls the amount by which the output distributions induced by two neighboring datasets may differ: A smaller privacy budget value enforces a stronger privacy guarantee. Differential privacy research has been studied from both theoretical and application perspectives [56, 57]. The mechanisms of achieving differential privacy mainly include adding Laplace noise [55], the exponential mechanism [58], and the functional perturbation method [56].

It is significant and timely to combine differential privacy and deep learning, i.e., the two state-of-the-art techniques in privacy preserving and machine learning. However, this is a challenging task, and only a few scientific studies have been conducted. In [59], Shokri and Shmatikov proposed a distributed training method, which injects noise into *"gradients"* of parameters, to preserve privacy in neural networks. In this method, the magnitude of injected noise and the privacy budget ϵ are accumulated in proportion to the number of training epochs and the number of shared parameters. Thus, it may consume an unnecessarily large portion of the privacy budget, as the number of training epochs and the number of shared parameters among multiple parties are often large [60].

To improve this, based on the composition theorem [61], Abadi et al. [62] proposed a privacy accountant, which keeps track of privacy spending and enforces applicable privacy policies. However, the approach is still dependent on the number of training epochs, as it introduces noise into *"gradients"* of parameters in every training step. With a small privacy budget ϵ, only a small number of epochs can be used to train the model [62]. In practice, that could potentially affect the model utility, when the number of training epochs needs to be large to guarantee the model accuracy.

A recent approach towards differentially private deep neural networks was explored by Phan et al. [60]. This work proposed deep private auto-encoders (dPAs), in which differential privacy is enforced by perturbing the cross-entropy errors in auto-encoders [39]. Their algorithm was designed particularly for auto-encoders, in which specific objective functions are applied. A different method, named **CryptoNets**, was proposed in [63] towards the application of neural networks to encrypted data. A data owner can send their encrypted data to a cloud service that hosts the network, and get encrypted predictions in return. This method is different from our context, since it does not aim at releasing learning models under privacy protections.

1.3.3 Privacy Protection with Adversarial Defence

The pervasiveness of machine learning exposes new vulnerabilities in software systems, in which deployed machine learning models can be used (a) to reveal sensitive information in private training data [64], and/or (b) to make the models misclassify, such as *adversarial examples* [65]. Efforts to prevent such attacks typically seek one of three solutions: **(1)** Models which preserve differential privacy (**DP**) [55], a rigorous formulation of privacy in probabilistic terms; **(2)** Adversarial training algorithms, which augment training data to consist of benign examples and adversarial examples crafted during the training process, thereby empirically increasing the classification accuracy given adversarial examples [66, 67]; and **(3)** Certified robustness, in which the model classification given adversarial examples is theoretically guaranteed to be consistent, i.e., a small perturbation in the input does not change the predicted label [68, 69, 70].

On the one hand, *private models*, trained with existing privacy-preserving mechanisms [62, 59, 60, 71, 72, 73, 74], are unshielded under adversarial examples. On the other hand, *robust models*, trained with adversarial learning (with or without certified robustness to adversarial examples), do not offer privacy protections to the training data [75]. That one-sided approach poses serious risks to machine learning-based systems; since adversaries can attack a deployed model by using both privacy inference attacks and adversarial examples. To be safe, a model must be *i) private to protect the training data*, **and** *ii) robust to adversarial examples*. Unfortunately, there still lacks of study on how to develop such a model, which thus remains a largely open challenge [76].

Simply combining existing DP-preserving mechanisms and certified robustness conditions [68, 69, 77] cannot solve the problem, for many reasons. **(a)** Existing sensitivity bounds [60, 71, 72] and designs [73, 74, 76, 78, 79] have not been developed to protect the training data in adversarial training. It is obvious that using adversarial examples crafted from the private training data to train our models introduces a previously unknown privacy risk, disclosing the participation of the benign examples [75]. **(b)** There is an unrevealed interplay among DP preservation, adversarial learning, and robustness bounds. **(c)** Existing algorithms cannot be readily applied to address the trade-off among model utility, privacy loss, and robustness. **(d)** It is challenging in applying existing algorithms to train large DNNs given large data (i.e., scalability); since, they employ the vanilla *iterative batch-by-batch training*, in which only a single batch of data instances can be used at each training step, such that the *privacy loss* can be estimated [74, 76, 73, 78, 79]. That prevents us from applying scalable methods, e.g., *distributed adversarial training* [80], to achieve the same level of DP on large DNNs and datasets. Therefore, bounding the robustness of a model (which both protects the privacy and is robust against adversarial examples) *at scale* is nontrivial.

1.3.4 Privacy Protection in Lifelong Learning

Orthogonal to the difficulties, L2M models are vulnerable to adversarial attacks, i.e., privacy model attacks [81, 64, 82, 81, 83], when DNNs are trained on highly sensitive data, e.g., clinical records [53, 84], user profiles [85, 86], and medical images [87, 88].

In practice, the privacy risk will be more significant since an adversary can observe multiple versions of an L2M model released after training on each task. Different versions of the model parameters can be considered as an additional information leakage, compared with a model trained on a single task (**shown in Theorem 3.7**). Memorizing previous tasks while learning new tasks further exposes private information in the training set, by continuously accessing the data from the previously learned tasks (i.e., data stored in an episodic memory [54, 89, 90, 91, 92]); or accessing adversarial examples produced from generative memories to imitate real examples of past tasks [93, 94, 95]. Unfortunately, there is a lack of study offering privacy protection to the private training data in L2M.

To address this problem, we propose to preserve differential privacy (**DP**) [55], a rigorous formulation of privacy in probabilistic terms, in L2M. Applying existing DP-preserving mechanisms in deep learning [60, 96, 62, 59, 73, 74, 97], which have not been designed for L2M, cannot solve the problem, for many reasons. Compared with single trained models: **(1)** The continual learning and memorizing process consumes a large privacy budget, since the privacy loss can be accumulated in both learning and memorizing across tasks; **(2)** The growing of the episodic memory after each training task (i.e., by adding data into the episodic memory) makes it hard to bound the privacy budget; **(3)** Releasing one version of an L2M model will cause additional privacy risk in learning a new task or training on new data of past tasks; and **(4)** The difference in terms of data sizes, i.e., heterogeneity, among tasks introduces a challenge in addressing the trade-off between privacy protection and model utility, since i) different tasks requires different numbers of training steps, and ii) tasks can be trained in different orders. Both factors can affect DP protection (**Equation (3.80), Theorem 3.7**). These issues prevent us from training and releasing L2M models on streaming tasks and data, under DP protection. Thus, preserving DP in L2M remains a largely open challenge.

1.3.5 Contribution in this Dissertation

Our contribution in this dissertation is three-fold.

Contribution to the Basic Differential Privacy Mechanism There is an urgent demand for the development of a privacy preserving mechanism, such that: **(1)** It is totally independent of the number of training epochs in consuming privacy budget; **(2)** It has the ability to adaptively inject noise into features based on the contribution of each to the model output; and **(3)** It can be applied in a variety of deep neural networks. Mechanisms with such characteristics will significantly enhance the operation of privacy preservation in deep learning.

Motivated by this, we develop a novel mechanism, which is called **Ad**aptive **L**aplace **M**echanism (**AdLM**), to preserve differential privacy in deep learning. Our idea is to intentionally add *"more noise"* into features which are *"less relevant"* to the model output, and vice-versa. To achieve that, we inject Laplace noise into the computation of Layer-wise Relevance Propagation (LRP) [98] to estimate a differentially private relevance of each input feature to the model output. Given the perturbed features, we figure out a novel way to distribute adaptive noise into affine transformations and loss functions used in deep neural networks as a preprocessing step, so that preserving differential privacy is feasible. As a result, we expect to improve the utility of deep neural networks under ϵ-differential privacy. It is worth noting that our mechanism does not access the original data again in the training phase. Theoretical analysis derives the sensitivities and error bounds of our mechanism, and shows that they are totally independent of the number of epochs.

Different from [62, 59], in our mechanism, the injected noise and the privacy budget consumption do not accumulate in each training step. Consequently, the privacy budget consumption in our mechanism is totally independent of the number of training epochs. In addition, different from [60], our mechanism can be applied in a variety of deep learning networks with different activation functions. Convolution neural networks (**CNNs**) [47] are used as an example to validate the effectiveness of our mechanism. Rigorous experiments conducted on MNIST and CIFAR-10 datasets [99] show that our mechanism is effective and outperforms existing solutions.

Contribution to the Scalable Differential Privacy Mechanism Motivated by the open problem of differential privacy with adversarial learning, we develop a novel *stochastic batch (**StoBatch**) mechanism* to: **1)** preserve DP of the training data, **2)** be provably and practically robust to adversarial examples, **3)** retain high model utility, and **4)** be scalable to large DNNs and datasets.

In StoBatch, privacy-preserving noise is injected into inputs and hidden layers to achieve DP in learning private model parameters (**Theorem 3.2**). Then, we incorporate ensemble adversarial learning into our mechanism to improve the decision boundary under DP protections, by introducing a concept of *DP adversarial examples* crafted using benign examples in the private training data (**Equation (3.49)**). To address the trade-off between model utility and privacy loss, we propose a new DP adversarial objective function to tighten the model's global sensitivity (**Theorem 3.4**); thus, we reduce the amount of noise injected into our function, compared with existing work [60, 71, 72]. An end-to-end privacy analysis shows that, by slitting the private training data into *disjoint* and *fixed* batches across epochs, the privacy budget in our StoBatch is not accumulated across *gradient descent*-based training steps (**Theorems 3.4, 3.5**).

After preserving DP in learning model parameters, we establish a new connection between DP preservation in adversarial learning and certified robustness. Noise injected into different layers is considered as a sequence of randomizing mechanisms, providing different levels of robustness. By leveraging the *sequential composition theory* in DP [100], we derive a generalized robustness bound, which is a composition of these levels of robustness in both input and latent spaces (**Theorem 3.6** and **Corollary 3.1**), compared with only in the input space [70] or only in the latent space [101].

To bypass the iterative batch-by-batch training, we develop a *stochastic batch training*. In our algorithm, disjoint and fixed batches are distributed to local trainers, each of which learns DP parameters given its local data batches. A synchronous scheme can be leveraged to aggregate gradients observed from local trainers; thus enabling us to efficiently compute adversarial examples from multiple data batches at each iteration. This allows us to scale our mechanism to large DNNs and datasets, under the same DP guarantee. Rigorous experiments conducted on MNIST, CIFAR-10 [47, 99], and [102] datasets show that our mechanism notably enhances the robustness and scalability of DP DNNs.

Contribution to the Differential Privacy in Lifelong Learning Motivated by the problem in L2M, we introduce a new definition of lifelong differential privacy (**Lifelong DP**), in which the participation of any data tuple in any tasks is protected under a *consistently bounded* DP guarantee, given the released parameters in both learning new tasks and memorizing previous tasks (**Definition 3.5**). This is significant, by allowing us to train and release new versions of an L2M model, given a stream of tasks and data, under DP protection.

Based upon this, we propose a novel algorithm, denoted as **L2DP-ML**, to preserve Lifelong DP. In L2DP-ML, privacy-preserving noise is injected into inputs and hidden layers to achieve DP in learning private model parameters in each task (**Algorithm 3.6**). Then, we configure the episodic memory as a stream of fixed and disjoint batches of data (**Algorithm 3.7**), to efficiently achieve Lifelong DP (**Theorem 3.8**). The previous task memorizing constraint is solved, by inheriting the recipe of the well-known A-gem algorithm [54], under Lifelong DP. To our knowledge, our study establishes the first formal connection between DP preservation and L2M compared with existing work of DP in L2M [103, 104], in which there is a lack of a concrete definition of adjacent databases with unclear or not well-justified DP protection.

Rigorous experiments, conducted on permuted MNIST, permuted CIFAR-10 datasets [105], and an L2M task on our collected dataset for human activity recognition in the wild show promising results in preserving DP in L2M. Lifelong DP opens a long-term avenue to achieve better model utility with lower computational cost under DP protection in L2M.

1.4 Federated Learning on Mobile Devices

1.4.1 Background

Federated Learning (FL) [106] has the potential to bring deep learning (DL) on mobile devices, while preserving user privacy during model training. FL balances model performance and user privacy through three design features. First, each device trains a local model on its raw data. Second, the gradients of the local models from multiple users are sent to a server for aggregation to compute a global model that is more accurate than individual local models. Third, the server shares the global model with all users. During this federated training, the raw data from individual users never leave their devices. A wide range of mobile apps, e.g., predicting or classifying health conditions based on mobile sensing data, can benefit from running DL models on smart phones using FL, which offers privacy-preserving global training that incentivizes user participation.

Despite progress on theoretical aspects and algorithm/model design for FL [107, 108, 109, 110, 111], the lack of a publicly available FL system has precluded the widespread adoption of FL models on smart phones, despite the potential of such models to apply DL on mobile [sensing] data, in a privacy-preserving manner, for novel mobile apps. Furthermore, this has also limited our understanding of how real-world applications can benefit from FL. To the best of our knowledge, the only existing FL systems are either unavailable for the research and practice communities (e.g., Google [106], FedVision [112]), under development [113], or do not support mobile devices [114]. Most of the existing FL studies are based on simulations [115, 107, 108, 109, 110, 111], which may lead to an oversimplified view of the applicability of FL models in real-world. In the meantime, although demonstrated in several scenarios such as keyboard typing prediction [116], FL lacks real-world applications, which can drive the design of FL systems. Indeed, real-world benchmarks for FL are pivotal to help shape the developments of FL systems [117].

In this work, we take a unique application-system co-design approach to design, build, and evaluate an FL system. Our system design is informed by a critical mobile app: human activity recognition (HAR) on the phones, which is important for industry, public health, and research. Simply speaking, mobile apps using HAR can harness recognized human physical activities using data collected from phone sensors. From an industry point of view, accurate HAR can help the smart phone manufacturers to be smart about allocating resources and extending battery life. The Covid-19 pandemic highlights the public health importance of understanding individual & population behaviors under government orders and (health) emergencies [118]; furthermore, combining user activities with mental wellness surveys and prediction has the potential to develop personalized interventions to help individuals to better cope with anxiety, stress, and substance abuse, and other important societal issues [119]. Current research on HAR models uses centralized learning on data collected in controlled lab environments on standardized devices and controlled activities [120, 121, 122, 123, 124, 125, 126]. Instead, we use HAR in the wild (open environments, where the user mobility, activities, or application usage are not controlled in any way). The privacy-sensitive nature of the mobile (sensor) data make this application ideal for studying the design of FL systems. Furthermore, this paper presents the first HAR study under FL.

In addition to HAR, we analyzed other real-life applications [127, 106, 128, 116, 112] to inform our system design. A list of important questions emerges, and many of these questions are not addressed in existing FL system designs [106, 116, 113, 128] that largely ignored the constraints of mobile devices:

- How can we balance FL model performance with resource constraints on the phones?
- How to ensure the training conducted on phones is completed on time, despite limited resources, i.e., computation power and battery life?
- How can the server achieve seamless scalability in the presence of large and variable numbers of users who typically train different models and how can the system simultaneously cope with potential communication failures (e.g., connectivity lost on the phone)?
- How does the server aggregate individual training outcomes efficiently for an accurate model? After a global model is shared with the phones, how can a third-party DL app utilize this model?

Currently, there is no FL system in the literature that can address most of these questions. In terms of design, the closest related work is [106], which focuses on system scalability, secure aggregation, and fault-tolerance. However, it does not present a system implementation and evaluation, and how to use third-party models and make them available to third-party apps on mobile devices. Mobile operating system (OS) providers use FL in their OSs for applications such as next word prediction on the keyboard [116], but their solutions are application-specific and lack system details. Verma et al. [129] and Liu et al. [112] introduce web-services based FL architectures, which are not tailored to mobile devices. An under construction FL system [113] does not focus on application-system co-design aspects such as efficient data collection or scalability. Yet another open source system, FATE [114], currently has only elementary support for deep learning and does not have any support for mobile devices.

1.4.2 Federated Learning Systems

FL can be categorized into Horizontal FL, Vertical FL, and Federated Transfer Learning (FTL) [128]. In Horizontal FL, data are partitioned by device user Ids, such that users share the same feature space [128]. In Vertical FL, different organizations have a large overlapping user space with different feature spaces. These organizations aim at jointly training a model to predict the same model outcomes, without sharing their data. In FTL, the datasets of these organizations differ in both the user space and the feature space.

In Vertical FL and FTL, different organizations need to align their common users and exchange intermediate results by applying encryption techniques [130]. The server cannot just average the gradients, but it needs to minimize a joint loss. At inference stage, the organizations may have to send their individual intermediate results to the server to compute a final result. The systems of these two categories rely on cryptography and their interactions are more complex. Our FLSys focuses on Horizontal FL, with an option for extension to Vertical FL and FTL in the future. For simplicity, we will use FL to indicate Horizontal FL in the rest of the work.

The work in [106] describes the design of a scalable FL system for mobile devices. This system shares several of our design goals (e.g., scalability, fault-tolerance). However, this work does not present a system implementation and evaluation, as we do for FLSys. Also, FLSys addresses additional unanswered design questions such as how to train concurrently multiple models for different applications, and how third party app developers to use the system. Furthermore, unlike this work, FLSys focuses on data collected from the phone's sensors, which adds challenges related to efficient and effective data collection. Another system that shares some goals with FLSys is FedML [113], which is still under construction. In addition, FedML focuses more on software engineering aspects, rather than on system aspects such as efficient sensor data collection or scalability as in FLSys. A third open source system FATE [114] is still in its infancy with limited support of deep learning and does not work on mobile devices.

While a significant amount of FL research focuses on improving the security and privacy of federated training procedures [131, 132], this is outside the scope of our research. We focus on system design, implementation, and evaluation using HAR and SA models.

1.4.3 Coping with Non-IID Data in FL Training

A well-reported issue restricting the perform-ance of models trained by FL is non-IID data distribution across users [133, 109]. Different from centralized learning, the datasets among different users may follow different distributions in FL, because of the heterogeneous devices, imbalanced class distribution, different user behaviors, etc. As a result, DL models trained in FL algorithms usually suffer from inferior performance when compared with centralized models [133].

To mitigate the non-IID issue, several algorithms have been proposed [107, 108, 109, 110, 111]. In FedProx [107], a regularization is introduced to mitigate the gradient distortion from each device. Sarkar et al. [108] presented a cross-entropy loss to downweigh easy-to-classify examples and focus training on hard-to-classify examples. Verma et al. [111] proposed to estimate the global objective function by averaging different objective functions given a common region of features among users, and keep different objective functions estimated from local users' data in different regions of the feature space. Data augmentation approaches have been proposed [109], including a global data distribution based data augmentation [110]. The federated training of our HAR-Wild and SA models use a uniform data augmentation method, similar to these techniques.

1.4.4 Human Activity Recognition

Our HAR model focuses on sensing and classification of physical activities through smart phone sensors. Recent work show that deep learning models are effective in HAR tasks. For example, Ignatov [120] proposed a CNN based model to classify activities with raw 3-axis accelerometer data and statistical features computed from the data. Several pieces of work [121, 122, 126] proposed LSTM-based models and achieved similar performances.

Most research on HAR models uses centralized learning on data collected in controlled lab environments with standardized devices and controlled activities, in which the participants only focus on collecting sensor data with a usually high and fixed sampling rate frequency, i.e., 50Hz or higher. Although there are good publicly available HAR datasets, e.g., WISDM [123], UCI HAR [124], and Opportunity [125], they are not representative for real-life situations. Different from existing work, this work shows that HAR-Wild over FLSys performs well on the data collected in the wild, which are subject to fluctuating sample rates and non-IID data distribution.

1.4.5 Contributions in this Dissertation

In this dissertation, we presents **FLSys**, the first FL system in the literature created using an application-system co-design approach to address the aforementioned research questions.

FLSys is a key component toward creating an open ecosystem of FL models and apps that use these model. Such an FL ecosystem will allow third-party model/app developers to easily develop and deploy FL models/apps on smart phones. Consequently, the users will benefit from novel FL apps based on mobile [sensing] data collected on the phones.

To tackle fault-tolerance and resource constraints on the phones, FLSys utilizes an asynchronous interaction model between mobile devices and the cloud, which 1) allows the devices to self-select for training when they have enough data and resources, and 2) allows the sever to operate correctly in the presence of communication failures with the phones. For energy efficient data collection, FLSys supports on-demand configuration of sensor types, sampling rates, and the period for data flushing from memory to storage. The FL server in the cloud achieves good scalability through a design based on function as a service computation and scalable storage. FLSys is flexible, in the sense that it can train multiple models concurrently. It provides a common API for third-party apps to train different DL models with different FL aggregation methods in the cloud. While implemented in Android and AWS, FLSys has a general system design and API that can be extended to other mobile OSs and cloud platforms.

To study how HAR can be supported by FLSys in the wild, we collected data from 100+ college students in two areas during April - August 2020. The students used their own Android phones, and their daily-life activities were not constrained in any way by our experiment. Data collected on mobile devices is non-IID, which affects FL-trained models [127]. We have evaluated a variety of HAR models in both centralized and federated training, and designed **HAR-Wild**, a Convolution Neural Network (CNN) model with a data augmentation mechanism to mitigate the non-IID problem. To showcase the ability of FLSys to work with different FL models, we also built and evaluated a natural language sentiment analysis (SA) model on a dataset with 46,000+ tweets from 436 users.

We carried out a comprehensive evaluation of FLSys together with HAR-Wild and SA to quantify the model utility and the system feasibility in real life conditions. We performed the evaluation under three training settings: 1) centralized training, 2) simulated FL, and 3) Android FL. Centralized training provides an upper bound on model accuracy and is used to compare our HAR-Wild model with baseline approaches. The results demonstrate that HAR-Wild outperforms the baseline models in terms of accuracy. Furthermore, the federated HAR-Wild performance using simulations (TensorFlow and DL4J), Android emulations, and Android phone experiments is close to the upper bound performance achieved by the centralized model. The results on smart phones demonstrate that FLSys can perform communication and training tasks within the allocated time and resource limits, while the FL server is able to handle a variable number of users. Finally, micro-benchmarks on Android phones show FLSys with HAR-Wild and SA are practical in terms of training and inference time, memory and battery consumption.

CHAPTER 2

DRUG ABUSE DETECTION AND ANALYSIS IN ONLINE SOCIAL MEDIA

Chapter Abstract: Abuse of prescription drugs and of illicit drugs has been declared a “national emergency” [1]. This crisis includes the misuse and abuse of cannabinoids, opioids, tranquilizers, stimulants, inhalants, and other types of psychoactive drugs, which statistical analysis documents as a rising trend in the United States. Even though drug abuse has reached epidemic proportions [3], there still lacks tools and means to prevent drug abuse epidemics effectively, especially for local communities and organizations, who are at the front and center of the fight. This section of work has three aims: (1) To design data collection pipelines that collects potentially drug abuse-related posts, and deep-learning models that classify collected posts as drug abuse-related or not, in order to facilitate large-scale and time-sensitive detection of drug abuse trends through the lens of online social media; (2) To analyze the collected data and the classification results in terms of patterns of words, hashtags, drug name-behavior co-occurrence, time-of-day distribution and spatial distribution, which demonstrates the effectiveness of such method designed in the first aim, as well as providing insights for future research; and (3) Design and implement a tool to visualize the collected data, the classification results, and the analysis results in a web-based interactive user interface to support local communities.

2.1 An Ensemble Deep Learning Model for Drug Abuse Detection in Sparse Twitter-Sphere

2.1.1 Methods

In this subsection, we present the definition of the drug abuse-related risk behavior detection problem, our methods for collecting tweets, our methods for labeling tweets, and our ensemble deep learning approach.

Problem Definition In this work, our first goal is to build a Twitter dataset consisting of tweets that are related to drug abuse risk behaviors (positive tweets), and tweets that are not (negative tweets). The "drugs" in the term "drug abuse risk behaviors" in this study include Schedule 1 and Schedule 2 drugs and their derivatives [134], including marijuana, heroin, cocaine, fentanyl, etc. The reasons that we include marijuana even though it is legalized in several states are that: (1) Marijuana is still a controlled substance in the federal law, whether for medical use or recreational use; and (2) Marijuana can still cause harm to adolescents [7], can cause "use disorder" [24], and is related to traffic fatalities [6]. The term "abuse risk behavior" can be defined as "The existence of likely abusive activities, consequences, and endorsements of drugs." Tweets that contain links to or summarize news and reports related to drug abuse, and tweets that merely express opinions about drug abuse, are counted as negative in this study. Our main goal in this dissertation is to train a model that can accurately classify positive and negative tweets in a highly imbalanced (drug abuse) dataset.

Data Collection Although there are human-labeled drug abuse Twitter datasets (e.g., Sarker's dataset [15]) available, due to Twitter's data policy, which prohibits the direct sharing of tweet contents, by the time we access the tweets in that dataset, more than 40% of tweets are either removed or hidden from the public. This significantly affects the quality and integrity of existing publicly available datasets. Therefore, we need to build a new dataset from scratch. In our framework, raw tweets are collected through a set of Application Programming Interfaces (Twitter APIs) via keyword filtering. By defining a set of keywords, the API will fetch tweets that contain any of the keywords from either the real-time stream of tweets or from archived tweets. For a more complete coverage of drug-related topics, we selected three types of keywords: (1) Formal drug names, e.g., marijuana, cocaine, OxyContin, fentanyl, etc.; (2) Slang terms for drugs, e.g., pot, blunt, coke, crack, smack, etc.; and (3) Drug abuse-related behaviors and symptoms, e.g., high, amped, addicted, headache, dizzy, etc. The number of keywords we used is limited to 400 by the Twitter APIs.

Data Annotation We build a comperhaensive guide, accessable at https:// goo.gl/tqWddS, based on Sarker's guide [15]. Each one of the three members in out research team with experience in health informatics annotates the 1,794 tweets from Hu et al.'s study [49] independently following the guide. A final label for each seed tweet is determined by majority voting from three labels.

To acquire annotated tweets rapidly, at low cost, and with increased percentage of positive tweets, we (1) use these labeled tweets as "seed" tweets to train a SVM classifier; (2) run the SVM classifier on the unlabeled dataset, and randomly sample 5,000 machine labeled tweets that have prediction probability (esitmated with Platt scaling) > 0.8; and (3) post the 5,000 tweets (without identification information) onto the Amazon Mechanical Turk (AMT) crowdsourcing platform for annotation. AMT is a well-known crowdsourcing platform where Posters can post Human Intelligence Tasks (HITs) and Workers finish HITs for micro-payments. A literature study [135] evaluated AMT as a thrustworthy platform to obtain human labeled data. The same guide is used to guide the Workers how to annotate the tweets. Each tweet is posted as one HIT that requires the Worker to label it as positive or negative following the guide. Each HIT is replicated as three assignments to be completed by three individual Workers. We set the price of each assignment to be $0.05, a very generous price compared to what was reported in Buhrmester's work [135] All HITs are completed within hours after being posted. The final label of each tweet is aggregated from the three labels by majority voting. Our annotators also label 1,000 tweets randomly sampled from the 5,000 tweets with as a measure of quality check.

Feature Extraction Machine learning models require numerical features to work with. Feature extraction transforms text features into numerical features in the form of vectors. To cover the content ambiguity in drug abuse-related tweets, a variety of feature extraction methods are used in this study. In our word-level CNN models, we use pre-trained word embedding models that were trained on large corpora to transform words into dense vectors. We test several pre-trained models as Mahata's work [136] suggested. With our word-level CNN model, the Drug Chatter embedding has the best average performance on our dataset; thus, it is chosen as the pre-trained word embedding model for this study. The details of the tested word embedding models are shown in Table 2.1. Each tweet is converted to a sequence of 400-dimensional vectors. Considering that the length limit of each tweet nowadays is 280 chars, the sequence length is set to 40. In our char-level CNN, the preprocessing step only turns all characters to lower case as suggested by [137]. Each character is then converted into a 128-dimensional trainable randomly-initialized vector. Instead of being fixed, the character embeddings are trained along with other layers in the model.

We also replicate the features extracted in Sarker et al. study [15], including: (1) The tokenization process; (2) The abuse-indicating term features, consisting of the presence and the counts of abuse-indicating terms obtained from Hanson et al. [17]; (3) The drug-slang lexicon features, consisting of the presence and the counts of terms longer than five characters found in an online drug abuse dictionary [138]; (4) The word cluster features, represented by 150-dimensional one-hot vectors, were constructed by identifying words that belong to certain word clusters in a dataset [15] that contains 150 drug-related word clusters; and (5) The synonym expansion features, accomplished by identifying all synonyms of all nouns, verbs, and adjectives in the tokenized tweets using WordNet [139].

An Ensemble Deep Learning Model for Drug Abuse Detection in Sparse Twitter-Sphere In this subsection, we present our novel ensemble deep learning model for drug abuse risk behavior detection by integrating extracted features from tweets into CNN models. Our ensemble model takes the outputs of multiple prediction models, word-level CNN (W-CNN) and char-level CNN (C-CNN) [137] in our case, and feed them to a

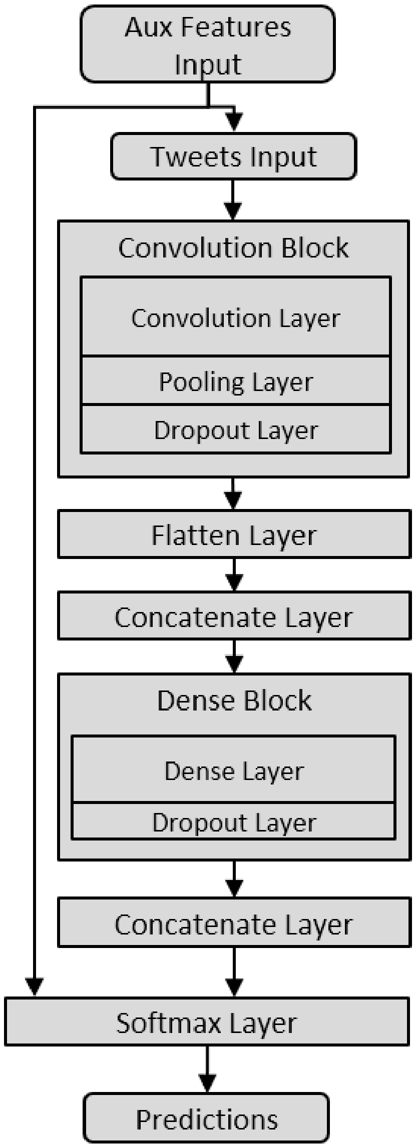
Table 2.1 Details of Pre-trained Word Embeddings

Name	Model	Corpus	Vocabulary	Dim.
GoogleNews [140]	Word2vec	100 billion words	3 million	300
Glove_Comm [141]	Glove	42 billion words	1.9 million	300
Godin [142]	Word2vec	400 million tweets	3 million	400
Drug_chatter [143]	Word2vec	1 billion tweets	1.6 million	400

meta-learner that gives the final predictions. We design W-CNN and C-CNN for this task. In fact, both the W-CNN and the C-CNN share a similar structure as shown in Figure 1.

The inputs of our W-CNN are vectors of shape [40, 400] where 40 is the maximum sequence length (number of words allowed) in an input tweet, and 400 is the length of the pre-trained word embeddings. The input of our C-CNN is shaped as [280, 128] where 280 is the maximum possible length of a tweet, and 128 is the length of the vector representation of each character in the charset. The auxiliary features in the input include: (1) The synonym expansion features in the form of synonymous words are directly concatenated with the input tweets (before they are transformed into vectors); and (2) The remaining auxiliary features, in the form of 154-dimensional vectors, are concatenated to the last hidden layer of the dense layers. For each convolution kernel size, the W-CNN model has two convolution layers with ReLU activation functions stacked together. Each is followed by a max-pooling layer.

The C-CNN model has one convolution layer for each convolution kernel size with Tanh activation function, followed by a global-max-pooling layer that performs max-pooling over all outputs of convolution layers with different kernel sizes. Both models have one dense layer block, consisting of two dense layers with 1,024 hidden units each, and one Softmax output layer with two units. The activation functions are slightly different, as the W-CNN model uses ReLU, while the C-CNN model uses SELU. The output of the



Input Parameters

Parameter	W-CNN	C-CNN
tweet input dimension	[40,400]	[280,128]
aux feature dimension	[154,]	[154,]

Convolution Block Parameters

Parameter	W-CNN	C-CNN
kernel width	{2,3,4,5}	{3,5,7,10}
# kernels	128 for each kernel width	256 for each kernel width
activation function	Relu	Tanh
pooling	MaxPoolin g	GloablMax-Pooling
dropout	Uniform-Dropout with rate=0.1	Alpha-Dropout with rate=0.1
# blocks	2	1

Dense Block Parameters

Parameter	W-CNN	C-CNN
# hidden units	1024	1024
activation function	Relu	Selu
dropout	Uniform-Dropout with rate=0.5	Uniform-Dropout with rate=0.5
# blocks	2	2

Figure 2.1 Ensemble CNN model structures.

last hidden layer is concatenated with vectors of abuse-indicating term features, drug-slang lexicon features, and word cluster features, before being fed into the output layer.

Finally, a number of independently trained CNN models of both types are ensembled together by majority voting. Model ensembles were also used in Sarker et al. study [9] to reduce variability and bias, in order to improve prediction performance. We apply the same ensemble strategy to both our deep learning models and the baseline models.

Experimental Design Our main objective in this experiment is to directly compare the performances of the ensemble traditional machine learning model and the ensemble deep learning model. For the ensemble traditional machine learning model, two of each type of baseline models, six in total, are trained and ensembled together. For the ensemble deep learning model, six models of three types (two for each type) are used. The three types are denoted as follows. (1) "char_aux" is the char-level CNN model with auxiliary features. (2) "char_cnn" is the plain char-level CNN without any auxiliary features. (3) "word_aux" is the word-level CNN model with all auxiliary features. For deep learning models, it is extremely easy to overfit, due to the rather small number of training and test data elements; thus, the model is saved at each training epoch, and the best epoch is found among the saved models. For each class distribution scenario, each model is trained with the same six sets of training data and tested on the corresponding test data. All results reported are averaged results from the 6-fold cross-validation.

2.1.2 Experimental Results

Data Annotation Results From Jan 2017 to Feb 2017, we collected 3,265,153 tweets in total. The "seed" dataset that we annotated to be used to train the pre-filter consistes of 1,794 tweets, including 280 positive labels and 1,514 negative labels. Our annotator achieved the agreement score of 0.414, measured by Krippendoff's Alpha. For the AMT labeled dataset, we removed duplicate tweets from it, resulting dataset contains 4,736 tweets with 2,657 positive labels and 2,079 negative labels. The agreement score is 0.456 measured by Alpha, which can be considered as a reliable result in our study as demostrated in [144], since: (1) We are performing data annotation with data aggregation to reduce variability, instead of typical content analysis [145]; (2) The Krippendoff's Alpha is sensitive to data imbalance; and (3) We focus on sparse and imbalanced data distributions. For the 1,000 tweets for quality check, we got the Kappa score of 0.910 between our final labels and the labels we obtain from AMT. This is showing that our annotation guide was followed consistently by both our annotators and AMT Workers.

To simulate the data imbalance scenarios, we configured the class distribution and pre-sampled the dataset into six blocks for each distribution scenario, for 6-fold cross-validation. Each model was trained and tested on the same sets of training and test data to ensure a fair comparison. The number of data points included in each distribution scenario was maximized, but it was inevitably different between scenarios. Table 2.2 shows the dataset in each class distribution scenario.

Table 2.2 Imbalanced Tweets Dataset Variants

Class Distribution (positive: negative)	**# of training data**	**# of testing data**
50:50 split	3450	690
40:60 split	2850	570
30:70 split	2450	490
20:80 split	2150	430
10:90 split	1900	380

Drug Abuse Detection Results Table 2.3 shows the results for all individual models and two ensemble models. The ensemble model results are separated from the individual models for easier viewing. The highest value of each measure is marked in bold font. There is an interesting trend in the results of ensemble models. When the data is balanced or nearly balanced, the traditional ensemble machine learning model has a better performance than the ensemble deep learning model. At 50:50 and 40:60 splits, the ensemble machine learning model is superior over the ensemble deep learning model for most of the criteria. This is partially due to the relatively small dataset size. When the data becomes more imbalanced, e.g., at a 30:70 split, the ensemble deep learning model becomes better and has a higher F1-score for positive labels, compared with the traditional ensemble machine learning model. At 20:80 and 10:90 splits, the ensemble deep learning model takes the lead, most significantly in each measure for positive labels. The larger model capacity and the ability of the deep learning models to learn more complex non-linear functions can better distinguish the semantic differences between positive tweets and negative tweets, when the distribution of classes is heavily imbalanced.

Looking at individual machine learning models, Random Forest and SVM show a strong performance on all datasets, and they are especially good when the dataset is balanced. Naïve Bayes also has a good performance on a balanced dataset, but on an imbalanced dataset, it is heavily biased towards negative labels and has a poor performance for positive labels. Deep learning models generally have more stable performance, compared to traditional machine learning models, across all datasets, and a smaller difference between precision and recall, but their peak performances are not as good. Comparing between deep learning models, auxiliary features do not give C-CNN significant performance boost, and W-CNN is also not as good as the C-CNN model. However, in additional results that are not shown in this dissertation due to space limitations, auxiliary features give the plain W-CNN model a performance boost.

By investigating the performance of each individual model and the ensemble model that includes it, we can see that our ensemble strategy works well for deep learning models, as most of the measures for the ensemble model are higher than for any of its components corresponding measures. This effect was only observed a few times for traditional machine learning models. We expect that, by using more complicated ensemble strategies, deep learning has the potential to reach an even better performance level.

Table 2.3 Results of Each Dataset Variant

Class Distribution: 50:50 split

Measure	**char_aux**	**char_cnn**	**word_aux**	**SVM**	**RF**	**NB**	**Ensemble CNN**	**Ensemble ML**
Accuracy	0.8506	0.8477	0.8466	0.8415	**0.8586**	0.8384	0.851	**0.8575**
Precision_p	0.8315	0.824	0.8198	0.8063	**0.8404**	0.8319	**0.8468**	0.835
Recall_p	0.8797	0.8845	0.8894	**0.9**	0.886	0.8493	0.8575	**0.8918**
F1_score_p	0.8549	0.8531	0.8529	0.8504	**0.8624**	0.8402	0.852	**0.8623**

Class Distribution: 40:60 split

Measure	**char_aux**	**char_cnn**	**word_aux**	**SVM**	**RF**	**NB**	**Ensemble CNN**	**Ensemble ML**
Accuracy	0.8528	**0.8563**	0.843	0.8444	0.8494	0.8427	0.8567	**0.8582**
Precision_p	0.8007	0.8055	0.7818	**0.8104**	0.777	0.7862	**0.8079**	0.8047
Recall_p	0.8421	0.8454	0.8443	0.7982	**0.8746**	0.8341	0.8428	**0.8531**
F1_score_p	0.8207	**0.8248**	0.8113	0.8041	0.8229	0.8093	0.8249	**0.828**

Class Distribution: 30:70 split

Measure	**char_aux**	**char_cnn**	**word_aux**	**SVM**	**RF**	**NB**	**Ensemble CNN**	**Ensemble ML**
Accuracy	0.8522	0.8507	0.8483	0.8429	**0.8537**	0.8452	**0.8599**	0.8595
Precision_p	0.7253	0.7223	0.718	**0.7467**	0.7137	0.7218	0.7402	**0.7426**
Recall_p	0.8209	0.818	0.8158	0.7234	**0.8583**	0.7914	**0.8231**	0.8163
F1_score_p	0.7695	0.7666	0.7635	0.7336	**0.7789**	0.7538	**0.7792**	0.7771

Class Distribution: 20:80 split

Measure	**char_aux**	**char_cnn**	**word_aux**	**SVM**	**RF**	**NB**	**Ensemble CNN**	**Ensemble ML**
Accuracy	**0.8624**	0.8568	0.8506	0.8384	0.8475	0.8527	**0.8674**	0.8508
Precision_p	**0.6325**	0.6128	0.5965	0.564	0.5838	0.6261	**0.6416**	0.5908
Recall_p	0.7558	0.7868	0.8023	**0.8547**	0.8295	0.6609	0.7713	**0.8295**
F1_score_p	**0.6878**	**0.6878**	0.6823	0.6792	0.685	0.6425	**0.7001**	0.69

Class Distribution: 10:90 split

Measure	**char_aux**	**char_cnn**	**word_aux**	**SVM**	**RF**	**NB**	**Ensemble CNN**	**Ensemble ML**
Accuracy	0.8638	0.8664	0.8445	0.8355	0.8592	**0.8961**	**0.8728**	0.8636
Precision_p	0.4112	0.4153	0.376	0.3609	0.3875	**0.4762**	**0.4338**	0.3975
Recall_p	0.7368	0.7346	0.7171	**0.8114**	0.6776	0.2939	**0.7281**	0.6754
F1_score_p	0.5243	**0.5275**	0.4882	0.499	0.4925	0.3611	**0.5389**	0.4999

2.1.3 Conclusions

In this study, we investigated how the data imbalance issue influences the performance of classifiers that are trained for identifying tweets that are related to drug abuse. We first collected a dataset with a broad selection of drug abuse-related keywords and slang terms. We explored the use of the Amazon Mechanical Turk platform as a reliable source for acquiring human-labeled tweets, and we obtained a solid dataset. We designed an ensemble deep learning classification model with both word-level and char-level CNNs, and we conducted a direct comparison with traditional machine learning models on our dataset, with simulated class imbalance. Experimental results show that our ensemble deep learning models have better performance than traditional machine learning models when the data is off-balance. Results also show that the ensemble strategy we used is effective for improving deep learning models. Finally, our analysis of the collected three million tweets, labeled by our model, shows an interesting temporal pattern that agrees with our intuition.

2.2 An Insight Analysis and Detection of Drug-Abuse Risk Behavior on Twitter with Self-Taught Deep Learning

2.2.1 Method

In this subsection, we present the definition of the drug abuse risk behavior detection problem, our system for collecting tweets, labeling tweets, and our deep self-taught learning approach. The system overview is shown in Figure 2.2.

Problem Definition We use the term "drug abuse risk behavior" in the wider sense, including misuse and use of Schedule 1 drugs that are illegal; and misuse of Schedule 2 drugs, e.g., *Oxycodone*, which includes the use thereof for non-medical purposes, and the symptoms and side-effects of misuse. Our task is to develop classification models that can classify a given unlabeled tweet into one of the two classes: a drug abuse risk behavior tweet (**positive**), or a non-drug abuse risk behavior (**negative**) tweet. The main criteria for classifying a tweet as drug abuse risk can be condensed into: "*The existence of abusive activities or endorsements of drugs.*" Meanwhile, news, reports, and opinions about drug abuse are the signals of tweets that are not considered as containing abuse risk.

Tweet Data Collection In our crawling system, raw tweets are collected through Twitter APIs. For the collection of focused Twitter data, we use a list of the names of illegal and prescription drugs [146] that have been commonly abused over time, e.g., *Barbiturates, OxyContin, Ritalin, Cocaine, LSD, Opiates, Heroin, Codeine, Fentanyl*, etc. However, the data is very noisy, since: (1) There is no indication of how to distinguish between drug abuse and legitimate use (of prescription drugs) in collected Tweets, and (2) Many of slang terms are used in expressing drug abuse risk behavior. To address this problem, we added slang terms for drugs and abuse-indicating terms, e.g., "high," "stoned," "blunt," "addicted," etc., into our keyword search library. These slang terms are clearly expressing that the tweets in question were about drug abuse. As a result, most of the collected data is drug abuse-related.

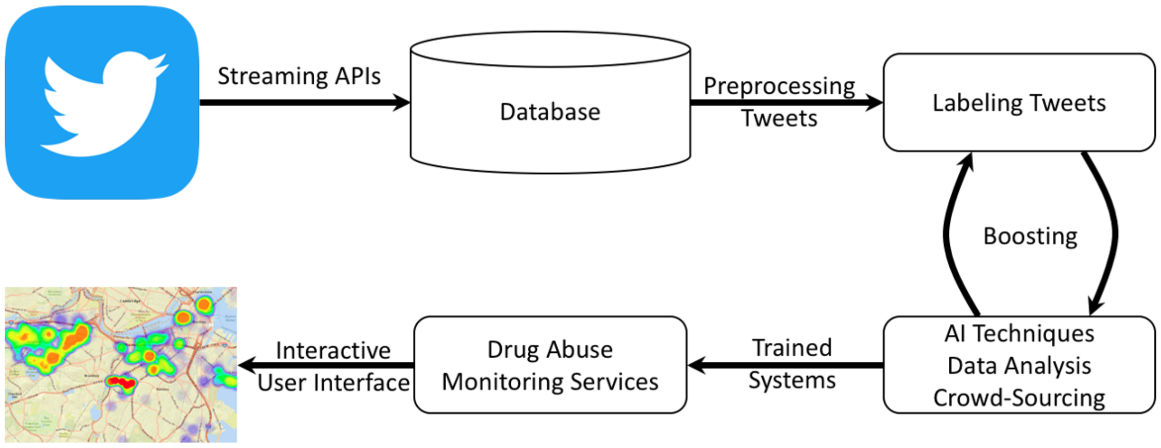


Figure 2.2 Drug Abuse Detection System. There are 4 steps as follows: (1) Tweets will be collected through Twitter APIs. (2) Preprocessed tweets will be labeled by humans, AI techniques, and crowd-sourcing techniques. (3) Labeled tweets will be used to augment the training data of our AI models and data analysis tasks to identify tweets with drug abuse risk behaviors, through a self-taught algorithm. And (4) Trained systems will be used in different drug abuse monitoring services and interactive user interfaces.

To obtain trustworthy annotated data, we design two integrative steps in labeling tweets. In the first step, 1,794 tweets randomly chosen from collected tweets were manually annotated as positive or negative by three team members who have experience in health informatics. Several instances of positive tweets and negative tweets are illustrated in Table 2.4. These labeled tweets are considered seed tweets, which then are used to train traditional binary classifiers, e.g., SVM, Naive Bayes, etc., to predict whether a tweet is a drug abuse risk behavior tweet or not. The trained classifiers are applied to unlabeled tweets to predict their labels, which are called machine labels. In the second step, 5,000 positive machine-labeled tweets with high classification confidence are verified again on Amazon Mechanical Turk (AMT), which is a well-known crowd-sourcing platform. To improve the trustworthiness and to avoid bias in the annotated data, each tweet is labeled by three individual workers. The workers are instructed to follow with the same annotation instructions that our annotators have followed. Our annotators also labeled a random sample of 1,000 tweets and compare the labels with the results from AMT, as a quality check.

Table 2.4 Instances of Manually Annotated Positive Tweets and Negative Tweets

	Tweets
Positive	"Ever since my Acid trips like whenever I get super high I just start - lightly hallucinating and it's tbh creepy."
	"drove like 10 miles on these icy ass roads all to get some weed if imma - be locked up in my house for awhile imma need some weed."
	"Smoking a blunt at home so much better than going to the woods in - Brooksville and puking on yourself Bc you drank too much reball."
Negative	"Just watched Fear and Loathing in Las Vegas for the first time - and I think I should have been on acid to fully understand it."
	"today I was asked if I do heroin because I went to Lancaster????"
	"Morgan told me my Bitmoji looks like a heroin addict?"

Tweet Vectorization Raw tweets need to be first pre-processed, then represented as vectors, before they can be used in training machine learning models. In this study, we choose a commonly used pre-processing pipeline, followed by three different vectorization methods. The pre-processing pipeline consists of following steps:

- The tweets are tokenized and lower-cased. The special entities, i.e., including Emojis, URLs, mentions, and hashtags, are removed or replaced with special keywords. The non-word characters, i.e., including HTML symbols, punctuation marks, and foreign characters, are removed. Words with three or more repeating characters are reduced to at most three successive characters.
- Stop-words are removed according to a custom stop-word list. Stemming is applied using the standard Porter Stemmer.

After the preprocessing steps, common vectorization methods are used to extract features from tweets, including: (1) Term frequency, denoted as *tf*, (2) *Tf-idf*, and (3) Word2vec [140]. Word2vec is an advanced and effective word embedding method that converts each word into a dense vector of fixed length. We considered two different Word2vec models: (i) A custom Word2vec model, which was trained on our three million drug abuse-related tweets. The model contains 300-dimensional vectors for 1,130,962 words and phrases; and (ii) Google Word2vec, which is a well-known pre-trained Word2vec model built from part of a Google News dataset with about 100 billion words, and the model contains 300-dimensional vectors for three million words and phrases.

2.2.2 Deep Self-Taught Learning Approach

By applying both traditional and advanced machine learning models, such as SVM, Naive Bayes, CNN, and LSTM to the small and static annotated data, i.e., 6,794 tweets, we can achieve reasonable classification accuracies of nearly 80%, as indicated in Figure 2.4 when the number of iteration k is zero, which is equivalent to applying models without the proposed self-taught method. To develop a scalable and trustworthy drug abuse risk behavior detection model, we need to: (1) Improve classification models to achieve higher accuracy and performance; and (2) Leverage the large number of unlabeled tweets, i.e., three million tweets related to drug abuse, to improve the system performance. Therefore, we propose a deep self-taught learning model by repeatedly augmenting the training data with machine-labeled tweets. The pseudo-code of our algorithm is as follows:

- **[Step 1:]** Randomly initialize labeled data D consisting of 5,794 annotated tweets as the training set. Initialize a test data T consisting of the remaining 1,000 annotated tweets.
- **[Step 2:]** Train a binary classification model M using the labeled data D. M could be a CNN model or an LSTM model.
- **[Step 3:]** Use the model M to label the unlabeled data U, which simply consists of three million unlabeled tweets. The set of new labeled tweets is denoted as $\overline{D}$, which is also called machine-labeled data.
- **[Step 4:]** Sample tweets from the machine-labeled dataset $\overline{D}$ with a high classification confidence, and then add the sampled tweets D^+ into the labeled data D to form a new training dataset: $D = D \bigcup D^+$. A tweet is considered to have a high classification confidence if it has a classification probability $p \in [0, 1]$ higher than a predefined sampling threshold δ. Sampled machine-labeled tweets will not be sampled again: $U = U - D^+$.
- **[Step 5:]** Repeat Steps 2-4 for k iterations, where k is a user-predefined number. Return the trained model M.

With the self-taught learning method, the training data contains the annotated data D, which is automatically augmented with highly confident, machine-labeled tweets, in each iteration. This approach has the potential of increasing the classification performance of our model over time. In addition, the unlabeled data can be collected from the Twitter APIs in real time, to capture the evolving of English (slang) terms about drug abuse risk behaviors. In the literature, data augmentation approaches have been applied to improve the accuracy of deep learning models [39]. However, the existing approaches [39, 42, 43, 44] are quite different from our proposed model, since they focused on image classification tasks, instead of drug abuse risk behavior detection as in our study. Note that, to ensure fairness, test data T is separated from other data sources during the training process.

2.2.3 Experiments

Dataset The **seed dataset** contains 1,794 tweets that were manually labeled by three annotators, including 280 positive tweets and 1,514 negative tweets. The agreement score among three annotators is 0.414, measured by Krippendoff's Alpha. We then selected 5,000 tweets labeled by the machine learning model (i.e., SVM) with a high confidence level ($\delta > 0.7$), and rendered them verified on AMT. The AMT workers have the agreement score of 0.456, measured by Krippendoff's Alpha. Note that both agreement scores should be considered as reliable result in out study settings [144], since: (1) Our task is to reduce variability in data annotation, instead of typical content analysis [145]; (2) The Krippendoff's Alpha is sensitive to data imbalance and sparseness, which are the characteristics of our dataset. Our integrative labeling approach resulted in a reliable and well-balanced annotated data set, with 6,794 labeled tweets, including 3,102 positive labels and 3,677 negative labels. For the unlabeled data, we have the three million drug abuse-related tweets with geo-location information covering the entire continental U.S. (lower 48 states and D.C.).

Baseline methods In our experiments, Random Forest (**RF**), Naive Bayes (**NB**), and **SVM** are employed as baseline approaches for the binary classification task, i.e., to classify whether a tweet is a drug abuse risk behavior tweet or not. Table 2.5 shows the parameter settings of baseline approaches and the proposed models. Note that for the Naive Bayes method, we use Gaussian Naive Bayes with Word2vec embedding. Meanwhile, we use term frequency (i.e., *tf*) and *tf-idf* vectorization for Multinominal Naive Bayes. This is because: (1) The vectors generated by term frequency-based vectorization has a very high number of dimensions and could be only represented by sparse-matrix, which was not supported by the chosen implementation of Gaussian Naive Bayes; and (2) The Multinominal Naive Bayes require non-negative inputs, but vectors generated by Word2vec embedding has negative values. Regarding our self-taught CNN (**st-CNN**) and self-taught LSTM (**st-LSTM**) models, the Adam Optimizer algorithm with default learning rate is used for training. The number of iterations k is set to 6, and the sampling threshold δ is set to 0.7, for all methods.

All the experiments have been conducted on a single GPU, i.e., NVIDIA TITAN Xp with 12 GB memory and 3,072 CUDA cores.

Table 2.5 Parameter Settings for All Models

Baseline Model	Parameter Setting	
SVM	c=5.0, gamma=0.01, kernel:rbf	
Random Forest	n_estimators=500, class_weight=balanced, max_depth=20	
Naive Bayes (Gaussian)	default	
Naive Bayes (Multinominal)	default	
Proposed Model	**Layers**	**Parameter Setting**
Self-taught CNN (st-CNN)	embedding	size: 300, max_length: 20
	dropout	dropout_rate: 0.2
	convolutional	kernerl_sizes: [2,3,4], number_kernels: 20 activation_function: Relu, strides: 1
	max pooling	pool_size: 2
	flatten	no parameter
	concatenate	no parameter
	dropout	dropout_rate: 0.5
	two dense layers	dense_layer_1: size: 520x500; dense_layer_2: size: 500x2
Self-taught LSTM (st-LSTM)	embedding	size: 300, max_length: 20
	dropout	dropout_rate: 0.2
	LSTM	sequence_output: False
	dropout	dropout_rate: 0.5
	two dense layers	dense_layer_1: size: 300x500; dense_layer_2: size: 500x2

Measures Accuracy, recall, and F1-value are used to validate the effectiveness of the proposed and baseline approaches. Due to the small size and the imbalanced label distribution, we adopted the Monte Carlo Cross-Validation technique. In each run, a fixed number of data instances is sampled (i.e., without replacement) as the test dataset, and the rest of the data as the training dataset. Multiple runs (i.e., 3 times) are generated for each model in each set of parameters and experimental configurations. We report the average of these runs as result. Definitions of the accuracy, recall, and F1-value are given as follows, where T_P,T_N,F_P,F_N are the number of true positives, true negatives, false positives, and false negatives, correspondingly.

$$\text{Accuracy} = \frac{T_P+T_N}{T_P+T_N+F_P+F_N};\ \text{Recall} = \frac{T_P}{T_P+F_N};\ \text{F1-value} = \frac{2T_P}{2T_P+F_P+F_N}$$

Experiment questions: Our task of validation concerns three key issues: **(1)** Which parameter configurations are optimal for the baseline models on the seed dataset, i.e., SVM, RF, and NB? **(2)** Which self-taught learning model is the best in terms of accuracy, recall, and F1-value, given the 6,794 annotated tweets and the three million unlabeled tweets? and **(3)** Which vectorization setting is more effective? To address these concerns, our series of experiments are as follows.

2.2.4 Experimental Results

Experiment on seed dataset with baseline models Figure 2.3 illustrates the accuracy, recall, and F1-value of each algorithm with different parameter configurations, i.e., term frequency *tf*, *tf-idf*, and *Word2vec*, on the (annotated) seed dataset. The term "*custom*" is used to indicate the Word2vec embedding trained with our own drug abuse-related tweets, compared with the pre-trained Google News Word2vec embedding, denoted as "*google*." It is clear that the SVM model using the custom-trained Word2vec embedding achieves the best and the most balanced performance in terms of all three measures, i.e., accuracy, recall, and F1-value, at approximately 67%. Other configurations usually have a lower recall, which suggests that the decisions they make bias towards the major class, i.e., non-drug abuse risk behavior tweets. From the angle of classifiers, SVM model achieves the best overall performance. Random Forest has slightly less average accuracy than the SVM model, but worse recall and F1-value. Furthermore, from the view of vectorization approach, it is clear that Word2vec embedding outperforms term frequency and *tf-idf* in most of the cases. Several possible combinations of settings are not shown in Figure 2.3 due to poor performances.

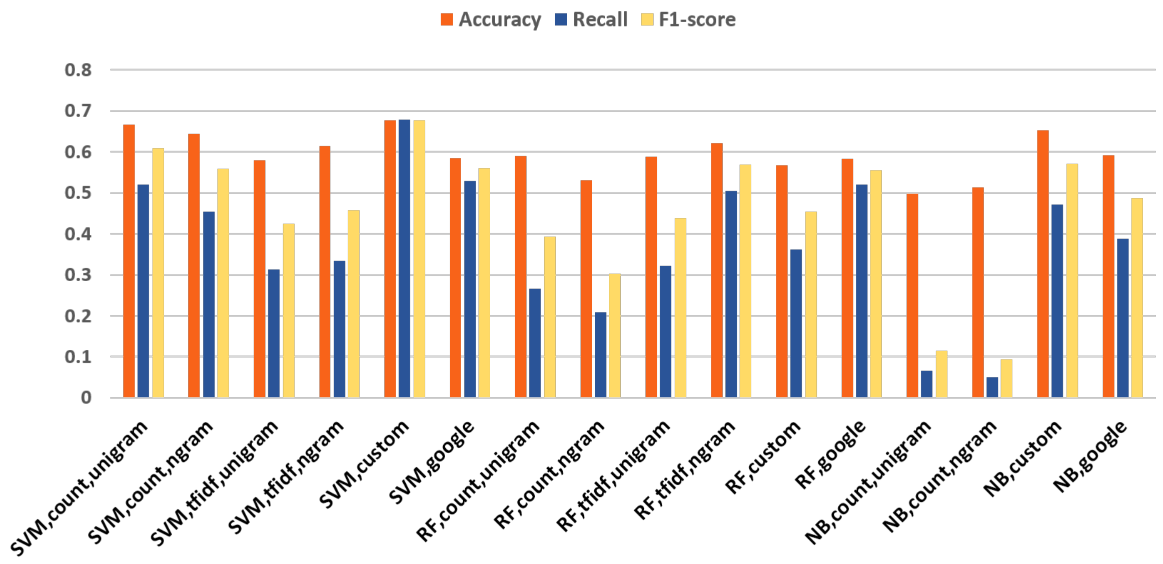


Figure 2.3 Accuracy, Recall, and F1-value of each baseline model on the seed dataset.

Experiment on self-taught learning models As shown in the previous experiment, SVM model using the custom-trained Word2vec embedding achieves the best performance, we decided to apply the same model structure to compare with our deep self-taught learning approaches. In this experiment, at each epoch, 10,000 machine-labeled tweets were randomly sampled and merged into the training set. Figure 2.4 shows the experimental results of the five self-taught models, including self-taught CNN (**st-CNN**), self-taught LSTM (**st-LSTM**), self-taught SVM (**st-SVM**), self-taught NB (**st-NB**), and self-taught RF (**st-RF**). All configurations of classifiers and vectorization methods are tested. For the sake of clarity, we only illustrate the best-performing setting for each model in Figure 2.4. It is clear that our proposed deep self-taught learning approaches (i.e., st-LSTM and st-CNN) outperform traditional models, i.e., st-SVM, st-NB, and st-RF, in terms of accuracy, recall, and F1-value, in all cases. Deep learning models achieve 86.53%, 88.6%, and 86.63% in terms of accuracy, recall, and F1-value correspondingly.

Experiment on vectorization settings The impact of two different Word2vec representations on the st-CNN, i.e., the custom Word2vec embedding we trained from our corpus,

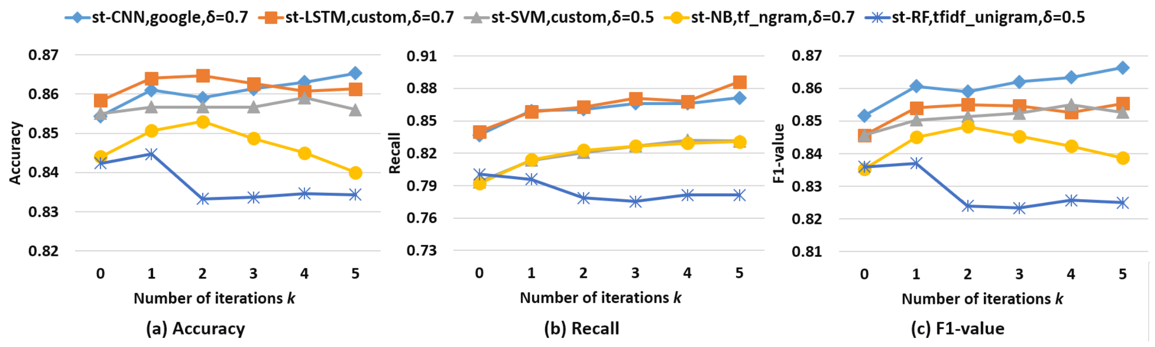


Figure 2.4 Accuracy, Recall, and F1-value of the five self-taught learning models, including st-CNN, st-LSTM, st-SVM, st-NB, and st-RF.

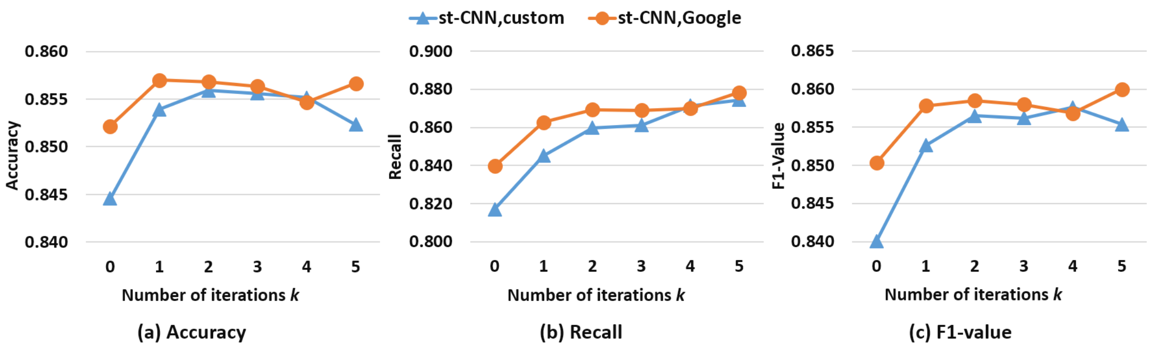


Figure 2.5 Performance comparison between custom Word2vec embedding and Google News Word2vec embedding.

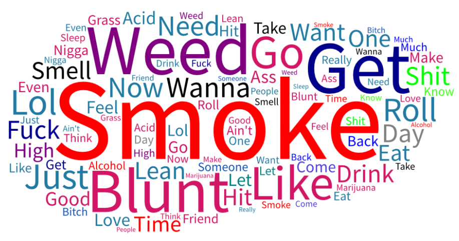
and pre-trained Google News Word2vec embedding, is shown in Figure 2.5. The Google News Word2vec achieves 0.1%, 0.4%, and 0.3% improvements in terms of accuracy, recall, and F1-value (86.63%, 89%, 86.83%, respectively) compared with the custom trained Word2vec embedding. In addition, it is clear that Google News Word2vec embedding outperforms the custom trained Word2vec in most of the cases. This is because the Google News Word2vec embedding was trained on a large-scale corpus, which is significantly richer in contextual information, compared with our short, noisy, and sparse Twitter datasets.

2.2.5 An Insight Analysis of Drug Abuse Risk Behavior on Twitter

To gain insights in drug abuse risk behaviors on Twitter, we use our best performing deep self-taught learning model to annotate over three million drug abuse-related tweets with geo-tags and perform quantitative analysis. There are 117,326 tweets classified as positive, and 3,077,827 tweets classified as negative. The positive tweets correspond to 3.67% of the whole dataset. We performed analysis from three aspects: word and phase distributions, temporal distributions, and spatial distributions.

Word and phase distributions We first visualize the top frequent words by word cloud, as shown in Figure 2.6. The word distribution in positive tweets (Figure 2.6(a)) is remarkably different from word distribution in negative tweets (Figure 2.6(b)). In fact, drug abuse tweets usually consist of abuse-indicating terms, and drug names, such as "blunt," "high," "smoke," "weed," "marijuana," "grass," "juic," etc., (Figure 2.6(a)). In addition, the high concentration of dirty words, e.g., "s**t," "f**k," "as*," "bit**," etc., clearly suggests the expression patterns that the drug abusers may have (Figure 2.6(a)). This expression pattern does not likely exist in negative tweets. Then, we further show the comparison of normalized word frequency between positive tweets and negative tweets (words from positive tweets got normalized by the number of positive tweets, and negative words by negative tweets), regarding the 25 most frequent words in positive tweets (Figure 2.7) and 25 most frequent words in negative tweets (Figure 2.8). Note that in Figure 2.7, the y-axis is clipped at 0.25, which is the value of word "weed", while the word "smoke" has the normalized frequency of 0.44. These two figures further show that: (1) Positive-frequent words are more likely to have lower normalized frequency in negative tweets, and vise-versa; and (2) Some ordinary words, i.e., "go", "want", "day", and "good", still share similar normalized frequency between positive and negative tweets.

Hashtags also play an import role in the Twitter sphere as a way for users to: (1) To express their opinion more clearly; and (2) To improve information sharing


(a) Positive tweets

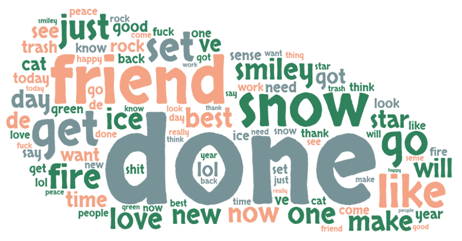

(b) Negative tweets

Figure 2.6 Word frequency distribution of the classification results.

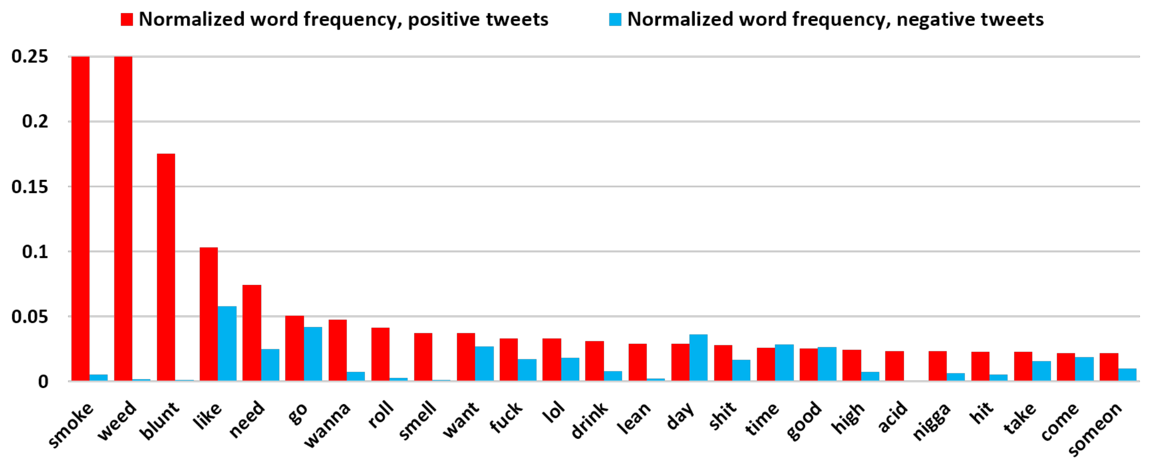


Figure 2.7 Normalized frequency of top 20 frequent words in positive tweets, compared with in negative tweets.

efficiency. Tweets that share same Hashtags can be grouped together and easily found, while popular Hashtags can make the tweets more visible to wider audience. Table 2.6 shows the most frequent Hashtags in positive tweets and negative tweets. It is clear that the Hashtags in positive tweets are almost exclusively related to drug abuse, while the Hashtags in negative tweets cover much wider range of topics.

Finally for word and phase analysis, we extract the co-occurrence frequencies of combinations of drug name and drug abuse behavior. For each combination, we count the number of positive tweets and negative tweets that contain all words in that combination, then sort it by the absolute difference of normalized frequency between positive tweets and negative tweets. Table 2.7 shows the top 25 observed combinations. The "*Relative_ratio*"

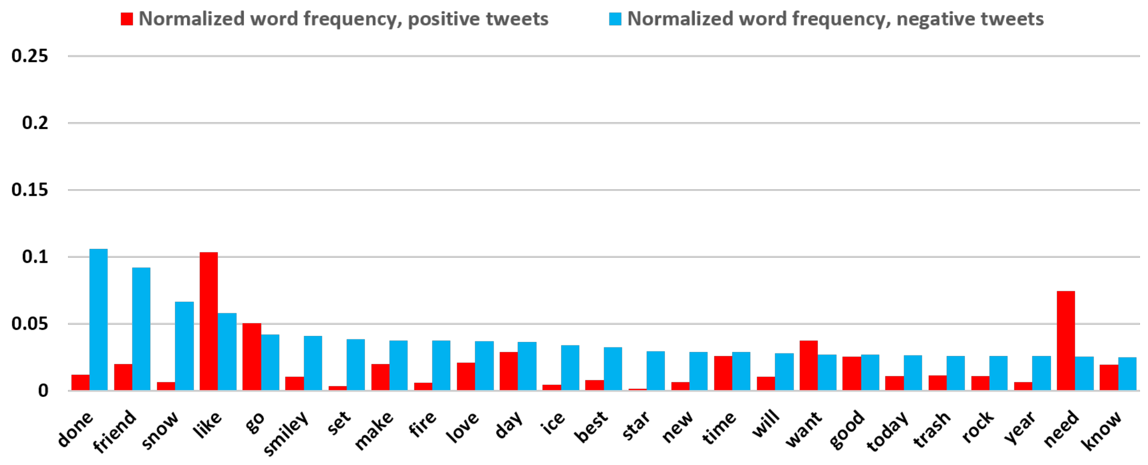


Figure 2.8 Normalized frequency of top 20 frequent words in negative tweets, compared with in positive tweets.

Table 2.6 Most Frequent Hashtags in Positive Tweets and Negative Tweets

Positive tweets	#weed, #smoke, #cannabis, #marijuana, #glassofig, #scientificglass, #WeedFirm, #maryjane, #dabs, #kush, #3wordsbetterthanIloveyou, #MarijuanaFunFacts, #pot, #dank, #high, #thc, #stoner, #blunt, #highlife, #AcademyAward, #OscarNominations, #ganja, #waterpipes, #np, #herblife
Negative tweets	#job, #snow, #hiring, #photo, #traffic, #CareerArc, #NBAVote, #Simon, #winter, #jobs, #Hospitality, #peace, #WomensMarch, #love, #Toronto, #Trump, #STAR, #nowplaying, #Orlando, #AZ, #np, #Veterans, #Retail, #SoundCloud, #nyc, #Inauguration, #cat, #weather, #MAGA

column is showing the ratio that the combination appears in positive tweets over the appears in all tweets. This analysis spots the more frequently used drug abuse risk behavior indication word combinations, which will support further data collection.

Temporal analysis To examine if there are different time patterns for positive tweets and negative tweets to be posted, we extract the local posting time of each tweet, then perform

Table 2.7 Drug Name and Abuse Behavior Co-occurrence Frequency Differences Between Positive Tweets and Negative Tweets

Combo	Pos_count	Neg_count	Ratio_diff	Relative_ratio
trash high	1131	1387	0.9189%	1166.04%
acid trip	547	239	0.4585%	1863.66%
acid drop	256	168	0.2127%	1603.13%
glass amp	374	3472	0.2060%	171.11%
acid take	222	167	0.1838%	1509.61%
lean amp	280	2391	0.1610%	192.55%
coke high	195	186	0.1602%	1343.14%
coke take	185	512	0.1410%	646.57%
lean hit	180	745	0.1292%	446.33%
acid amp	162	367	0.1262%	761.96%
molly pop	160	328	0.1257%	823.11%
acid hit	132	138	0.1080%	1278.34%
lean pop	121	118	0.0993%	1327.49%
acid use	115	238	0.0903%	817.21%
shrooms trip	105	55	0.0877%	1751.49%
lean high	108	136	0.0876%	1147.54%
lean use	159	1479	0.0875%	170.62%
blow high	125	675	0.0846%	337.93%
upper high	112	382	0.0830%	537.16%
dope high	106	509	0.0738%	383.46%
coke amp	137	1413	0.0709%	146.07%
acid high	65	57	0.0535%	1402.44%
coke snort	82	571	0.0513%	251.20%
molly amp	88	777	0.0498%	183.80%
crack hit	108	1360	0.0479%	104.18%

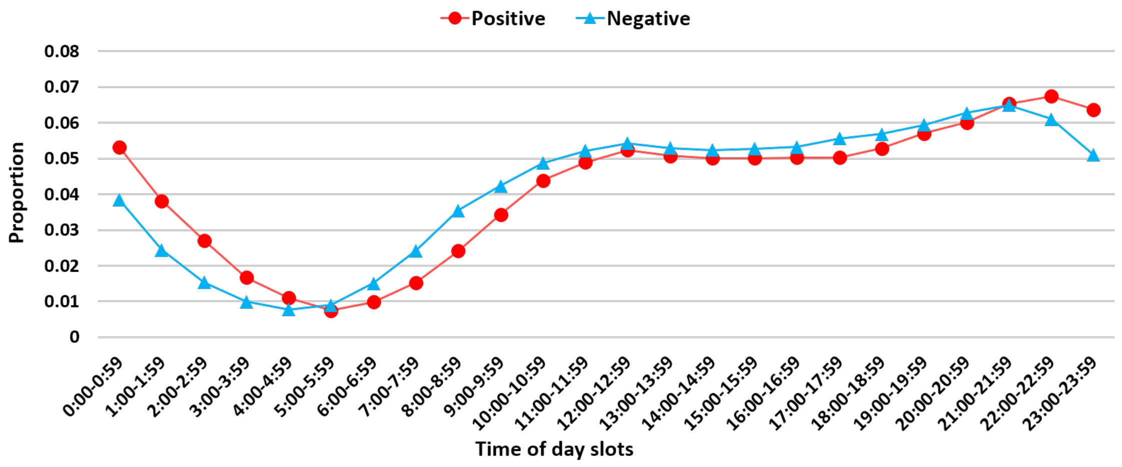


Figure 2.9 Time of day distribution comparison between positive tweets and negative tweets.

1-hour-interval binning. As shown in Figure 2.9, where x-axis are time slots, and y-axis is the proportion (normalized count) of tweets. The results shown in Figure 2.9 are very interesting. The time patterns are obviously different between positive tweets and negative tweets. In fact, the chi-square test results on the data in Figure 2.9 shown in Table 2.8 clarifies that the pattern differences are significant for the time frames of 'All day' and 'Night time.' This result shows a very plausible phenomenon that tweets with drug abuse risk behaviors are more active in night time than in day time.

Table 2.8 Chi-Square Test of Time of Day Distribution

Type	Chi square	P-value(95%)
All day	46.467257	0.002615305***
Day time (8 A.M. to 6 P.M.)	6.87318202	0.650321116
Night time (6 P.M. to 8 A.M.)	39.5940753	0.000160637***

Spatial analysis The geo-location information tags in tweets are very useful for capturing the distribution of drug abuse risk behaviors. The geo-tagging information on Twitter usually comes in two forms: GPS coordinates, or a "Place Object" associated with the

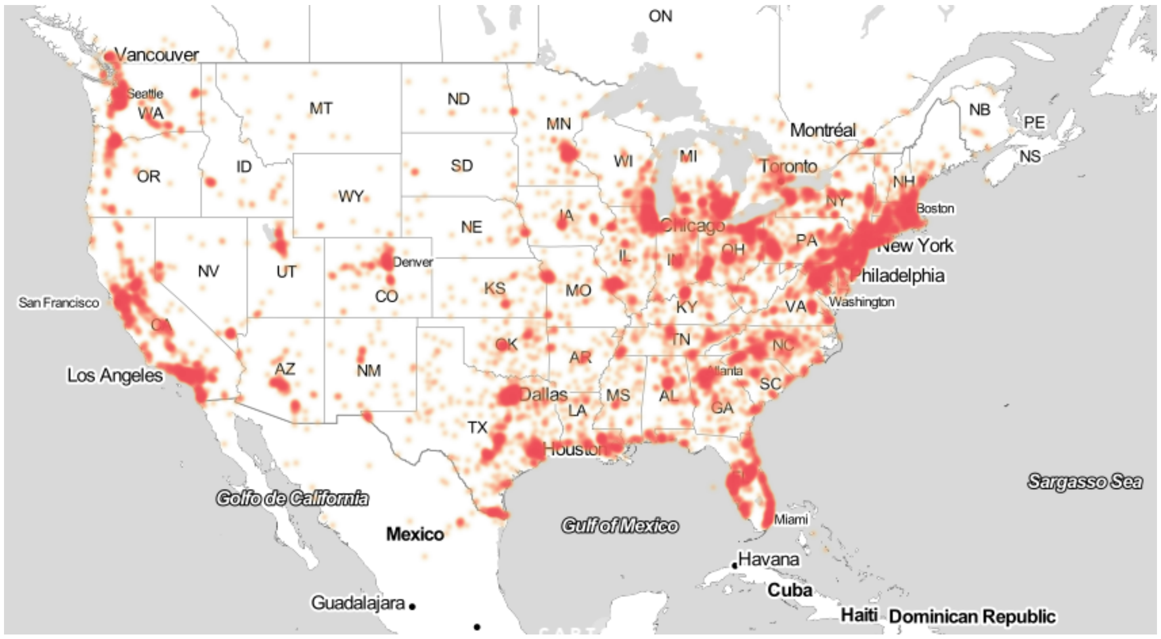


Figure 2.10 Dot map of positive tweets across the United States.

tweet. We first visualize geo-distribution of the positive tweets by plotting each geo-tag across the continental United States in Figure 2.10. By making this fine granularity, we can confirm that the collected tweets generally follow the population distribution. Then, we aggregate the geo-tags into state level, normalized with state's population of age group 12 or older, and draw the Figure 2.11 with the numbers scaled to [-1,1]. From Figure 2.11, we can see that the District of Columbia has an extremely high ratio of positive tweets, follow by Louisiana, Texas, and Nevada that have relative high rate. Other states with high rate including California, Georgia, Maryland and Delaware. Furthermore, the distribution of other states' data showing that the collected tweets align relatively well with state level population distribution.

The other spatial analysis we perform is the alignment between our state level counts of positive tweets, normalized with state population, and the 2016-2017 National Survey on Drug Use and Health (NSDUH) survey data. Here the normalization is meant to decorrelate the count of tweets from the population of each state, and is done by simply dividing the count of positive tweets by the population (2017 census estimation) for each state. We

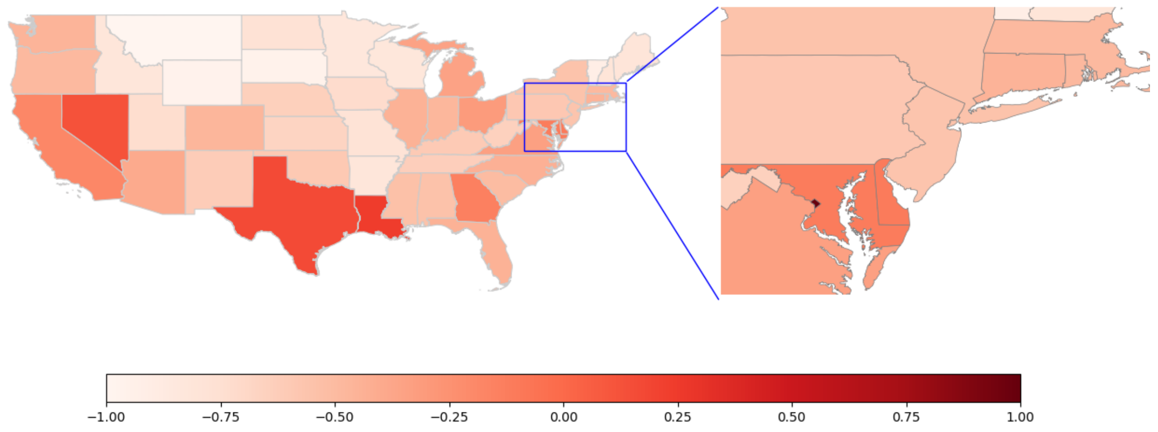


Figure 2.11 Number of positive tweets, per state, normalized by state population of 12 or older.

choose to perform normalization with population for two reasons: (1) We have little to no control of the sampling process, in terms of geo-location distribution, when crawling data from Twitter, which means the bias is unavoidable and uncontrollable; and (2) Thus the state population figures are more reliable, stable, and representative. NSDUH is a creditable source of drug abuse-related population scale estimation. If our Twitter data can align with the reliable survey data, we can argue that the Twitter based studies have the prediction power that should not to be ignored. By computing the Pearson's R between the normalized number of tweets and the NSDUH prevalence rate, over the same age group (12 or older), it is surprising to find that in our study even without further categorization, the Twitter data is significantly correlated ($p < 0.05$) with some of the most important categories in the NSDUH study: (1) "Illicit Drug Use Other Than Marijuana in the Past Month" ($r = 0.387$); (2) "Cocaine Use in the Past Year" ($r = 0.421$); (3) "Methamphetamine Use in the Past Year" ($r = -0.372$); (4) "Pain Reliever Use Disorder in the Past Year" ($r = -0.375$); and (5) "Needing But Not Receiving Treatment at a Specialty Facility for Illicit Drug Use in the Past Year" ($r = 0.336$). We argue that, when large quantity of Twitter data is available, we can perform more detailed and creditable studies on the population scale.

2.2.6 Discussion and Limitations

According to our experimental results, our deep self-taught learning models achieved promising performance in drug abuse risk behavior detection in Twitter. Many assumptions call for further experiments. First, how to optimize the classification performance by exploring the correlations among parameters and experimental configurations. For instance, for SVM and RF models, unigram feature works better than n-gram feature on term frequency; however, for *tf-idf*, it is the opposite situation. Second, the pre-trained Google News Word2vec embedding performs better than the custom-trained Word2vec embedding may also be situational. These findings indicate the necessity of leveraging size and quality of the training data for training word embedding, given that the available data may better fit the classification task but be short in quantity. Nevertheless, among the measures, recall receives a more significant boost than accuracy and F1-value. We may argue that the proposed self-taught algorithm helped correcting the bias in the classifiers caused by the imbalanced nature of the training dataset. More experiments need to be conducted to verify this interesting point.

2.2.7 Future Research

The study we presented in this dissertation can be improved in many ways. Here we elaborate several of the future research directions. Firstly, we plan to incorporate the well-trained classifier into a real-time drug abuse risk behavior monitoring and analysis system that aims at providing community-level stakeholders with timely accessible detection results for supporting their efforts, such as recovery services and public educations, on combating the opioid crisis. Secondly, we can utilize more information that can be extracted from tweets, such as user tweeting history, user demographic attributes, and user interactions, to further improve the model in terms of performance, scope, and credibility. Thirdly, the extra information that we extract further enables the analysis of connections among users and tweets, on both social network plane and geo-spatial network plane, which can help to acquire knowledge regarding how the drug trend propagates through both planes. Last but not least, we may expand the study to other major online social media platforms, i.e., Reddit and Instagram, and more specialized online forum Blulelight.

2.2.8 Conclusion

In this study, we proposed a large-scale drug abuse risk behavior tweet collection mechanism based on supervised machine learning and data crowd-sourcing techniques. Challenges came from the noisy and sparse characteristics of Twitter data, as well as the limited availability of annotated data. To address this problem, we propose deep self-taught learning algorithms to improve drug abuse risk behavior tweet detection models by leveraging a large number of unlabeled tweets. An extensive experiment and data analysis were carried out on three million drug abuse-related tweets with geo-location information, to validate the effectiveness and reliability of our system. Experimental results shown that our models significantly outperform traditional models. In fact, our models correspondingly achieve 86.53%, 88.6%, and 86.63% in terms of accuracy, recall, and F1-value. This is a very promising result, which significantly improves upon the state-of-the-art results.

Further data analysis gain insights into the expression patterns and the geo-distribution that the drug abusers may have on Twitter. For example, the words and phrases used in drug abuse risk behavior-positive tweets have distinctive frequencies that can be used in data collection to improve the quality of raw data. The uneven geographical distribution of tweets makes it appealing to perform further analysis that associates tweets with other geographical data.

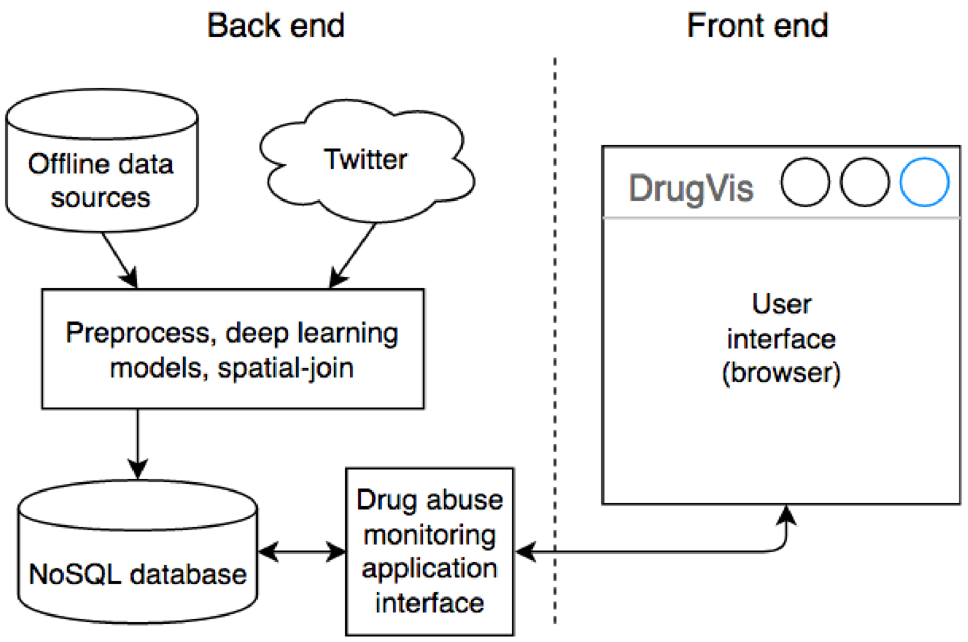

Back end
Front end
Offline data sources
Twitter
Preprocess, deep learning models, spatial-join
NoSQL database
Drug abuse monitoring application interface
DrugVis
User interface (browser)

Figure 2.12 Basic architecture of the DrugTracker system.

2.3 DrugTracker: A Community-focused Drug Abuse Monitoring and Supporting System using Social Media and Geospatial Data

2.3.1 DrugTracker System

We opt to implement DrugTracker as a web-based visualization system, since web-based systems are more flexible and requires virtually no setup process for end-users. Our system (Figure 2.12) includes two major parts: **(1)** The back-end, which runs on a server and provides data services, including data collection, data pre-processing, deep learning models for drug abuse risk behavior detection, and data management; and **(2)** The front-end, which runs on web browsers to provide interactive User Interfaces (UIs), for making queries and visualizing analysis results.

Back-end Services The back-end does the heavy lifting in the system, which runs a complete pipeline of collecting and processing data coming from social media and other sources (e.g., census data). There are three major modules in the back-end, including the data collection module, the data processing module, and the data management module.

Table 2.9 Geo-tag Types and Frequency

Geo Type	Sub-type	Percentage
Place	Country	0.29%
	Admin	13.63%
	City	70.69%
	Neighborhood	0.32%
	POI	1.06%
Coordinates	-	14.01%

For the data collection, we use tweets as a major source of geo-tagged social media data for its availability and abundance. We collect tweets through the publicly available Streaming API [147] using a typical keyword-based crawler well-integrated with trained deep learning models (i.e., CNN and LSTM models) designed to detect drug abuse risk behaviors in tweets [49, 148]. In our previous work [49, 148], we built our human labeled drug abuse risk behavior dataset, and demonstrated that a deep learning model, which was trained with both labeled data and large number of unlabeled data, can achieve state-of-art classification performance (86.63% of Accuracy, 89% of Recall, 86.83% of F1-value) on our dataset. The module is able to continuously collect newest tweets, to feed tweets to deep learning models, and to update the system with live data, so that the drug trend can be tracked and analyzed in near real-time.

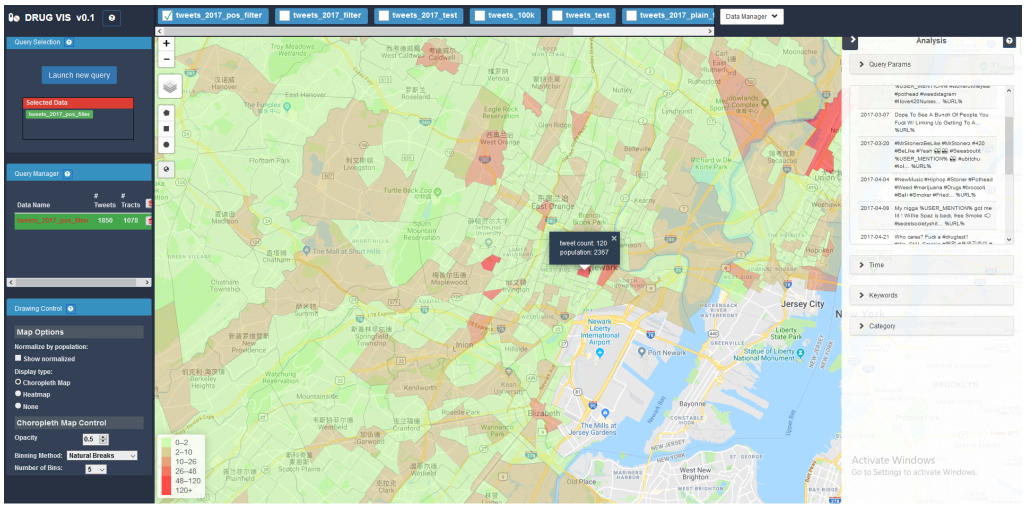
The use of geo-tags in tweets is not straight forward. Statistics of the 2017 dataset, as shown in Table 2.9, tell us that there are two major types of available geo-tags: '*place*' Object and '*coordinates*' Object. The '*coordinates*' Objects are GPS points that comes from Twitter users who have location service turned on, while each '*place*' Object refers to a named place entity that the tweet is associated with (but not necessarily originating from) [147]. However, the '*place*' Objects have several types but the resolution/granularity of each Object of same type may vary. In this demonstration, we opt to only use '*coordinates*' as geo-tags, as they provide the highest location precision with an adequate quantity of data points.

For offline data, such as geographical data and demographic data, we use publicly available data from trusted sources (e.g., Census Bureau). A work-flow is built to pre-join the tweets data with offline data as a pre-processing step for performance reason. In our system, we first collect Census Tract data, in the format of Shapefile, and population census data, in the form of spread-sheets. Then we joined (table-join and spatial-join) these data with tweets, so that each tweet record in our database is associated with offline data.

The pre-processed tweets data is stored and managed in a NoSQL database (e.g., MongoDB). For the fields in the original tweet objects, only those fields that are used by the front-end are stored. Several further pre-processing steps are preformed, including: **(1)** The time-stamp of each tweet, which is in UTC, is converted to local time using geo-location information; **(2)** Re-identification information in each tweet's texts, i.e., User Mention and URL, are removed; and **(3)** The sub-category of each tweet is extracted by identifying drug abuse-related keywords. Indexes are created for the fields of time-stamp, text, keywords, and geo-location to support fast text queries and spatial queries. Python and PHP scripts are served as the interface between the back-end and the front-end that execute queries and generate responses to the front-end.

Front-end Interactive Visualization The front-end is built based on the open-sourced NeighborVis System [149]. The basic UI layout and some components are inherited, while new functions are added to incorporate the different functions offered in our system. The front-end is a dynamic web page constructed with HTML and Javascript. Javascript framework Leaflet is used for the core mapping functionality. Other frameworks, such as Heatmap-js and D3 are used for drawing heatmap and charts, respectively.

One noticeable change we made to the system is that the tweets displayed to the users in the front-end are always aggregated into some desired form instead of individually.



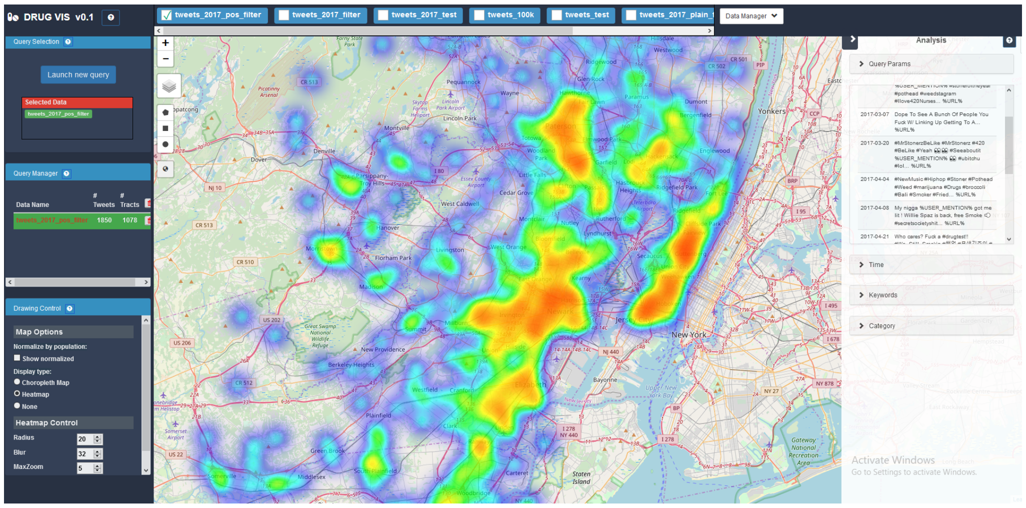
Data Name	# Tweets	# Tracts
tweets_2017_pos_filter	1850	1078

Map Options
Normalize by population:
☐ Show normalized
Display type:
[✓ Choropleth Map]
Heatmap
None

Choropleth Map Control
Opacity: 0.5
Binning Method: Natural Breaks
Number of Bins: 5

	%USER_MENTION% #stoneroftheyear #pothead #weedstagram #Ilove420Nurses... %URL%
2017-03-07	Dope To See A Bunch Of People You Fuck W/ Linking Up Getting To A... %URL%
2017-03-20	#MrStonerzBeLike #MrStonerz #420 #BeLike #Yeah 👀👀 #Seeaboutit %USER_MENTION% 👀 #ubitchu #lol... %URL%
2017-04-04	#NewMusic #Hiphop #Stoner #Pothead #Weed #marijuana #Drugs #broccoli #Bali #Smoker #Fried... %URL%
2017-04-08	My nigga %USER_MENTION% got me lit ! Willie Spaz is back, free Smoke 👈 #secretsocietyshit... %URL%
2017-04-21	Who cares? Fuck a #drugtest!!

(a) Choropleth



Data Name	# Tweets	# Tracts
tweets_2017_pos_filter	1850	1078

Map Options
Normalize by population:
☐ Show normalized
Display type:
Choropleth Map
[✓ Heatmap]
None

Heatmap Control
Radius: 20
Blur: 32
MaxZoom: 5

	%USER_MENTION% #stoneroftheyear #pothead #weedstagram #Ilove420Nurses... %URL%
2017-03-07	Dope To See A Bunch Of People You Fuck W/ Linking Up Getting To A... %URL%
2017-03-20	#MrStonerzBeLike #MrStonerz #420 #BeLike #Yeah 👀👀 #Seeaboutit %USER_MENTION% 👀 #ubitchu #lol... %URL%
2017-04-04	#NewMusic #Hiphop #Stoner #Pothead #Weed #marijuana #Drugs #broccoli #Bali #Smoker #Fried... %URL%
2017-04-08	My nigga %USER_MENTION% got me lit ! Willie Spaz is back, free Smoke 👈 #secretsocietyshit... %URL%
2017-04-21	Who cares? Fuck a #drugtest!!

(b) Heatmap

Figure 2.13 Mapping Area: (a) Choropleth and (b) Heatmap.

This is done for two reasons: **(1)** To protect the sensitive information that the tweets and the classification results contain; and **(2)** To enable the displaying and analysis of tweets at population scale, as it is impractical to show millions of tweets on the map. The aggregation can be done on different types of administrative regions or entities, depending on the users' needs. For demonstration, we aggregate our data into Census Tract level. The UI of our system is demonstrated in Figures 2.13-2.15. The visualization layout is divided into three sections from left to right: **(1)** Query Management; **(2)** Mapping Area; and **(3)** Statistical Information.

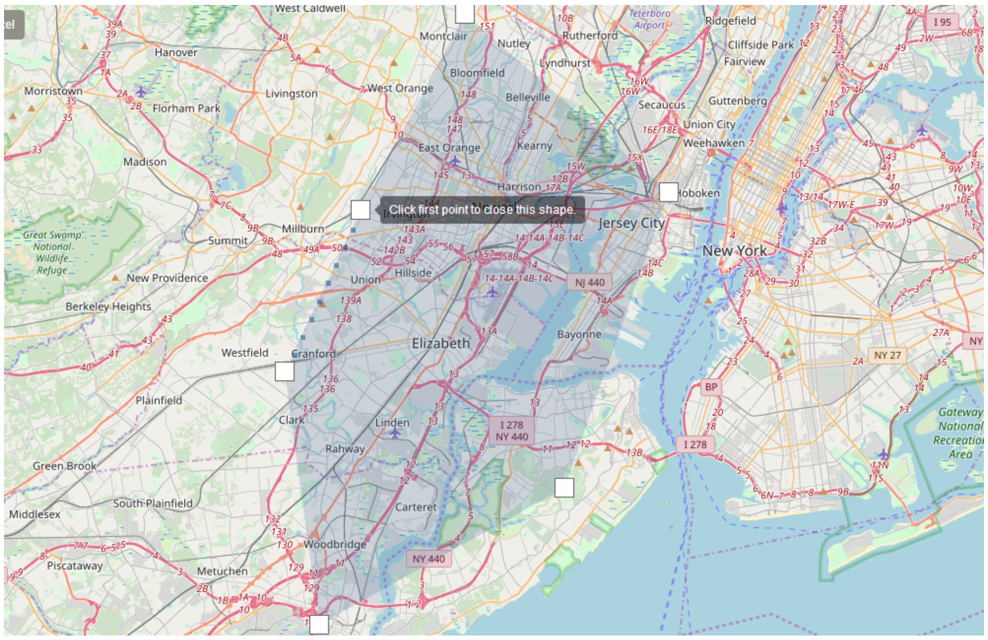
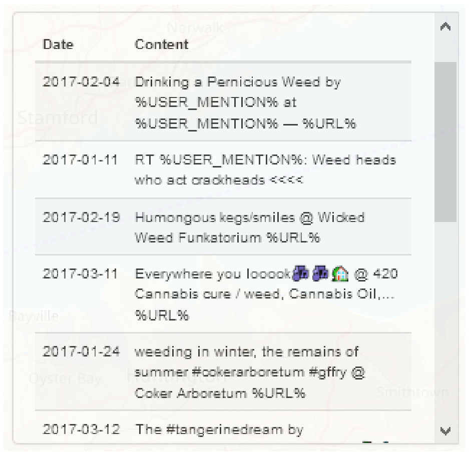


Figure 2.14 An example of polygon-shaped query area.

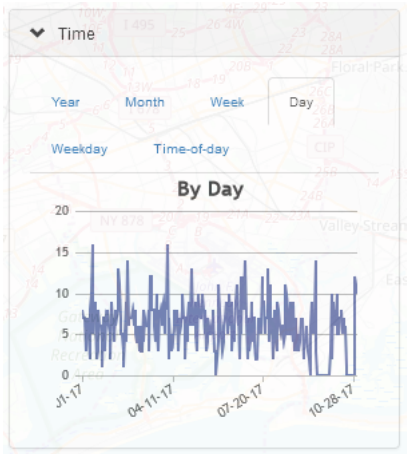
Query Management. To begin, users should provide a query by clicking a button on the left panel to launch a query interface (Figure 2.13). A query has two mandatory constraints: temporal constraint and spatial constraint, and one optional constraint: content constraint. The temporal constraint limits the local posting-time of tweets and is specified by a start and an end date. The spatial constraint limits the location where the tweets are from and can be either a user defined area (optional shapes are circle, rectangle, and polygon Figure 2.14), or a list of states. The actual query area has a minimum granularity (Census Tract level in demonstration) to prevent the disclosure of individual tweet's location. The content constraint can be a list of keywords and phrases, e.g., "get high," "smoke blunt," etc. Users can also query multiple datasets at once to improve efficiency.

Once the query is submitted, the back-end will process the query and send aggregated results to front-end for displaying in the mapping area. Our system provides two basic mapping options: Choropleth and Heatmap (Figure 2.13). Choropleth displays the number of tweets that match the query within each area with a color mapping from green (low) to red (high). Jenks natural breaks is the default classification (binning) method. The choropleth

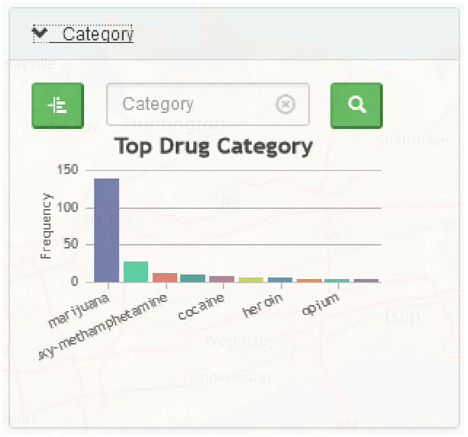


Date	Content
2017-02-04	Drinking a Pernicious Weed by %USER_MENTION% at %USER_MENTION% — %URL%
2017-01-11	RT %USER_MENTION%: Weed heads who act crackheads <<<<
2017-02-19	Humongous kegs/smiles @ Wicked Weed Funkatorium %URL%
2017-03-11	Everywhere you loooook @ 420 Cannabis cure / weed, Cannabis Oil,... %URL%
2017-01-24	weeding in winter, the remains of summer #cokerarboretum #gffry @ Coker Arboretum %URL%
2017-03-12	The #tangerinedream by

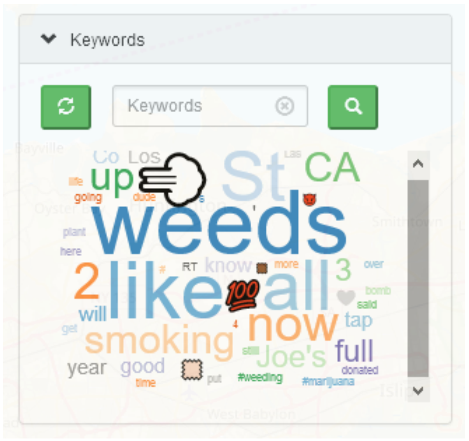
(a) Tweet samples



(b) Temporal analysis



(c) Category frequency



(d) Word cloud

Figure 2.15 Analysis functions: (a) Tweet samples, (b) Temporal analysis, (c) Category frequency, and (d) Word cloud.

can also display the tweets data conjugated with selected offline data (e.g., normalize the number of tweets by the population in each Census Tract). Heatmap provides a way to view the data in a more analog way, by showing the density in a spectrum of colors from blue (low) to red (high). On the left-side panel, users can select which dataset and query to view, and can fine-tune the parameters of the generated maps (e.g., opacity, number of bins, and heatmap intensity).

There is a collapsible panel (Figure 2.15) on the right consisting of five sub-panels, each of which presents a set of information corresponding to the current query that aids the analysis, including: **(1)** Query information, shows the parameters that are used for this query; **(2)** Random samples of tweets in the query (Figure 2.15a); **(3)** Temporal analysis of different scopes, including by year, by month, by week, by day, by weekday, and by hour-of-day (Figure 2.15b); **(4)** Word cloud of most popular words in among the tweets in the query (Figure 2.15d); and **(5)** Bar chart of frequency of each category (e.g., type of drug mentioned in each tweet) (Figure 2.15c). If the user is interested in more detailed analysis, by clicking the items in these panels, e.g., a keyword in (Figure 2.15d), and a category in (Figure 2.15c), a sub-query with updated parameters will be launched and new results will be displayed. The user can easily switch between queries to compare them.

2.3.2 Conclusion and Future Work

In this study, we developed a community-focused drug abuse monitoring and supporting system, called DrugTracker, using social media and geospatial data in near real-time. Our DrugTracker system provides vital source of information when combating the drug abuse epidemiology, and proposed a function rich visualization system that can help local communities and organizations being informed about drug trends, locating drug abuse hot-spots, and reaching online users who may in need for help.

Some future work can be done on the proposed system. Here we just name a few: (1) Integrates more varieties of offline geospatial data that fits the needs of different aggregation levels; (2) Integrates more advanced privacy preserving methods that enables more detailed analysis with lower risk of unwanted leak of privacy; and (3) Further enrich the system with social connections (e.g., following, user mention) to enable the association of social connection information with geospatial data that aids the analysis.

CHAPTER 3

DIFFERENTIAL PRIVACY IN DEEP LEARNING WITH CERTIFIED ROBUSTNESS BOUNDS

Chapter Abstract Today, deep learning has become the tool of choice in many areas of engineering, such as autonomous systems, signal and information processing, and data analytics. Deep learning systems are, therefore, not only applied in classic settings, such as speech and handwriting recognition, but also progressively operate at the core of security and privacy critical applications. The pervasiveness of machine learning exposes new vulnerabilities in software systems, in which deployed machine learning models can be used (a) to reveal sensitive information in private training data [64], and/or (b) to make the models misclassify, such as *adversarial examples* [65]. In this subsection, we present our effort in three aspects of privacy preserving in deep learning: (1) The developing a novel mechanism to preserve differential privacy in deep neural networks, such that:**1.** The privacy budget consumption is totally independent of the number of training steps; **2.** It has the ability to adaptively inject noise into features based on the contribution of each to the output; and **3.** It could be applied in a variety of different deep neural networks; (2) The preserving of differential privacy (**DP**) in lifelong learning (**L2M**); and (3) The developing of a scalable algorithm to preserve differential privacy (**DP**) in adversarial learning for deep neural networks (**DNNs**), with certified robustness to adversarial examples.

3.1 Adaptive Laplace Mechanism: Differential Privacy Preservation in Deep Learning

3.1.1 Preliminaries and Related Work

In this subsection, we revisit differential privacy, existing techniques in preserving differential privacy in deep learning, and the Layer-wise Relevance Propagation (LRP) algorithm [98].

Let D be a database that contains n tuples $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ and d+1 attributes $X_1, X_2, \dots, X_d, Y$, and for each tuple $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id}, y_i)$. We assume, without loss of generality, $\sqrt{\sum_{j=1}^{d} x_{ij}^2} \leq 1$ where $x_{ij} \geq 0$. This assumption can be easily enforced by changing each x_{ij} to $\frac{x_{ij}-\alpha_j}{(\beta_j-\alpha_j)\cdot\sqrt{d}}$, where α_j and β_j denote the minimum and maximum values in the domain of X_j.

To be general, let us consider a classification task with M possible categorical outcomes, i.e., the data label y_i given $\mathbf{x}_i \in L$ is assigned to only one of the M categories. Each y_i can be considered as a vector of M categories $y_i = \{y_{i1}, \dots, y_{iM}\}$. If the l-th category is the class of $\mathbf{x}_i$, then $y_{il} = 1$, otherwise $y_{il} = 0$. Our objective is to construct a differentially private deep neural network from D that (i) takes $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})$ as input and (ii) outputs a prediction of y_i that is as accurate as possible. To evaluate whether model parameters θ lead to an accurate model, a cost function $\mathcal{F}_D(\theta)$ is used to measure the difference between the original and predicted values of y_i.

3.1.2 ϵ-Differential Privacy

As the released model parameter θ may disclose sensitive information of D, to protect the privacy, we require that the model training should be performed with an algorithm that satisfies *ϵ-differential privacy*. The definition of differential privacy is as follows:

Definition 3.1. *ϵ-Differential Privacy [55]. A randomized algorithm A fulfills ϵ-differential privacy, if for any two databases D and D' differing at most one tuple, and for all $O \subseteq Range(A)$, we have:*

$$Pr[A(D) = O] \leq e^{\epsilon} Pr[A(D') = O] \tag{3.1}$$

where the privacy budget ϵ controls the amount by which the distributions induced by D and D' may differ. A smaller ϵ enforces a stronger privacy guarantee of A.

A general method for preserving ϵ-differential privacy of any function $\mathcal{F}$ (on D) is the *Laplace mechanism* [55], where the output of $\mathcal{F}$ is a vector of real numbers. In fact, the mechanism exploits the global sensitivity of $\mathcal{F}$ over any two neighboring data sets (differing at most one record), which is denoted as $GS_{\mathcal{F}}(D)$. Given $GS_{\mathcal{F}}(D)$, the Laplace mechanism ensures ϵ-differential privacy by injecting noise η into each value in the output of $\mathcal{F}(D)$: $pdf(\eta) = \frac{\epsilon}{2GS_{\mathcal{F}}(D)} exp(-|\eta| \cdot \frac{\epsilon}{GS_{\mathcal{F}}(D)})$, where η is drawn i.i.d. from Laplace distribution with zero mean and scale $GS_{\mathcal{F}}(D)/\epsilon$.

Research in differential privacy has been significantly studied, from both the theoretical perspective, e.g., [56, 150], and the application perspective, e.g., data collection [151], spatio-temporal correlations [152, 153], data streams [154], stochastic gradient descents [155], recommendation [57], regression [56], online learning [156], publishing contingency tables [157], and spectral graph analysis [158].

3.1.3 Differential Privacy in Deep Learning

Deep neural networks define parameterized functions from inputs $\mathbf{x}_i \in D$ to outputs, i.e., a prediction of y_i, as compositions of many layers of hidden neurons and nonlinear functions. For instance, Figure 3.1 illustrates a multilayer neural network, in which there are k hidden layers $H = \{\mathbf{h}_1, \ldots, \mathbf{h}_k\}$. Rectified linear units (ReLUs) and sigmoids are widely used examples of activation functions. By adjusting parameters of these neurons, such parameterized functions can be trained with the goal of fitting a finite set of input-output data instances. We specify a loss function $\mathcal{F}_D(\theta)$ that represents the penalty for mismatching

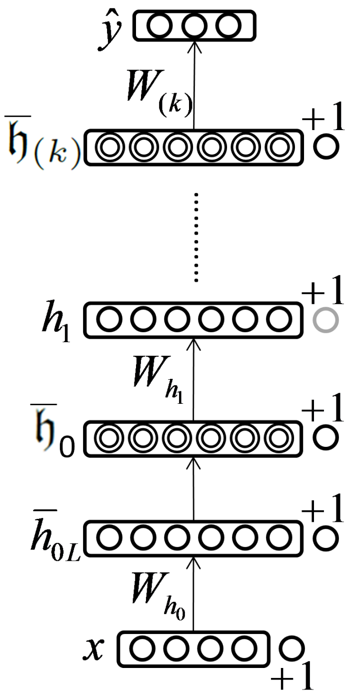


Figure 3.1 An instance of differentially private neural networks.

between the predicted and original values of y_i. $\mathcal{F}_D(\theta)$ on parameters θ is the average of the loss over the training examples $\{\mathbf{x}_1,..., \mathbf{x}_n\}$. Stochastic gradient descent (SGD) algorithm is used to minimize the cross-entropy error [39], given the model outputs and true data labels.

In the work of Abadi et al. [62], to preserve differential privacy, normal (Gaussian) distribution noise is added into the gradients $\tilde{g}$ of parameters W as follows. At each training step t, the algorithm first takes a random sample L_t with sampling probability L/n, where L is a group size and n is the number of tuples in D. For each tuple $\mathbf{x}_i \in L_t$, the gradient $\mathbf{g}_t(\mathbf{x}_i) = \nabla_{\theta_t}\mathcal{F}_{\mathbf{x}_i}(\theta_t)$ is computed. Then the gradients will be bounded by clipping each gradient in l_2 norm, i.e., the gradient vector $\mathbf{g}_t$ is replaced by $\mathbf{g}_t/\max(1, \|\mathbf{g}_t^2\|/C)$ for a predefined threshold C. Normal distribution noise is added into gradients of parameters θ as:

$$\tilde{g}_t \leftarrow \frac{1}{L}\sum_i \Big(\frac{\mathbf{g}_t(\mathbf{x}_i)}{\max(1, \frac{\|\mathbf{g}_t(\mathbf{x}_i)^2\|}{C})} + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\Big) \tag{3.2}$$

$$\theta_t \leftarrow \theta_t - \xi_t \tilde{g}_t \tag{3.3}$$

where ξ_t is a learning rate at the training step t.

Finally, differentially private parameters θ, denoted $\overline{\theta}$, learned by the algorithm are shared to the public and other parties. Overall, the algorithm introduces noise into *"gradients"* of parameters at every training step. The magnitude of injected noise and the privacy budget ϵ are accumulated in proportion to the number of training epochs.

Compared with the work in [62], the goal is similar: learning differentially private parameters $\overline{\theta}$. However, we develop a novel mechanism in which the privacy budget consumption is independent of the number of training epochs. Our mechanism is different. We redistribute the noise so that *"more noise"* will be added into features which are *"less relevant"* to the model output, and vice-versa. Moreover, we inject noise into coefficients of affine transformations and loss functions, such that differentially private parameters can be learned.

3.1.4 Layer-wise Relevance Propagation

Layer-wise Relevance Propagation (LRP) [98] is a well-accepted algorithm, which is applied to compute the relevance of each input feature x_{ij} to the model outcome $\mathcal{F}_{\mathbf{x}_i}(\theta)$. Given the relevance, denoted $R^{(k)}_m(\mathbf{x}_i)$, of a certain neuron m at the layer k, i.e., $m \in \mathbf{h}_k$, for the model outcome $\mathcal{F}_{\mathbf{x}_i}(\theta)$, LRP algorithm aims at obtaining a decomposition of such relevance in terms of messages sent to neurons of the previous layers, i.e., the layer $(k$-$1)$-th. These messages are called $R^{(k-1,k)}_{p\leftarrow m}(\mathbf{x}_i)$. The overall relevance of each neuron in the lower layer is determined by summing up the relevance coming from all upper-layer neurons:

$$R_p^{(k-1)}(\mathbf{x}_i) = \sum_{m \in \mathbf{h}_k} R_{p \leftarrow m}^{(k-1,k)}(\mathbf{x}_i) \tag{3.4}$$

where the relevance decomposition is based on the ratio of local and global affine transformations and is given by:

$$R_{p \leftarrow m}^{(k-1,k)}(\mathbf{x}_i) = \begin{cases} \frac{z_{pm}(\mathbf{x}_i)}{z_m(\mathbf{x}_i)+\mu} R_m^{(k)}(\mathbf{x}_i) & z_m(\mathbf{x}_i) \geq 0 \\ \frac{z_{pm}(\mathbf{x}_i)}{z_m(\mathbf{x}_i)-\mu} R_m^{(k)}(\mathbf{x}_i) & z_m(\mathbf{x}_i) < 0 \end{cases} \tag{3.5}$$

with: $z_m(\mathbf{x}_i)$ is the affine transformation of neuron $m \in \mathbf{h}_k$:

$$z_{pm}(\mathbf{x}_i) = p_{\mathbf{x}_i} \times W_{pm} \tag{3.6}$$

$$z_m(\mathbf{x}_i) = \sum_{p \in \mathbf{h}_k} z_{pm}(\mathbf{x}_i) + b_m \tag{3.7}$$

s.t. $p_{\mathbf{x}_i}$ is the value of neuron p given $\mathbf{x}_i$, W_{pm} is a weight connecting the neuron p to neuron m, and b_m is a bias term. A predefined stabilizer $\mu \geq 0$ is introduced to overcome unboundedness.

In Equation (3.5), in order to back propagate the relevance, we need to compute the relevance $R_m^{(k)}(\mathbf{x}_i)$ at the last hidden layer, i.e., the k-th layer, from the output layer. Given the output variable o, $R_m^{(k)}(\mathbf{x}_i)$ is computed as follows:

$$R_m^{(k)}(\mathbf{x}_i) = \begin{cases} \frac{z_{mo}(\mathbf{x}_i)}{z_o(\mathbf{x}_i)+\mu} \mathcal{F}_{\mathbf{x}_i}(\theta) & z_o(\mathbf{x}_i) \geq 0 \\ \frac{z_{mo}(\mathbf{x}_i)}{z_o(\mathbf{x}_i)-\mu} \mathcal{F}_{\mathbf{x}_i}(\theta) & z_o(\mathbf{x}_i) < 0 \end{cases} \tag{3.8}$$

Given k hidden layers $\{\mathbf{h}_1, \ldots, \mathbf{h}_k\}$, by using Equations (3.4), (3.5), and (3.8), we can compute the relevance of every hidden neuron and input feature. As in [98], the following equation holds:

$$\mathcal{F}_{\mathbf{x}_i}(\theta) = \sum_{m \in \mathbf{h}_k} R_m^{(k)}(\mathbf{x}_i) = \ldots = \sum_{x_{ij} \in \mathbf{x}_i} R_{x_{ij}}(\mathbf{x}_i) \tag{3.9}$$

where $R_{x_{ij}}(\mathbf{x}_i)$ is the relevance of the feature x_{ij} given the model outcome $\mathcal{F}_{\mathbf{x}_i}(\theta)$. To ensure that the relevance $R_{x_{ij}}(\mathbf{x}_i) \in [-1, 1]$, each $R_{x_{ij}}(\mathbf{x}_i)$ is normalized to $\frac{R_{x_{ij}}(\mathbf{x}_i) - \chi}{(\varphi - \chi)}$, where φ and χ denote the maximum and minimum values in the domain of $\left\{R_{x_{i1}}(\mathbf{x}_i), \ldots, R_{x_{id}}(\mathbf{x}_i)\right\}$.

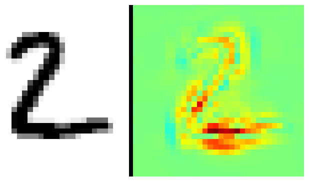


Figure 3.2 An instance of relevance of each input feature given to the classification output (MNIST dataset). Red neurons indicate stronger relevances, and green neurons indicate weaker relevances.

3.1.5 Adaptive Laplace Mechanism (AdLM)

In this subsection, we formally present our mechanism. Given a loss function $\mathcal{F}(\theta)$ with model parameters θ, the network is trained by optimizing the loss function $\mathcal{F}(\theta)$ on D by applying SGD algorithm on T random training batches consequently. At each training step, a single training batch L is used. A batch L is a random set of training samples in D with a predefined batch size $|L|$.

The pseudo-codes of Algorithm 3.1 outline five basic steps in our mechanism to learn differentially private parameters of the model. The five basic steps are as follows:

Step 1 (Lines 1-7). In the first step, we obtain the average relevances of all the j-th input features, denoted as $R_j(D)$, by applying the LRP algorithm on a well-trained deep neural network on the database D. $R_j(D)$ is computed as follows:

$$R_j(D) = \frac{1}{|D|} \sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i) \tag{3.10}$$

Then, we derive differentially private relevances, denoted as $\overline{R}_j$, by injecting Laplace noise into R_j for all the j-th input features. The total privacy budget in this step is ϵ_1.

Step 2 (Lines 8-14). In the second step, we derive a differentially private affine transformation layer, denoted $\mathbf{h}_0$. Every hidden neuron $h_{0j} \in \mathbf{h}_0$ will be perturbed by injecting adaptive Laplace noise into its affine transformation to preserve differential privacy given a batch L. Based on $\overline{R}_j$, *"more noise"* is injected into features which are *"less relevant"* to the model output, and vice-versa. The total privacy budget used in this step is ϵ_2. The perturbed affine transformation layer is denoted as $\overline{\mathbf{h}}_{0L}$ (Figure 3.1).

Step 3 (Line 15). In the third step, we stack hidden layers $\{\mathbf{h}_1, \dots, \mathbf{h}_k\}$ on top of the differentially private hidden layer $\overline{\mathbf{h}}_{0L}$ to construct the deep private neural network (Figure 3.1). The computations of $\mathbf{h}_1, \dots, \mathbf{h}_k$ are done based on the differentially private layer $\overline{\mathbf{h}}_{0L}$ without accessing any information from the original data. Therefore, the computations do not disclose any information. Before each stacking operation, a normalization layer, denoted $\overline{\mathfrak{h}}$, is applied to bound non-linear activation functions, such as ReLUs (Figure 3.1).

Step 4 (Lines 16-19). After constructing a private structure of hidden layers $\{\overline{\mathbf{h}}_{0L}, \mathbf{h}_1, \dots, \mathbf{h}_k\}$, we need to protect the labels y_i at the output layer. To achieve this, we derive a polynomial approximation of the loss function $\mathcal{F}$. Then, we perturb the loss function $\mathcal{F}$ by injecting Laplace noise with a privacy budget ϵ_3 into its coefficients to preserve differential privacy on each training batch L, denoted $\overline{\mathcal{F}}_L(\theta)$.

Step 5 (Lines 20-30). Finally, the parameter θ_T is derived by minimizing the loss function $\overline{\mathcal{F}}_L(\theta)$ on T training steps sequentially. In each step t, stochastic gradient descent (SGD) algorithm is used to update parameters θ_t given a random batch L of training samples in D. This essentially is an optimization process, without using any additional information from the original data.

In our mechanism, differential privacy is preserved, since it is enforced at every computation task that needs to access the original data D. Laplace noise is injected into our model only once, as a preprocessing step to preserve differential privacy in the computation of the relevance $\overline{R}_j(D)$, the first layer $\overline{\mathbf{h}}_{0L}$, and the loss function $\overline{\mathcal{F}}_L(\theta)$. Thereafter, the training phase will not access the original data again. The privacy budget consumption does not accumulate in each training step. As such, it is independent of the number of training epochs.

3.1.6 Private Relevance

In this subsection, we preserve differential privacy in the computation of the relevance of each j-th input feature on database D by injecting Laplace noise into $R_j(D)$. We set $\Delta_{\mathbf{R}} = \frac{2d}{|D|}$ based on the maximum values of all the relevances $R_j(D)$ (line 4, Algorithm 3.1). In lines 5-6, the relevance of each j-th input feature $R_j(D)$ is perturbed by adding Laplace noise $Lap(\frac{\Delta_{\mathbf{R}}}{\epsilon_1})$. The perturbed relevance is denoted as $\overline{R}_j$. In line 7, we obtain the set of all perturbed relevances $\overline{\mathbf{R}}(D)$:

$$\overline{\mathbf{R}}(D) = \{\overline{R}_j\}_{j\in[1,d]} \tag{3.11}$$

$$\textit{where } \overline{R}_j = \frac{1}{|D|}\sum_{\mathbf{x}_i\in D} R_{x_{ij}}(\mathbf{x}_i) + Lap(\frac{\Delta_{\mathbf{R}}}{\epsilon_1}) \tag{3.12}$$

Algorithm 3.1 Adaptive Laplace Mechanism (Database D, hidden layers H, loss function $\mathcal{F}(\theta)$, and privacy budgets ϵ_1, ϵ_2, and ϵ_3, the number of batches T, the batch size $|L|$)

```
 1: Compute the average relevance by applying the LRP Algorithm
 2: $\forall j \in [1,d] : R_j(D) = \frac{1}{|D|}\sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i)$ #Equation3.10#
 3: Inject Laplace noise into the average relevance of each $j$-th input feature
 4: $\Delta_{\mathbf{R}} = 2d/|D|$ #Lemma 3.1#
 5: for $j \in [1,d]$ do
 6:     $\overline{R}_j \leftarrow \frac{1}{|D|}\sum_{\mathbf{x}_i \in D} R_{x_{ij}}(\mathbf{x}_i) + Lap(\frac{\Delta_{\mathbf{R}}}{\epsilon_1})$
 7: $\overline{\mathbf{R}}(D) = \{\overline{R}_j\}_{j \in [1,d]}$
 8: Inject Laplace noise into coefficients of the differentially private layer $\mathbf{h}_0$
 9: $\Delta_{\mathbf{h}_0} = 2\sum_{h \in \mathbf{h}_0} d$ #Lemma 3.3#
10: for $j \in [1,d]$ do
11:     $\epsilon_j \leftarrow \beta_j \times \epsilon_2$ #Equation (3.18)#
12: for $\mathbf{x}_i \in D, j \in [1,d]$ do
13:     $\overline{x}_{ij} \leftarrow x_{ij} + \frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_j})$ #perturb input feature $x_{ij}$#
14: $\overline{b} \leftarrow b + \frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$ #perturb bias b#
15: Construct hidden layers $\{\mathbf{h}_1, \dots, \mathbf{h}_k\}$ and normalization layers $\{\overline{\mathfrak{h}}_1, \dots, \overline{\mathfrak{h}}_{(k)}\}$
16: Inject Laplace noise into coefficients of the approximated loss function $\widehat{\mathcal{F}}$
17: $\Delta_{\mathcal{F}} = M(|\overline{\mathfrak{h}}_{(k)}| + \frac{1}{4}|\overline{\mathfrak{h}}_{(k)}|^2)$ #Lemma 3.5#
18: for $\mathbf{x}_i \in D, R \in [0,2], l \in [1,M]$ do
19:     $\overline{\phi}^{(R)}_{l\mathbf{x}_i} \leftarrow \phi^{(R)}_{l\mathbf{x}_i} + \frac{1}{|L|}Lap(\frac{\Delta_{\mathcal{F}}}{\epsilon_3})$ #perturb coefficients of $\widehat{\mathcal{F}}$
20: Initialize $\theta_0$ randomly
21: for $t \in [T]$ do
22:     Take a random training batch $L$
23:     Construct differentially private affine transformation layer
24:     $\overline{\mathbf{h}}_{0L}(W_0) \leftarrow \{\overline{h}_L(W)\}_{h \in \mathbf{h}_0}$
25:     s.t. $\overline{h}_L(W) = \sum_{\mathbf{x}_i \in L}\big(\overline{\mathbf{x}}_i W^T + \overline{b})]\big)$
26:     Construct differentially private loss function
27:     $\overline{\mathcal{F}}_L(\theta_t) = \sum_{l=1}^{M}\sum_{\mathbf{x}_i \in L}\sum_{R=0}^{2}\big(\overline{\phi}^{(R)}_{l\mathbf{x}_i} W^T_{l(k)}\big)^R$
28:     Compute gradient descents
29:     $\theta_{t+1} \leftarrow \theta_t - \eta_t \frac{1}{|L|}\nabla_{\theta_t}\overline{\mathcal{F}}_L(\theta_t)$ #$\eta_t$ is a learning rate#
30: Return $\theta_T$ #$(\epsilon_1 + \epsilon_2 + \epsilon_3)$-differentially private#
```

The computation of $\overline{\mathbf{R}}(D)$ is ϵ_1-differential private. The correctness is based on the following lemmas.

Lemma 3.1. *Let D and D' be any two neighboring databases. Given $\mathbf{R}(D)$ and $\mathbf{R}(D')$ be the relevance of all input features on D and D', respectively, and denote their representations as*

$$\mathbf{R}(D) = \big\{R_j(D)\big\}_{j\in[1,d]} \text{ s.t. } R_j(D) = \frac{1}{|D|}\sum_{\mathbf{x}_i\in D} R_{x_{ij}}(\mathbf{x}_i)$$
$$\mathbf{R}(D') = \big\{R_j(D')\big\}_{j\in[1,d]} \text{ s.t. } R_j(D') = \frac{1}{|D'|}\sum_{\mathbf{x}'_i\in D'} R_{x'_{ij}}(\mathbf{x}'_i)$$

Then, we have the following inequality:

$$\frac{1}{|D|}\sum_{j=1}^{d}\Big\|\sum_{\mathbf{x}_i\in D} R_{x_{ij}}(\mathbf{x}_i) - \sum_{\mathbf{x}'_i\in D'} R_{x'_{ij}}(\mathbf{x}'_i)\Big\|_1 \leq \frac{2d}{|D|} \tag{3.13}$$

where d is the number of features in each tuple $\mathbf{x}_i \in D$.

Proof of Lemma 3.1

Proof. Assume that D and D' differ in the last tuple. Let $\mathbf{x}_n$ ($\mathbf{x}'_n$) be the last tuple in D (D'). We have that

$$\begin{aligned}\Delta_{\mathbf{R}} &= \frac{1}{|D|}\sum_{j=1}^{d}\Big\|\sum_{\mathbf{x}_i\in D} R_{x_{ij}}(\mathbf{x}_i) - \sum_{\mathbf{x}'_i\in D'} R_{x'_{ij}}(\mathbf{x}'_i)\Big\|_1 \\ &= \frac{1}{|D|}\sum_{j=1}^{d}\Big\|R_{x_{nj}}(\mathbf{x}_n) - R_{x'_{nj}}(\mathbf{x}'_n)\Big\|_1 \\ &\leq \frac{2}{|D|}\max_{\mathbf{x}_i\in D}\sum_{j=1}^{d}\|R_{x_{ij}}(\mathbf{x}_i)\|_1 \leq \frac{2d}{|D|}\end{aligned}$$

Equation (3.13) holds. $\square$

Lemma 3.2. *Algorithm 3.1 preserves ϵ_1-differential privacy in the computation of* $\overline{\mathbf{R}}(D)$.

Proof of Lemma 3.2:

Proof. Let D and D' be two neighbor databases. Without loss of generality, assume that D and D' differ in the last tuple $\mathbf{x}_n$ ($\mathbf{x}'_n$). $\overline{R}_j(D)$ is calculated as done in line 6, Algorithm 3.1, and $\overline{\mathcal{R}}(D) = \{\overline{R}_j(D)\}_{j\in[1,d]}$ is the output of the Algorithm 3.1 (line 7). The perturbation of the relevance $R_j(D)$ can be rewritten as:

$$\overline{R}_j = \frac{1}{|D|}\sum_{\mathbf{x}_i\in D} R_{x_{ij}}(\mathbf{x}_i) + Lap(\frac{\Delta_{\mathbf{R}}}{\epsilon_1}) \tag{3.14}$$

Since all the input features are perturbed, we have that

$$\begin{aligned}
&\frac{Pr\big(\overline{\mathbf{R}}(D)\big)}{Pr\big(\overline{\mathbf{R}}(D')\big)} = \frac{\prod_{j=1}^{d}\exp\big(\frac{\epsilon_1\|\frac{1}{|D|}\sum_{\mathbf{x}_i\in D}R_j(\mathbf{x}_i)-\overline{R}_j\|_1}{\Delta_{\mathbf{R}}}\big)}{\prod_{j=1}^{d}\exp\big(\frac{\epsilon_1\|\frac{1}{|D|}\sum_{\mathbf{x}'_i\in D}R_j(\mathbf{x}'_i)-\overline{R}_j\|_1}{\Delta_{\mathbf{R}}}\big)}\\
&\leq \prod_{j=1}^{d}\exp(\frac{\epsilon_1}{|D|\Delta_{\mathbf{R}}}\Big\|\sum_{\mathbf{x}_i\in D}R_j(\mathbf{x}_i) - \sum_{\mathbf{x}'_i\in D'}R_j(\mathbf{x}'_i)\Big\|_1)\\
&\leq \prod_{j=1}^{d}\exp(\frac{\epsilon_1}{|D|\Delta_{\mathbf{R}}}\Big\|R_j(\mathbf{x}_n) - R_j(\mathbf{x}'_n)\Big\|_1)\\
&\leq \prod_{j=1}^{d}\exp(\frac{\epsilon_1}{|D|\Delta_{\mathbf{R}}}2\max_{\mathbf{x}_n\in D}\big\|R_j(\mathbf{x}_n)\big\|_1)\\
&\leq \exp(\epsilon_1\frac{2\max_{\mathbf{x}_n\in L}\sum_{j=1}^{d}\|R_j(\mathbf{x}_n)\|_1}{|D|\Delta_{\mathbf{R}}})\\
&\leq \exp(\epsilon_1)
\end{aligned}$$

Consequently, the computation of $\overline{\mathbf{R}}(D)$ preserves ϵ_1-differential privacy in Algorithm 3.1. $\square$

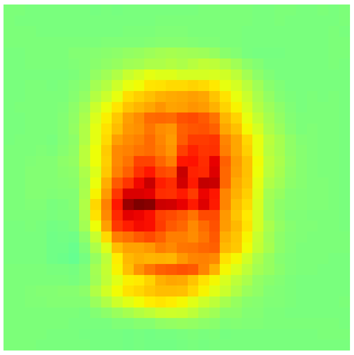


Figure 3.3 The average differentially private relevance of each input feature given MNIST dataset.

Lemma 3.2 shows that the computation of the relevances $\overline{\mathbf{R}}(D)$ is differentially private. Figure 3.3 illustrates the differentially private relevance $\overline{R}_j$ (Equation (3.12)) of each j-th coefficient given the database D. It is worth noting that the relevance distribution is not identical. In the next subsection, $\overline{\mathbf{R}}(D)$ is used to redistribute the noise injected into the affine transformation layer $\mathbf{h}_0$ in our deep neural network.

3.1.7 Private Affine Transformation Layer with Adaptive Noise

In general, before applying activation functions such as ReLU and sigmoid, the affine transformation of a hidden neuron $h \in \mathbf{h}_0$ can be presented as:

$$h_{\mathbf{x}_i}(W) = b + \mathbf{x}_i W^T \tag{3.15}$$

where b is a static bias, and W is the parameter of h. Given a training batch L, h can be rewritten as:

$$h_L(W) = \sum_{\mathbf{x}_i \in L} (b + \mathbf{x}_i W^T) \tag{3.16}$$

Given the above representation of each neuron $h_L(W_h)$, we preserve differential privacy in the computation of $\mathbf{h}_0$ on L by injecting Laplace noise into inputs b and $\mathbf{x}_i$ of every neuron $h_L(W) \in \mathbf{h}_0$. Intuitively, we can apply an identical noise distribution $\frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$ to all input features, where $\Delta_{\mathbf{h}_0} = 2\sum_{h\in\mathbf{h}_0} d$ (line 9, Algorithm 3.1). This approach works well when every input feature has an identical contribution to the model outcome. This approach is described in the following subsection **Identical Laplace Mechanism (ILM)**.

Identical Laplace Mechanism (ILM) We can add an identical noise distribution of $\frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$ to all input features, where $\Delta_{\mathbf{h}_0} = 2\sum_{h\in\mathbf{h}_0} d$ (line 9, Algorithm 3.1) to preserve differential privacy in the computation of $\mathbf{h}_0$. In fact, we have that $\forall \mathbf{x}_i \in D, j \in [1,d]$: $\overline{x}_{ij} = x_{ij} + \frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$. For each $h \in \overline{h}_{0L}$, h can be re-written as:

$$\overline{h}_L(W) = \sum_{j=1}^{d}\Big[\sum_{\mathbf{x}_i\in L}\big(x_{ij} + \frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})\big)W^T\Big] + \sum_{\mathbf{x}_i\in L}\big(b + \frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})\big) \qquad (3.17)$$

Let us consider the static bias $b = 1$ as the 0-th input feature and its associated parameter W_b, i.e., $x_{i0} = b = 1$ and $W = W_b \cup W$, we have that

$$\begin{aligned}\overline{h}_L(W) &= \sum_{j=0}^{d}\Big[\sum_{\mathbf{x}_i\in L}\big(x_{ij} + \frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})\big)W^T\Big] \\ &= \sum_{j=0}^{d}\Big[\sum_{\mathbf{x}_i\in L} x_{ij} + Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})\Big]W^T = \sum_{j=0}^{d}\overline{\phi}_j^h W^T\end{aligned}$$

where $\overline{\phi}_j^h = \big[\sum_{\mathbf{x}_i\in L} x_{ij} + Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})\big]$.

We can see that $\overline{\phi}_j^h$ is the perturbation of the input feature x_{ij} associated with the j-th parameter $W_j \in W$ of the hidden neuron h on L. Since all the hidden neurons h in $\mathbf{h}_0$ are perturbed, we have that:

$$Pr\big(\overline{\mathbf{h}}_{0L}(W_0)\big) = \prod_{h\in\mathbf{h}_0}\prod_{j=0}^{d} exp\Big(\frac{\epsilon_2\|\sum_{\mathbf{x}_i\in L} x_{ij} - \overline{\phi}_j^h\|}{\Delta_{\mathbf{h}_0}}\Big)$$

$\Delta_{\mathbf{h}_0}$ is set to $2\sum_{h\in\mathbf{h}_0} d$ and $\overline{\mathbf{h}}_{0L}(W_0) = \{\overline{h}_L(W)\}_{h\in\mathbf{h}_0}$ is the output, we have that

$$\begin{aligned}
&\frac{Pr\big(\overline{\mathbf{h}}_{0L}(W_0)\big)}{Pr\big(\overline{\mathbf{h}}_{0L'}(W_0)\big)} = \frac{\prod_{h\in\mathbf{h}_0}\prod_{j=0}^{d} \exp\big(\frac{\epsilon_2\|\sum_{\mathbf{x}_i\in L} x_{ij} - \overline{\phi}_j^h\|_1}{\Delta_{\mathbf{h}_0}}\big)}{\prod_{h\in\mathbf{h}_0}\prod_{j=0}^{d} \exp\big(\frac{\epsilon_2\|\sum_{\mathbf{x}'_i\in L'} x'_{ij} - \overline{\phi}_j^h\|_1}{\Delta_{\mathbf{h}_0}}\big)} \\
&\leq \prod_{h\in\mathbf{h}_0}\prod_{j=0}^{d} \exp\Big(\frac{\epsilon_2}{\Delta_{\mathbf{h}_0}}\Big\|\sum_{\mathbf{x}_i\in L} x_{ij} - \sum_{\mathbf{x}'_i\in L'} x'_{ij}\Big\|_1\Big) \\
&\leq \prod_{h\in\mathbf{h}_0}\prod_{j=1}^{d} \exp\Big(\frac{\epsilon_2}{\Delta_{\mathbf{h}_0}}\Big\|x_{nj} - x'_{nj}\Big\|_1\Big) \\
&\leq \prod_{h\in\mathbf{h}_0}\prod_{j=1}^{d} \exp\Big(\frac{\epsilon_2}{\Delta_{\mathbf{h}_0}} 2\max_{\mathbf{x}_n\in L}\|x_{nj}\|_1\Big) \leq \prod_{h\in\mathbf{h}_0}\prod_{j=1}^{d} \exp\Big(\frac{2\epsilon_2}{\Delta_{\mathbf{h}_0}}\Big) \\
&\leq \exp\Big(\epsilon_2\frac{2\sum_{h\in\mathbf{h}_0} d}{\Delta_{\mathbf{h}_0}}\Big) = \exp(\epsilon_2)
\end{aligned}$$

Consequently, based on the above analysis, the computation of $\overline{\mathbf{h}}_{0L}(W_0)$ preserves ϵ_2-differential privacy in Algorithm 3.1 by injecting an identical Laplace noise $\frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$ into all input features. In addition, given the identical Laplace noise, we do not need to use the differentially private relevance $\overline{\mathbf{R}}(D)$, since we do not need to redistribute the noise in the first affine transformation layer $\mathbf{h}_0$.

Problems with the ILM In practice, this assumption usually is violated. For instance, Figure 3.2 illustrates the relevance, estimated by the LRP algorithm [98], of each input feature given different handwritten digits. It is clear that the relevances are not identical. The differentially private relevances are not identical as well (Figure 3.3). Therefore, injecting the same magnitude of noise into all input features may affect the utility of differentially private neural networks.

To address this problem, we propose an Adaptive Laplace Mechanism (**AdLM**), to adaptively redistribute the injected noise to improve the performance. Given hidden units $h_{\mathbf{x}_i}(W)$ in Equation (3.15), our key idea is to intentionally add *more noise* into input features which are *less relevant* to the model output Y, and vice-versa. As a result, we expect to improve the utility of the model under differential privacy. In fact, we introduce a privacy budget ratio β_j and the privacy budget ϵ_j for each j-th input feature as follows:

$$\beta_j = \frac{d \times |\overline{R}_j|}{\sum_{j=1}^{d} |\overline{R}_j|} \quad s.t. \quad \epsilon_j = \beta_j \times \epsilon_2 \tag{3.18}$$

We set $\Delta_{\mathbf{h}_0} = 2\sum_{h \in \mathbf{h}_0} d$ based on the maximum values of all the input features x_{ij} (line 9, Algorithm 3.1). In line 11, β_j can be considered as the fraction of the contribution to $\Delta_{\mathbf{h}_0}$ from the j-th input feature to the hidden neuron $h \in \mathbf{h}_0$. In lines 12-13, each input feature x_{ij} of every hidden neuron h in the first affine transformation layer $\mathbf{h}_0$ is perturbed by adding adaptive Laplace noise $\frac{1}{|L|}Lap(\Delta_{\mathbf{h}_0}/\epsilon_j)$. The perturbed input features are denoted as $\overline{\mathbf{x}}_i$. In lines 20-21, given a random training batch L, we construct the differentially private affine transformation layer $\overline{\mathbf{h}}_{0L}$, which consists of perturbed hidden neurons $\overline{h}_L(W)$:

$$\overline{\mathbf{h}}_{0L}(W_0) = \left\{\overline{h}_L(W)\right\}_{h \in \mathbf{h}_0} s.t.\ \overline{h}_L(W) = \sum_{\mathbf{x}_i \in L} \left(\overline{\mathbf{x}}_i W^T + \overline{b}\right)$$

where $\overline{b} = b + \frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$ is the perturbed bias (line 14). The following lemma shows that Algorithm 3.1 preserves ϵ_2-differential privacy in the computation of $\overline{\mathbf{h}}_{0L}$.

Lemma 3.3. *Let L and L' be any two neighboring batches. Given parameter W_0, let $\mathbf{h}_{0L}$ and $\mathbf{h}_{0L'}$ be the first affine transformation layers on L and L', respectively, and denote their representations as follows:*

$$\mathbf{h}_{0L}(W_0) = \{h_L(W)\}_{h\in\mathbf{h}_0} \text{ s.t. } h_L(W) = \sum_{\mathbf{x}_i\in L}(b + \mathbf{x}_i W^T)$$
$$\mathbf{h}_{0L'}(W_0) = \{h_{L'}(W)\}_{h\in\mathbf{h}_0} \text{ s.t. } h_{L'}(W) = \sum_{\mathbf{x}'_i\in L'}(b + \mathbf{x}'_i W^T)$$

Then, we have the following inequality:

$$\Delta_{\mathbf{h}_0} = \sum_{h\in\mathbf{h}_0}\sum_{j=1}^{d}\Big\|\sum_{\mathbf{x}_i\in L}x_{ij} - \sum_{\mathbf{x}'_i\in L'}x'_{ij}\Big\|_1 \leq 2\sum_{h\in\mathbf{h}_0} d \tag{3.19}$$

where d is the number of features in each tuple $\mathbf{x}_i \in D$.

Proof of Lemma 3.3:

Proof. Assume that L and L' differ in the last tuple. Let $\mathbf{x}_n$ ($\mathbf{x}'_n$) be the last tuple in L (L'). We have that

$$\Delta_{\mathbf{h}_0} = \sum_{h\in\mathbf{h}_0}\sum_{j=1}^{d}\Big\|\sum_{\mathbf{x}_i\in L}x_{ij} - \sum_{\mathbf{x}'_i\in L'}x'_{ij}\Big\|_1$$
$$= \sum_{h\in\mathbf{h}_0}\sum_{j=1}^{d}\|x_{nj} - x'_{nj}\|_1 \leq 2\max_{\mathbf{x}_i\in L}\sum_{h\in\mathbf{h}_0}\sum_{j=1}^{d}\|x_{ij}\|_1 \tag{3.20}$$

Since $\forall \mathbf{x}_i, j : x_{ij} \in [0,1]$, from Equation (3.20) we have that: $\Delta_{\mathbf{h}_0} \leq 2\sum_{h\in\mathbf{h}_0} d$. Equation (3.19) holds. $\square$

Lemma 3.4. *Algorithm 3.1 preserves ϵ_2-differential privacy in the computation of $\overline{\mathbf{h}}_{0L}(W_0)$ (lines 24-25).*

Proof of Lemma 3.4:

Proof. From lines 24-25 in the Algorithm 3.1, for each $h \in \overline{h}_{0L}$, h can be re-written as:

$$\overline{h}_L(W) = \sum_{j=1}^{d} \Big[\sum_{\mathbf{x}_i \in L} \big(x_{ij} + \frac{1}{|L|} Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_j}) \big) W^T \Big] + \sum_{\mathbf{x}_i \in L} \big(b + \frac{1}{|L|} Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2}) \big) \tag{3.21}$$

Let us consider the static bias $b = 1$ as the 0-th input feature and its associated parameter W_b, i.e., $x_{i0} = b = 1$ and $W = W_b \cup W$, we have that

$$\overline{h}_L(W) = \sum_{j=0}^{d} \Big[\sum_{\mathbf{x}_i \in L} \big(x_{ij} + \frac{1}{|L|} Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_j}) \big) W^T \Big] \tag{3.22}$$

$$= \sum_{j=0}^{d} \Big[\sum_{\mathbf{x}_i \in L} x_{ij} + Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_j}) \Big] W^T = \sum_{j=0}^{d} \overline{\phi}_j^h W^T \tag{3.23}$$

where $\overline{\phi}_j^h = \big[\sum_{\mathbf{x}_i \in L} x_{ij} + Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_j}) \big]$.

We can see that $\overline{\phi}_j^h$ is the perturbation of the input feature x_{ij} associated with the j-th parameter $W_j \in W$ of the hidden neuron h on L. Since all the hidden neurons h in $\mathbf{h}_0$ are perturbed, we have that:

$$Pr\big(\overline{\mathbf{h}}_{0L}(W_0)\big) = \prod_{h\in\mathbf{h}_0} \prod_{j=0}^{d} exp\big(\frac{\epsilon_j \| \sum_{\mathbf{x}_i \in L} x_{ij} - \overline{\phi}_j^h \|}{\Delta_{\mathbf{h}_0}}\big)$$

$\Delta_{\mathbf{h}_0}$ is set to $2\sum_{h\in\mathbf{h}_0} d$ (line 9 in Algorithm 3.1). $\overline{\mathbf{h}}_{0L}(W_0)$ is the output (lines 24-25 in Algorithm 3.1). We have that

$$\frac{Pr\big(\overline{\mathbf{h}}_{0L}(W_0)\big)}{Pr\big(\overline{\mathbf{h}}_{0L'}(W_0)\big)} = \frac{\prod_{h\in\mathbf{h}_0}\prod_{j=0}^{d}\exp\big(\frac{\epsilon_j\|\sum_{\mathbf{x}_i\in L} x_{ij}-\overline{\phi}_j^h\|_1}{\Delta_{\mathbf{h}_0}}\big)}{\prod_{h\in\mathbf{h}_0}\prod_{j=0}^{d}\exp\big(\frac{\epsilon_j\|\sum_{\mathbf{x}'_i\in L'} x'_{ij}-\overline{\phi}_j^h\|_1}{\Delta_{\mathbf{h}_0}}\big)}$$

$$\leq \prod_{h\in\mathbf{h}_0}\prod_{j=0}^{d}\exp\Big(\frac{\epsilon_j}{\Delta_{\mathbf{h}_0}}\Big\|\sum_{\mathbf{x}_i\in L} x_{ij} - \sum_{\mathbf{x}'_i\in L'} x'_{ij}\Big\|_1\Big)$$

$$\leq \prod_{h\in\mathbf{h}_0}\prod_{j=1}^{d}\exp\Big(\frac{\epsilon_j}{\Delta_{\mathbf{h}_0}}2\max_{\mathbf{x}_n\in L}\big\|x_{nj}\big\|_1\Big) \leq \prod_{h\in\mathbf{h}_0}\prod_{j=1}^{d}\exp\Big(\frac{2\epsilon_j}{\Delta_{\mathbf{h}_0}}\Big)$$

$$\leq \prod_{h\in\mathbf{h}_0}\prod_{j=1}^{d}\exp\Big(\epsilon_2\frac{2\frac{d\times|\overline{R}_j|}{\sum_{j=1}^{d}|\overline{R}_j|}}{\Delta_{\mathbf{h}_0}}\Big)$$

$$\leq \exp\Big(\epsilon_2\frac{2\sum_{h\in\mathbf{h}_0} d\big[\sum_{j=1}^{d}\frac{|\overline{R}_j|}{\sum_{j=1}^{d}|\overline{R}_j|}\big]}{\Delta_{\mathbf{h}_0}}\Big) = \exp(\epsilon_2)$$

Consequently, the computation of $\overline{\mathbf{h}}_{0L}(W_0)$ preserves ϵ_2-differential privacy in Algorithm 3.1. $\square$

Lemma 3.4 shows that we can redistribute the noise in the computation of the first hidden layer $\overline{\mathbf{h}}_{0L}$ under differential privacy. In addition, given a batch L, without accessing additional information from the original data, none of the computations on top of $\overline{\mathbf{h}}_{0L}$ risk the privacy protection under differential privacy. These computation tasks include the application of activation functions, e.g., ReLU and sigmoid, on $\overline{\mathbf{h}}_{0L}$, the computation of hidden layers $\mathbf{h}_1, \ldots, \mathbf{h}_k$, local response normalizations, drop-out operations, polling layers, etc. (line 15, Algorithm 3.1). This result can be applied to both fully-connected layers and convolution layers. In this work, we applied ReLU on top of $\overline{\mathbf{h}}_{0L}$ and other layers $\mathbf{h}_1, \ldots, \mathbf{h}_k$. Local response normalization layers are used after the application of ReLUs in each hidden layer to bound ReLU functions.

Local Response Normalization The hidden units of the lower layer will be considered as the input of the next layer (Figure 3.1). To ensure that this input is bounded $\overline{h}_{\mathbf{x}_i} \in [0,1]$, as in [159, 60], we add a local response normalization (LRN) layer on top of each hidden layer. Given a fully-connected layer, as in [60], given an input $\mathbf{x}_i$, each perturbed neuron $\overline{h}_{\mathbf{x}_i}(W)$ can be directly normalized as follows: $\overline{\mathfrak{h}}_{\mathbf{x}_i} \leftarrow \Big(\overline{h}_{\mathbf{x}_i}(W) - \chi\Big)/(\varphi - \chi)$, where φ and χ denote the maximum and minimum values in the domain of $\{\overline{h}_{\mathbf{x}_i}\}_{i \in L}$.

Given a convolution layer with a perturbed neuron $\overline{h}^k_{ij}$ at location (i,j) in the k-th feature map, based on [159], our local response normalization (LRN) is presented as follows:

$$\overline{\mathfrak{h}}^k_{ij} \leftarrow \overline{h}^k_{ij}/\max\Big(\overline{h}^k_{ij}, \big(q + \alpha \sum_{m=\max(0,k-l/2)}^{\min(N-1,k+l/2)} (\overline{h}^m_{ij})^2\big)^{\beta}\Big) \tag{3.24}$$

where the constants $q, l, \alpha,$ and β are hyper-parameters, N is the total number of feature maps. As in [159], we used $q = 2, l = 5, \alpha = 10^{-4}$, and $\beta = 0.75$ in our experiments.

3.1.8 Perturbation of the Loss Function $\mathcal{F}_L(\theta)$

On top of our private deep neural network (Figure 3.1), we add an output layer with the loss function $\mathcal{F}_L(\theta)$ to predict Y. Since the loss function $\mathcal{F}_L(\theta)$ accesses the labels y_i given $\mathbf{x}_i \in L$ from the data, we need to protect the labels y_i at the output layer. First, we derive a polynomial approximation of the loss function based on Taylor Expansion [160]. Then, we inject Laplace noise into coefficients of the loss function $\mathcal{F}$ to preserve differential privacy on each training batch L.

The model output variables $\{\hat{y}_1, \ldots, \hat{y}_M\}$ are fully linked to the normalized highest hidden layer, denoted $\overline{\mathfrak{h}}_{(k)}$, by weighted connections $W_{(k)}$ (Figure 3.1). As common, the logistic function can be used as an activation function of the output variables. Given, l-th output variable $\hat{y}_l$ and $\mathbf{x}_i$, we have:

$$\hat{y}_{il} = \sigma\big(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)} W^T_{l(k)}\big) \tag{3.25}$$

where $\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}$ is the state of $\overline{\mathfrak{h}}_{(k)}$ derived from $\overline{\mathbf{h}}_{0\mathbf{x}_i}$ by navigating through the neural network.

Cross-entropy error [39] can be used as a loss function. It has been widely used and applied in real-world applications [39]. Therefore, it is critical to preserve differential privacy under the use of the cross-entropy error function. Other loss functions, e.g., square errors, can be applied in the output layer, as well. In our context, the cross-entropy error function is given by:

$$\begin{aligned}\mathcal{F}_L(\theta) &= -\sum_{l=1}^{M}\sum_{\mathbf{x}_i\in L}\Big(y_{il}\log\hat{y}_{il} + (1-y_{il})\log(1-\hat{y}_{il})\Big)\\ &= -\sum_{l=1}^{M}\sum_{\mathbf{x}_i\in L}\Big(y_{il}\log(1+e^{-\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}W^T_{l(k)}}) + (1-y_{il})\log(1+e^{\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}W^T_{l(k)}})\Big)\end{aligned} \tag{3.26}$$

Based on [60] and Taylor Expansion [160], we derive the polynomial approximation of $\mathcal{F}_L(\theta)$ as:

$$\widehat{\mathcal{F}}_L(\theta) = \sum_{l=1}^{M}\sum_{\mathbf{x}_i\in L}\sum_{q=1}^{2}\sum_{R=0}^{2}\frac{f_{ql}^{(R)}(0)}{R!}\big(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}W_{l(k)}^T\big)^R$$
$$= \sum_{l=1}^{M}\sum_{\mathbf{x}_i\in L}\Big[\sum_{q=1}^{2} f_{ql}^{(0)}(0) + \big(\sum_{q=1}^{2} f_{ql}^{(1)}(0)\big)\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}W_{l(k)}^T + \big(\sum_{q=1}^{2}\frac{f_{ql}^{(2)}(0)}{2!}\big)\big(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}W_{l(k)}^T\big)^2\Big] \quad (3.27)$$

where $\forall l \in [1, M] : f_{1l}(z) = y_{il}\log(1 + e^{-z})$ and $f_{2l}(z) = (1 - y_{il})\log(1 + e^{z})$.

To achieve ϵ_3-differential privacy, we employ *functional mechanism* [161] to perturb the loss function $\widehat{\mathcal{F}}_L(\theta)$ by injecting Laplace noise into its polynomial coefficients. So, we only need to perturb $\widehat{\mathcal{F}}_L(\theta)$ just once in each training batch. To be clear, we denote $\{\phi_{l\mathbf{x}_i}^{(0)}, \phi_{l\mathbf{x}_i}^{(1)}, \phi_{l\mathbf{x}_i}^{(2)}\}$ as the coefficients, where $\phi_{l\mathbf{x}_i}^{(0)} = \sum_{q=1}^{2} f_{ql}^{(0)}(0)$ and $\phi_{l\mathbf{x}_i}^{(1)}$ and $\phi_{l\mathbf{x}_i}^{(2)}$ are coefficients at the first order and the second order of the function $\widehat{\mathcal{F}}_L(\theta)$. In fact, $\phi_{l\mathbf{x}_i}^{(1)}$ and $\phi_{l\mathbf{x}_i}^{(2)}$ will be combinations between the approximation terms $\sum_{q=1}^{2} f_{ql}^{(1)}(0)$, $\sum_{q=1}^{2}\frac{f_{ql}^{(2)}(0)}{2!}$, and $\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}$.

In Algorithm 3.1, we set $\Delta_{\mathcal{F}} = M(|\overline{\mathfrak{h}}_{(k)}| + \frac{1}{4}|\overline{\mathfrak{h}}_{(k)}|^2)$ (*line 17*). In essence, coefficients $\phi_{l\mathbf{x}_i}^{(R)}$ with $R \in [0, 2]$ are functions of the label y_{il} only. Therefore, we can perform the perturbation by injecting Laplace noise $1/|L|Lap(\frac{\Delta_{\mathcal{F}}}{\epsilon_3})$ into $\phi_{l\mathbf{x}_i}^{(R)}$ for every training label $y_{il} \in D$ (*lines 18-19*). Then, the perturbed coefficients, denoted $\overline{\phi}_{l\mathbf{x}_i}^{(R)}$ are used to construct the differentially private loss function $\overline{\mathcal{F}}_L(\theta_t)$ (*line 27*) during the training process without accessing the original label y_{il} again (*lines 20-30*). Stochastic gradient descent and back-propagation algorithms are used to minimize the perturbed loss function $\overline{\mathcal{F}}_L(\theta_t)$.

Now, we are ready to state that the computation of $\overline{\mathcal{F}}_L(\theta_t)$ is ϵ_3-differentially private, and our mechanism preserves $(\epsilon_1 + \epsilon_2 + \epsilon_3)$-differential privacy in the following lemmas.

Lemma 3.5. *Let L and L' be any two neighboring batches. Let $\widehat{\mathcal{F}}_L(\theta)$ and $\widehat{\mathcal{F}}_{L'}(\theta)$ be the loss functions on L and L', respectively, then we have the following inequality:*

$$\Delta_{\mathcal{F}} = \sum_{l=1}^{M} \sum_{R=0}^{2} \Big\| \sum_{\mathbf{x}_i \in L} \phi_{l\mathbf{x}_i}^{(R)} - \sum_{\mathbf{x}'_i \in L'} \phi_{l\mathbf{x}'_i}^{(R)} \Big\| \leq M(|\overline{\mathfrak{h}}_{(k)}| + \frac{1}{4}|\overline{\mathfrak{h}}_{(k)}|^2)$$

where $|\overline{\mathfrak{h}}_{(k)}|$ is the number of hidden neurons in $\overline{\mathfrak{h}}_{(k)}$.

Proof of Lemma 3.5

Proof. Assume that L and L' differ in the last tuple. Let $\mathbf{x}_n$ ($\mathbf{x}'_n$) be the last tuple in L (L'). We have that

$$\begin{aligned} \Delta_{\mathcal{F}} &= \sum_{l=1}^{M} \sum_{R=0}^{2} \Big\| \sum_{\mathbf{x}_i \in L} \phi_{l\mathbf{x}_i}^{(R)} - \sum_{\mathbf{x}'_i \in L'} \phi_{l\mathbf{x}'_i}^{(R)} \Big\| \\ &= \sum_{l=1}^{M} \sum_{R=0}^{2} \|\phi_{l\mathbf{x}_n}^{(R)} - \phi_{l\mathbf{x}'_n}^{(R)}\| \end{aligned} \tag{3.28}$$

We can show that $\phi_{l\mathbf{x}_n}^{(0)} = \sum_{q=1}^{2} f_{ql}^{(0)}(0) = y_{nl}\log 2 + (1-y_{nl})\log 2 = \log 2$. Similarly, we can show that $\phi_{l\mathbf{x}'_n}^{(0)} = \log 2$. As a result, $\phi_{l\mathbf{x}_n}^{(0)} = \phi_{l\mathbf{x}'_n}^{(0)}$. Therefore

$$
\begin{aligned}
\Delta_{\mathcal{F}} &= \sum_{l=1}^{M}\sum_{R=0}^{2}\|\phi^{(R)}_{l\mathbf{x}_n} - \phi^{(R)}_{l\mathbf{x}'_n}\| = \sum_{l=1}^{M}\sum_{R=1}^{2}\|\phi^{(R)}_{l\mathbf{x}_n} - \phi^{(R)}_{l\mathbf{x}'_n}\| \\
&\leq \sum_{l=1}^{M}\sum_{R=1}^{2}\left(\|\phi^{(R)}_{l\mathbf{x}_n}\| + \|\phi^{(R)}_{l\mathbf{x}'_n}\|\right) \leq 2\max_{\mathbf{x}_n}\sum_{l=1}^{M}\sum_{R=1}^{2}\|\phi^{(R)}_{l\mathbf{x}_n}\| \\
&\leq 2\max_{\mathbf{x}_n}\Big[\sum_{l=1}^{M}(\frac{1}{2} - y_{nl})\sum_{e=1}^{|\overline{\mathfrak{h}}_{(k)}|}\overline{\mathfrak{h}}_{e\mathbf{x}_n(k)} + \sum_{l=1}^{M}\big(\frac{1}{8}\sum_{e,g}\overline{\mathfrak{h}}_{e\mathbf{x}_n(k)}\overline{\mathfrak{h}}_{g\mathbf{x}_n(k)}\big)\Big] \\
&\leq 2(\frac{1}{2}M \times |\overline{\mathfrak{h}}_{(k)}| + \frac{1}{8}M \times |\overline{\mathfrak{h}}_{(k)}|^2) \\
&= M(|\overline{\mathfrak{h}}_{(k)}| + \frac{1}{4}|\overline{\mathfrak{h}}_{(k)}|^2)
\end{aligned}
$$

where $\overline{\mathfrak{h}}_{e\mathbf{x}_n(k)}$ is the state of e-th hidden neuron in $\overline{\mathfrak{h}}_{(k)}$. □

Lemma 3.6. *Algorithm 3.1 preserves ϵ_3-differential privacy in the computation of $\overline{\mathcal{F}}_L(\theta_t)$ (line 27).*

Proof of Lemma 3.6

Proof. Let L and L' be two neighbor batches. Without loss of generality, assume that L and L' differ in the last tuple $\mathbf{x}_n$ ($\mathbf{x}'_n$). $\Delta_{\mathcal{F}}$ is calculated as done in line 17, Algorithm 3.1, and $\overline{\mathcal{F}}_L(\theta_t) = \sum_{l=1}^{M}\sum_{\mathbf{x}_i \in L}\sum_{R=0}^{2}\overline{\phi}^{(R)}_{l\mathbf{x}_i}\big(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}W^T_{l(k)}\big)^R$ is the output of line 27 of the Algorithm 3.1. Note that $\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}$ is the state of $\overline{\mathfrak{h}}_{(k)}$ derived from $\overline{\mathbf{h}}_{0\mathbf{x}_i}$ by navigating through the neural network. The perturbation of the coefficient $\phi^{(R)}_l$, denoted as $\overline{\phi}^{(R)}_l$, can be rewritten as:

$$
\overline{\phi}^{(R)}_l = \Big[\sum_{\mathbf{x}_i \in L}\phi^{(R)}_{l\mathbf{x}_i} + Lap(\frac{\Delta_{\mathcal{F}}}{\epsilon_3})\Big] \tag{3.29}
$$

We can see that $\overline{\phi}^{(R)}_l$ is the perturbation of the coefficient $\phi^{(R)}_l$ associated with the labels y_{il} in the training batch L. We have that

$$\frac{Pr\big(\overline{\mathcal{F}}_L(\theta_t)\big)}{Pr\big(\overline{\mathcal{F}}_{L'}(\theta_t)\big)} = \frac{\prod_{l=1}^{M}\prod_{R=0}^{2}\exp\big(\frac{\epsilon_3\|\sum_{\mathbf{x}_i\in L}\phi_{l\mathbf{x}_i}^{(R)} - \overline{\phi}_l^{(R)}\|_1}{\Delta_{\mathcal{F}}}\big)}{\prod_{l=1}^{M}\prod_{R=0}^{2}\exp\big(\frac{\epsilon_3\|\sum_{\mathbf{x}'_i\in L'}\phi_{l\mathbf{x}_i}^{(R)} - \overline{\phi}_l^{(R)}\|_1}{\Delta_{\mathcal{F}}}\big)}$$

$$\leq \prod_{l=1}^{M}\prod_{R=0}^{2}\exp(\frac{\epsilon_3}{\Delta_{\mathcal{F}}}\Big\|\sum_{\mathbf{x}_i\in L}\phi_{l\mathbf{x}_i}^{(R)} - \sum_{\mathbf{x}'_i\in L'}\phi_{l\mathbf{x}'_i}^{(R)}\Big\|_1)$$

$$\leq \prod_{l=1}^{M}\prod_{R=0}^{2}\exp(\frac{\epsilon_3}{\Delta_{\mathcal{F}}}\Big\|\phi_{l\mathbf{x}_n}^{(R)} - \phi_{l\mathbf{x}'_n}^{(R)}\Big\|_1)$$

$$\leq \prod_{l=1}^{M}\prod_{R=0}^{2}\exp(\frac{\epsilon_3}{\Delta_{\mathcal{F}}}2\max_{\mathbf{x}_n\in L}\big\|\phi_{l\mathbf{x}_n}^{(R)}\big\|_1)$$

$$\leq \exp(\epsilon_3\frac{2\max_{\mathbf{x}_n\in L}\sum_{l=1}^{M}\sum_{R=0}^{2}\|\phi_{l\mathbf{x}_n}^{(R)}\|_1}{\Delta_{\mathcal{F}}})$$

$$\leq \exp(\epsilon_3\frac{M(|\overline{\mathfrak{h}}_{(k)}| + \frac{1}{4}|\overline{\mathfrak{h}}_{(k)}|^2)}{\Delta_{\mathcal{F}}}) = \exp(\epsilon_3)$$

Consequently, the computation of $\overline{\mathcal{F}}_L(\theta_t)$ preserves ϵ_3-differential privacy in Algorithm 3.1. □

3.1.9 The Correctness and Characteristics of the AdLM

The following theorem illustrates that the Algorithm 3.1 preserves ϵ-differential privacy, where $\epsilon = \epsilon_1 + \epsilon_2 + \epsilon_3$.

Theorem 3.1. *Algorithm 3.1 preserves ϵ-differential privacy, where $\epsilon = \epsilon_1 + \epsilon_2 + \epsilon_3$.*

Proof of Theorem 3.1

Proof. At a specific training step $t \in T$, it is crystal clear that the computation of $\overline{\mathbf{h}}_{0L}$ is ϵ_2-differentially private (Lemma 3.4). Therefore, the computation of the hidden layers $\mathbf{h}_1, \ldots, \mathbf{h}_k$ and normalization layers $\overline{\mathfrak{h}}$ are differentially private. This is because they do not access any additional information from the data. At the output layer, the loss function

$\overline{\mathcal{F}}_L(\theta_t)$ is ϵ_3-differentially private (Lemma 3.6). The computation of gradients $\frac{1}{|L|}\nabla_{\theta_t}\overline{\mathcal{F}}_L(\theta_t)$ and descents is an optimization process, without using any additional information from the original data. Thus, θ_{t+1} is differentially private (line 29, Algorithm 3.1).

This optimization process is repeated through T steps without querying the original data D (lines 21-30). This is done because Laplace noise is injected into input features x_{ij} and coefficients $\phi_{l\mathbf{x}_i}^{(R)}$ as preprocessing steps (lines 3-19). Note that $\overline{x}_{ij}$ and $\overline{\phi}_{l\mathbf{x}_i}^{(R)}$ are associated with features $\mathbf{x}_i$ and the label y_i, respectively. $\overline{\mathbf{h}}_{0L}$ and $\overline{\mathcal{F}}_L(\theta_t)$ are computed based on $\overline{x}_{ij}$ and $\overline{\phi}_{l\mathbf{x}_i}^{(R)}$. As a result, the noise and privacy budget consumption will not be accumulated during the training process.

Finally, $\overline{\mathcal{F}}_L(\theta_t)$ uses the outputs of $\overline{\mathbf{h}}_{0L}$, which essentially uses the differentially private relevances $\overline{\mathbf{R}}(D)$ as of one inputs. $\overline{\mathbf{R}}(D)$, $\overline{\mathbf{h}}_{0L}$, and $\overline{\mathcal{F}}_L(\theta_t)$ are achieved by applying $\epsilon_1, \epsilon_2,$ and ϵ_3-differential privacy mechanisms. Furthermore, $\overline{\mathcal{F}}_L(\theta_t)$ and $\overline{\mathbf{h}}_{0L}$ access the same training batch L at each training step. Therefore, based on the composition theorem [61], the total privacy budget in Algorithm 3.1 must be the summation of ϵ_1, ϵ_2, and ϵ_3.

Consequently, Algorithm 3.1 preserves ϵ-differential privacy, where $\epsilon = \epsilon_1 + \epsilon_2 + \epsilon_3$. □

Note that $\Delta_{\mathbf{R}}$ and $\Delta_{\mathbf{h}_0}$ are dependent on the number of input features d. $\Delta_{\mathbf{R}}$ is negatively proportional to the number of tuples in D. The larger the size of D, the less noise will be injected into the private relevance $\overline{\mathbf{R}}$. $\Delta_{\mathcal{F}}$ is only dependent on the number of neurons in the last hidden layer and the output layer. In addition, $\Delta_{\mathbf{R}}, \Delta_{\mathbf{h}_0}$, and $\Delta_{\mathcal{F}}$ do not depend on the number of training epochs. Consequently:

- **(1)** The privacy budget consumption in our model is totally independent of the number of training epochs.
- **(2)** In order to improve the model utility under differential privacy, our mechanism adaptively injects Laplace noise into features based on the contribution of each to the model output.
- **(3)** The average error incurred by our approximation approach, $\widehat{\mathcal{F}}_L(\theta)$, is bounded by a small number $M \times \frac{e^2+2e-1}{e(1+e)^2}$ as stated in the Lemma 3.7 in the subsection **Approximation Error Bounds** below.

- **(4)** The proposed mechanism can be applied to a variety of deep learning models, e.g., CNNs [47], deep auto-encoders [39], Restricted Boltzmann Machines [162], convolution deep belief networks [163], etc., as long as we can perturb the first affine transformation layer.

With these characteristics, our mechanism has a great potential to be applied in large datasets, without consuming excessive privacy budgets. In the experiment subsection, we will show that our mechanism leads to accurate results.

Approximation Error Bounds The following lemma illustrates the result of how much error our approximation approach, $\widehat{\mathcal{F}}_L(\theta)$ (Equation (3.27)), incurs. The error only depends on the number of possible classification outcomes M. In addition, the average error of the approximations is always bounded. As in [60, 161], the approximation of the loss function $\mathcal{F}_L(\theta)$ by applying Taylor Expansion without removing all polynomial terms with order larger than 2 is as follows:

$$\widetilde{\mathcal{F}}_L(\theta) = \sum_{l=1}^{M} \sum_{\mathbf{x}_i \in L} \sum_{q=1}^{2} \sum_{R=0}^{\infty} \frac{f_{ql}^{(R)}(z_{ql})}{R!} \big(g_{ql}(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}, W_{l(k)}) - z_{ql}\big)^R$$

$\forall l \in \{1, \dots, M\}$, let f_{1l}, f_{2l}, g_{1l}, and g_{2l} be four functions defined as follows:

$$\begin{aligned} g_{1l}(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}, W_j) &= \overline{\mathfrak{h}}_{\mathbf{x}_i(k)} W_{l(k)}^T \\ g_{2l}(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}, W_j) &= \overline{\mathfrak{h}}_{\mathbf{x}_i(k)} W_{l(k)}^T \\ f_{1l}(z_{1l}) &= y_{il} \log(1 + e^{-z_{1l}}) \\ f_{2l}(z_{2l}) &= (1 - y_{il}) \log(1 + e^{z_{2l}}) \end{aligned} \tag{3.30}$$

where $\forall q, l : z_{ql}$ is a real number.

$\forall q, l$, by setting $z_{ql} = 0$, the above equation can be simplified as:

$$\widetilde{\mathcal{F}}_L(\theta) = \sum_{l=1}^{M} \sum_{\mathbf{x}_i \in L} \sum_{q=1}^{2} \sum_{R=0}^{\infty} \frac{f_{ql}^{(R)}(0)}{R!} \big(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)} W_{l(k)}^T\big)^R \tag{3.31}$$

As in [60], our approximation approach works by truncating the Taylor series in Equation (3.31) to remove all polynomial terms with order larger than 2. This leads to a new objective function in Equation (3.27) with low-order polynomials as follows:

$$\begin{aligned}
\widehat{\mathcal{F}}_L(\theta) &= \sum_{l=1}^{M} \sum_{\mathbf{x}_i \in L} \sum_{q=1}^{2} \sum_{R=0}^{2} \frac{f_{ql}^{(R)}(0)}{R!} \big(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)} W_{l(k)}^T\big)^R \\
&= \sum_{l=1}^{M} \sum_{\mathbf{x}_i \in L} \Big[\sum_{q=1}^{2} f_{ql}^{(0)}(0) + \big(\sum_{q=1}^{2} f_{ql}^{(1)}(0) \big) \overline{\mathfrak{h}}_{\mathbf{x}_i(k)} W_{l(k)}^T \\
&\qquad + \big(\sum_{q=1}^{2} \frac{f_{ql}^{(2)}(0)}{2!} \big) \big(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)} W_{l(k)}^T\big)^2 \Big]
\end{aligned}$$

We are now ready to state the following lemma to show the approximation error bound of our approach.

Lemma 3.7. *Given two polynomial functions $\widetilde{\mathcal{F}}_L(\theta)$ (Equation (3.31)) and $\widehat{\mathcal{F}}_L(\theta)$ (Equation (3.27)), the average error of the approximation is always bounded as follows:*

$$|\widetilde{\mathcal{F}}_L(\widehat{\theta}) - \widetilde{\mathcal{F}}_L(\widetilde{\theta})| \leq M \times \frac{e^2 + 2e - 1}{e(1+e)^2} \tag{3.32}$$

where $\widetilde{\theta} = \arg\min_\theta \widetilde{\mathcal{F}}_L(\theta)$ and $\widehat{\theta} = \arg\min_\theta \widehat{\mathcal{F}}_L(\theta)$.

Proof of Lemma 3.7:

Proof. Let $\widetilde{\theta} = \arg\min_\theta \widetilde{\mathcal{F}}_L(\theta)$ and $\widehat{\theta} = \arg\min_\theta \widehat{\mathcal{F}}_L(\theta)$, $U = \max_\theta \big(\widetilde{\mathcal{F}}_L(\theta) - \widehat{\mathcal{F}}_L(\theta)\big)$ and $S = \min_\theta \big(\widetilde{\mathcal{F}}_L(\theta) - \widehat{\mathcal{F}}_L(\theta)\big)$. We have that $U \geq \widetilde{\mathcal{F}}_L(\widehat{\theta}) - \widehat{\mathcal{F}}_L(\widehat{\theta})$ and $\forall\theta^* : S \leq \widetilde{\mathcal{F}}_L(\theta^*) - \widehat{\mathcal{F}}_L(\theta^*)$. Therefore, we have:

$$\begin{aligned}
&\widetilde{\mathcal{F}}_L(\widehat{\theta}) - \widehat{\mathcal{F}}_L(\widehat{\theta}) - \widetilde{\mathcal{F}}_L(\theta^*) + \widehat{\mathcal{F}}_L(\theta^*) \leq U - S \\
&\Leftrightarrow \widetilde{\mathcal{F}}_L(\widehat{\theta}) - \widetilde{\mathcal{F}}_L(\theta^*) \leq U - S + \big(\widehat{\mathcal{F}}_L(\widehat{\theta}) - \widehat{\mathcal{F}}_L(\theta^*)\big)
\end{aligned}$$

In addition, $\widehat{\mathcal{F}}_L(\widehat{\theta}) - \widehat{\mathcal{F}}_L(\theta^*) \leq 0$, so $\widetilde{\mathcal{F}}_L(\widehat{\theta}) - \widetilde{\mathcal{F}}_L(\theta^*) \leq U - S$. If $U \geq 0$ and $S \leq 0$ then we have:

$$|\widetilde{\mathcal{F}}_L(\widehat{\theta}) - \widetilde{\mathcal{F}}_L(\theta^*)| \leq U - S \tag{3.33}$$

Equation (3.33) holds for every θ^*. Therefore, it still holds for $\widetilde{\theta}$. Equation (3.33) shows that the error incurred by truncating the Taylor series approximate function depends on the maximum and minimum values of $\widetilde{\mathcal{F}}_L(\theta) - \widehat{\mathcal{F}}_L(\theta)$. To quantify the magnitude of the error, we first rewrite $\widetilde{\mathcal{F}}_L(\theta) - \widehat{\mathcal{F}}_L(\theta)$ as:

$$\begin{aligned}
\widetilde{\mathcal{F}}_L(\theta) - \widehat{\mathcal{F}}_L(\theta) &= \sum_{l=1}^{M} \Big[\widetilde{\mathcal{F}}_L(W_{l(k)}) - \widehat{\mathcal{F}}_L(W_{l(k)})\Big] \\
&= \sum_{l=1}^{M} \Big[\sum_{\mathbf{x}_i \in L} \sum_{q=1}^{2} \sum_{R=3}^{\infty} \frac{f_{ql}^{(R)}(z_{ql})}{R!} \big(g_{ql}(\overline{\mathfrak{h}}_{\mathbf{x}_i(k)}, W_{l(k)}) - z_{ql}\big)^R \Big]
\end{aligned}$$

To derive the minimum and maximum values of the function above, we look into the remainder of the Taylor Expansion for each l. Let $z_l \in [z_{ql} - 1, z_{ql} + 1]$. According to the well-known result [164], $\frac{1}{|D|}\big(\widetilde{\mathcal{F}}_L(W_{l(k)}) - \widehat{\mathcal{F}}_L(W_{l(k)})\big)$ must be in the interval $\Big[\sum_q \frac{\min_{z_l} f_{ql}^{(3)}(z_l)(z_l - z_{ql})^3}{6}, \sum_l \frac{\max_{z_l} f_{ql}^{(3)}(z_l)(z_l - z_{ql})^3}{6}\Big]$.

If $\sum_q \frac{\max_{z_l} f_{ql}^{(3)}(z_l)(z_l - z_{ql})^3}{6} \geq 0$ and $\sum_q \frac{\min_{z_l} f_{ql}^{(3)}(z_l)(z_l - z_{ql})^3}{6} \leq 0$, then we have that:

$$\Big|\frac{1}{|L|}\Big[\widetilde{\mathcal{F}}_L(\theta) - \widehat{\mathcal{F}}_L(\theta)\Big]\Big| \\ \leq \sum_{l=1}^{M}\sum_q \frac{\max_{z_l} f_{ql}^{(3)}(z_l)(z_l - z_{ql})^3 - \min_{z_l} f_{ql}^{(3)}(z_l)(z_l - z_{ql})^3}{6} \tag{3.34}$$

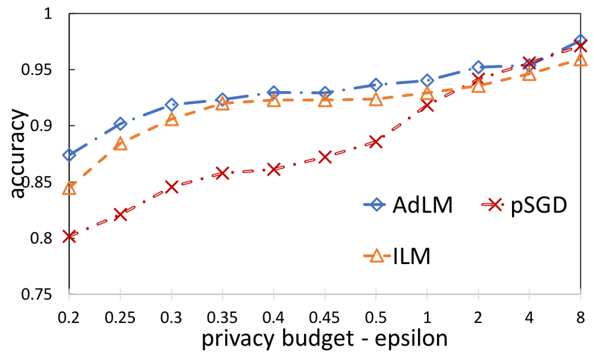
This analysis applies to the case of the cross-entropy error-based loss function as follows. First, for the functions $f_{1l}(z_{1l}) = y_{il}\log(1+e^{-z_{1l}})$ and $f_{2l}(z_{2l}) = (1-y_{il})\log(1+e^{z_{2l}})$, we have

$$f_{1l}^{(3)}(z_{1l}) = \frac{2y_{il}e^{z_{1l}}}{(1+e^{z_{1l}})^3}$$
$$f_{2l}^{(3)}(z_{2l}) = (1-y_{il})\frac{e^{-z_{2l}}(e^{-z_{2l}}-1)}{(1+e^{-z_{2l}})^3}$$

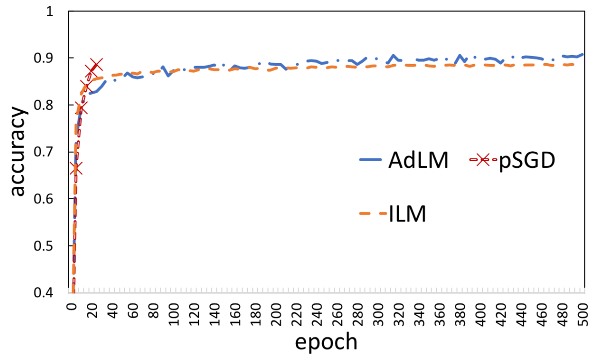
It can be verified that $\arg\min_{z_{1l}} f_{1l}^{(3)}(z_{1l}) = \frac{-2e}{(1+e)^3} < 0$, $\arg\max_{z_{1l}} f_{1l}^{(3)}(z_{1l}) = \frac{2e}{(1+e)^3} > 0$, $\arg\min_{z_{2l}} f_{2l}^{(3)}(z_{2l}) = \frac{1-e}{e(1+e)^3} < 0$, and $\arg\max_{z_{2l}} f_{2l}^{(3)}(z_{2l}) = \frac{e(e-1)}{(1+e)^3} > 0$. Thus, the average error of the approximation is at most

$$\big|\widetilde{\mathcal{F}}_L(\widehat{\theta}) - \widetilde{\mathcal{F}}_L(\widetilde{\theta})\big| \leq M \times \Big[\big(\frac{2e}{(1+e)^3} - \frac{-2e}{(1+e)^3}\big) \\ + \big(\frac{e(e-1)}{(1+e)^3} - \frac{1-e}{e(1+e)^3}\big)\Big] = M \times \frac{e^2+2e-1}{e(1+e)^2}$$

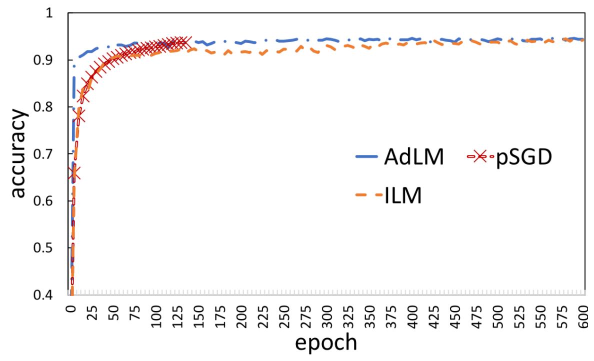
Therefore, Equation (3.32) holds. □



(a) accuracy vs. ϵ



(b) $\epsilon = 0.5$ (large noise)

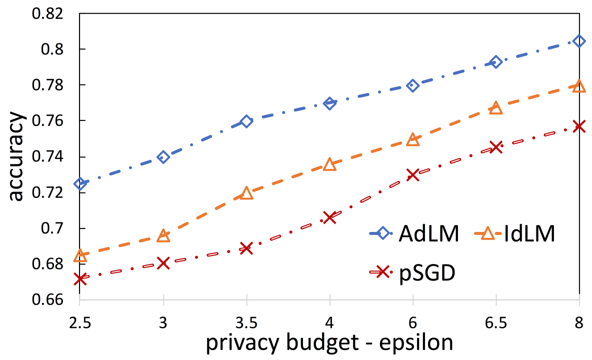


(c) $\epsilon = 2.0$ (small noise)

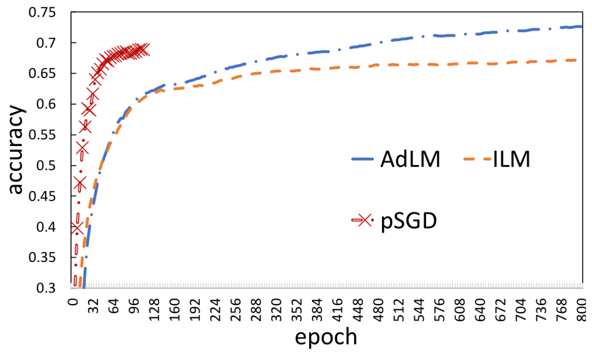
Figure 3.4 Accuracy for different noise levels on the MNIST dataset.

3.1.10 Experimental Results

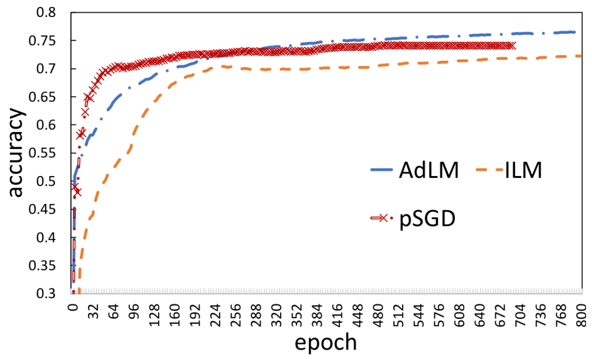
We have carried out an extensive experiment on two well-known image datasets, MNIST and CIFAR-10. The MNIST database of handwritten digits consists of 60,000 training



(a) accuracy vs. ϵ



(b) $\epsilon = 2.5$ (large noise)



(c) $\epsilon = 8.0$ (small noise)

Figure 3.5 Accuracy for different noise levels on the CIFAR-10 dataset.

examples, and a test set of 10,000 examples [47]. Each example is a 28×28 size gray-level image. The CIFAR-10 dataset consists of color images categorized into 10 classes, such as birds, dogs, trucks, airplanes, etc. The dataset is partitioned into 50,000 training examples and 10,000 test examples [99].

Competitive Models We compare our mechanism with the state-of-the-art differentially private stochastic gradient descent (**pSGD**) for deep learning proposed by [62]. CNNs are used in our experiments for both algorithms. The hyper-parameters in pSGD are set to the default values recommended by Abadi et al. [62]. To comprehensively examine the proposed approaches, our mechanism is implemented in two different settings: **(1)** The Adaptive Laplace Mechanism (Algorithm 3.1)-based CNN with ReLUs, simply denoted **AdLM**; and **(2)** An **I**dentical **L**aplace **M**echanism-based CNN with ReLUs (**ILM**), in which an identical Laplace noise $\frac{1}{|L|}Lap(\frac{\Delta_{\mathbf{h}_0}}{\epsilon_2})$ is injected into each feature x_{ij} to preserve ϵ_2-differential privacy in the computation of the affine transformation layer $\mathbf{h}_0$. In the ILM algorithm (described in Subsection 3.1.7), we do not need to use the differentially private relevances $\overline{\mathbf{R}}(D)$.

MNIST Dataset The designs of the three models are the same on the MNIST dataset. We used two convolution layers, one with 32 features and one with 64 features. Each hidden neuron connects with a 5x5 unit patch. A fully-connected layer with 25 units and an output layer of 10 classes (i.e., 10 digits) with cross-entropy loss with LRN are used. The batch size is 1,800. This also is the structure of the pre-trained model, which is learned and used to compute the average relevances $\mathbf{R}(D)$. The experiments were conducted on a single GPU, i.e., NVIDIA GTX TITAN X, 12 GB with 3,072 CUDA cores.

Figure 3.4 a illustrates the prediction accuracy of each model as a function of the privacy budget ϵ on the MNIST dataset. It is clear that our models, i.e., AdLM and ILM, outperform the pSGD, especially when the privacy budget ϵ is small. This is a crucial result, since smaller privacy budget values enforce stronger privacy guarantees. When the privacy budget ϵ is large, e.g., $\epsilon = 2, 4, 8$, which means small noise is injected into the model, the efficiencies of all the models are almost converged to higher prediction accuracies.

The AdLM model achieves the best performance. Given a very small privacy budget $\epsilon = 0.25$, it achieves 90.2% in terms of prediction accuracy, compared with 88.46% obtained by the ILM and 82.09% obtained by the pSGD. Overall, given small values of the privacy budget ϵ, i.e., $0.2 \leq \epsilon \leq 0.5$, the AdLM improves the prediction accuracy by 7.7% on average (i.e., 91.62%) compared with the pSGD (i.e., 83.93%). The result is statistically significant with $p < 0.01$ (t-test).

Figures 3.4b-c illustrate the prediction accuracy of each model vs. the number of epochs under $\epsilon = 0.5$ and $\epsilon = 2.0$, respectively. Given large noise, i.e., $\epsilon = 0.5$, the pSGD quickly achieves higher prediction accuracies (i.e., 88.59%) after a small number of epochs, compared with other models (Figure 3.4b). However, the pSDG can only be applied to train the model by using a limited number of epochs; specifically because the privacy budget is accumulated after every training step. Meanwhile, our mechanism is totally independent of the number of epochs in the consumption of privacy budget. Therefore, after 500 epochs, our models outperform the pSGD. The AdLM achieves the best performance, in terms of prediction accuracy: 93.66%, whereas the ILM and the pSGD reached only 92.39% and 88.59%, respectively. Interestingly, given small noise, i.e., $\epsilon = 2.0$, our models achieve higher accuracies than the pSGD after a small number of epochs (Figure 3.4c). This result illustrates the crucial benefits of being independent of the number of training epochs in preserving differential privacy in deep learning. With our mechanism, we can keep training our models without accumulating noise and privacy budget.

CIFAR-10 Dataset The designs of the three models are the same on the CIFAR-10 dataset. We used three convolution layers, two with 128 features and one with 256 features. Each hidden neuron connects with a 3x3 unit patch in the first layer, and a 5x5 unit patch in other layers. One fully-connected layer with 30 neurons, and an output layer of 10 classes with a cross-entropy loss with LRN are used. The batch size is set to 7,200. This also is the structure of the pre-trained model, which is learned and used to compute the average relevances $\mathbf{R}(D)$.

Figure 3.5a shows the prediction accuracies of each model as a function of the privacy budget ϵ on the CIFAR-10 dataset. Figures 3.5b-c illustrate the prediction accuracy of each model vs. the number of epochs under different noise levels. Similar to the results on the MNIST dataset, the results on CIFAR-10 strengthen our observations: **(1)** Our mechanism outperforms the pSGD in terms of prediction accuracy, given both modest and large values of the privacy budget ϵ (Figure 3.5a); and **(2)** Our mechanism has the ability to work with large-scale datasets, since it is totally independent of the number of training epochs in the consumption of privacy budget (Figures 3.5b-c).

In fact, the AdLM improves the prediction accuracy by 5.9% on average (i.e., to 77%) compared with the pSGD (i.e., 71.1%). The result is statistically significant with $p < 0.01$ (t-test). Given large noise, i.e., $\epsilon = 2.5$, our models including the AdLM and ILM outperform the pSGD after 800 epochs (Figure 3.5b).

Adaptive Laplace Noise It is important to note that by adaptively redistributing the noise into input features based on the relevance of each to the model output, we can achieve much better prediction accuracies in both MNIST and CIFAR-10 datasets given both small and large values of privacy budget ϵ. This is clearly demonstrated in Figures 3.4a and 3.5a, since the AdLM outperforms the ILM in all cases. Overall, the AdLM improves the prediction accuracy by 2% and 5% on average on MNIST and CIFAR-10 datasets correspondingly, compared with the ILM. The result is statistically significant with $p < 0.05$ (t-test). Note that the ILM injected an identical amount of noise into all input features, regardless of their contributions to the model output. This is an important result, since our mechanism is the first of its kind, which can redistribute the noise injected into the deep learning model to improve the utility. In addition, the reallocation of ϵ_1, ϵ_2, and ϵ_3 could further improve the utility. This is an open research direction in the future work.

Computational Efficiency In terms of computation efficiency, there are two differences in our mechanism, compared with a regular deep neural network: **(i)** The pre-trained model; and **(ii)** The noise injection task. In practice, the pre-trained model is not necessarily identical to the differentially private network trained by our AdLM. A simple model can be used as a pre-trained model to approximate the average relevance $\mathbf{R}(D)$, as long as the pre-trained model is effective in terms of prediction accuracy even over a small training dataset. Achieving this is quite straight-forward, because: **(1)** The pre-trained model is noiseless; and **(2)** The number of training epochs used to learn a pre-trained model is small compared with the one of differentially private models. In fact, we only correspondingly need 12 and 50 extra epochs to learn the pre-trained models on MNIST and CIFAR-10 datasets. Training pre-trained models takes about 10 minutes on a single GPU, i.e., NVIDIA GTX TITAN X, 12 GB with 3,072 CUDA cores. Therefore, the model pre-training for the computation of $\mathbf{R}(D)$ is efficient.

Another difference in our mechanism is the noise injection into input attributes and coefficients of the loss function $\widehat{\mathcal{F}}$. In this task, the computations of $\Delta_{\mathbf{R}}$, $\Delta_{\mathbf{h}_0}$, $\Delta_{\mathcal{F}}$, $\overline{R}_j$, β_j, $\overline{x}$, and $\overline{\phi}$ are efficient and straight-forward, since there is not any operation such as arg min, arg max, sorting, etc. The complexity of these computations is $O\big(|D|(d + M)\big)$, which is linear to the size of the database D. In addition, these computations can be efficiently performed in either a serial process or a parallel process. Therefore, this task does not affect the computational efficiency of our mechanism much.

3.1.11 Conclusions

In this part of the dissertation, we proposed a novel mechanism, called Adaptive Laplace Mechanism (AdLM), to preserve differential privacy in deep learning. Our mechanism conducts both sensitivity analysis and noise insertion on deep neural networks. It is totally independent of the number of training epochs in the consumption of privacy budget. That makes our mechanism more practical. In addition, our mechanism is the first of its kind to have the ability to redistribute the noise insertion toward the improvement of model utility in deep learning. In fact, our mechanism has the ability to intentionally add *more noise* into input features which are *less relevant* to the model output, and vice-versa. Different activation functions can be applied in our mechanism, as well. These distinctive characteristics guarantee the ability to apply our mechanism on large datasets in different deep learning models and in different contexts. Our mechanism can clearly enhance the application of differential privacy in deep learning. Rigorous experimental evaluations conducted on well-known datasets validated our theoretical results and the effectiveness of our mechanism.

3.2 Scalable Differential Privacy with Certified Robustness in Adversarial Learning

3.2.1 Background

In this subsection, we revisit DP, adversarial learning, and certified robustness.

Differential Privacy Let D be a database that contains N tuples, each of which contains data $x \in [-1, 1]^d$ and a *ground-truth label* $y \in \mathbb{Z}_K$ (one-hot vector), with K possible categorical outcomes $y = \{y_1, \ldots, y_K\}$. A single *true class label* $y_x \in y$ given $x \in D$ is assigned to only one of the K categories. On input x and parameters θ, a model outputs class scores $f : \mathbb{R}^d \to \mathbb{R}^K$ that maps x to a vector of scores $f(x) = \{f_1(x), \ldots, f_K(x)\}$ s.t. $\forall k \in [1, K] : f_k(x) \in [0, 1]$ and $\sum_{k=1}^{K} f_k(x) = 1$. The class with the highest score value is selected as the *predicted label* for x, denoted as $y(x) = \max_{k \in K} f_k(x)$. A loss function $L(f(x), y)$ presents the penalty for mismatching between the predicted values $f(x)$ and original values y. The notations and terminologies used in this work are summarized in

Table 3.1 of Subsection 3.2.2). Let us briefly revisit DP DNNs, starting with the definition of DP.

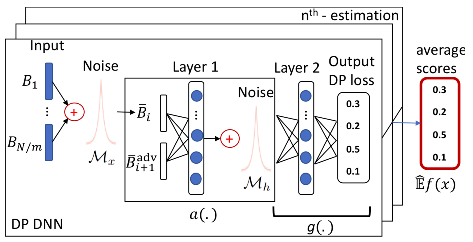
Definition 3.2. *(ϵ, δ)-DP [55]. A randomized algorithm A fulfills (ϵ, δ)-DP, if for any two databases D and D' differing at most one tuple, and for all $O \subseteq Range(A)$, we have:*

$$Pr[A(D) = O] \leq e^{\epsilon} Pr[A(D') = O] + \delta$$

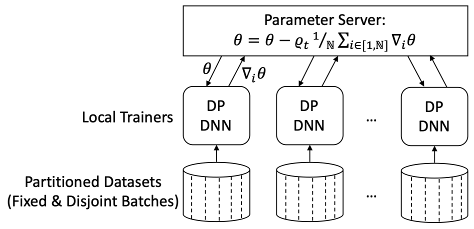
ϵ controls the amount by which the distributions induced by D and D' may differ, δ is a broken probability.

DP also applies to general metrics $\rho(D, D') \leq 1$, where ρ can be l_p-norms [165]. DP-preserving algorithms in DNNs can be categorized into three lines: 1) introducing noise into *parameter gradients* [62, 166, 59, 73, 74, 76], 2) injecting noise into objective functions [60, 71, 72], and 3) injecting noise into labels [167].

Adversarial Learning For some target model f and inputs (x, y_x), the adversary's goal is to find an *adversarial example* $x^{\text{adv}} = x + \alpha$, where α is the perturbation introduced by the attacker, such that: **(1)** x^{adv} and x are close, and **(2)** the model misclassifies x^{adv}, i.e., $y(x^{\text{adv}}) \neq y(x)$. In this work, *we consider well-known $l_{p \in \{1,2,\infty\}}(\mu)$-norm bounded attacks [168], where μ is the radius of the p-norm ball.* To improve the robustness of models, prior work focused on two directions: 1) Producing correct predictions on adversarial examples, while not compromising the accuracy on legitimate inputs [66, 67, 169, 170, 171, 172, 173, 174]; and 2) Detecting adversarial examples [175, 176, 177, 178, 179]. Among existing solutions, adversarial training appears to hold the greatest promise for learning robust models [180]. A well-known algorithm was proposed in [181]. The algorithm is revisited in the Algorithm 3.2 in the following subsection.



(a) An instance of DP DNNs and verified inference


Parameter Server:
$\theta = \theta - \varrho_t \, ^1/_{\mathbb{N}} \sum_{i\in[1,\mathbb{N}]} \nabla_i\theta$
θ
$\nabla_i\theta$
Local Trainers
DP DNN
DP DNN
...
DP DNN
Partitioned Datasets
(Fixed & Disjoint Batches)

(b) An instance of stochastic batch training

Figure 3.6 Stochastic Batch mechanism.

Pseudo-code of Adversarial Training [181] Let $l_p(\mu) = \{\alpha \in \mathbb{R}^d : \|\alpha\|_p \leq \mu\}$ be the l_p-norm ball of radius μ. One of the goals in adversarial learning is to minimize the risk over adversarial examples: $\theta^* = \arg\min_\theta \mathbb{E}_{(x,y_{\text{true}})\sim\mathcal{D}}\big[\max_{\|\alpha\|_p\leq\mu} L\big(f(x+\alpha,\theta), y_x\big)\big]$, where an attack is used to approximate solutions to the inner maximization problem, and the outer minimization problem corresponds to training the model f with parameters θ over these adversarial examples $x^{\text{adv}} = x + \alpha$. There are two basic adversarial example attacks. The first one is a *single-step* algorithm, e.g., **FGSM** algorithm [168], in which only a single gradient computation is required to find adversarial examples by solving the inner maximization $\max_{\|\alpha\|_p\leq\mu} L\big(f(x+\alpha,\theta), y_x\big)$. The second one is an *iterative* algorithm, e.g., **Iterative-FGSM** algorithm [182], in which multiple gradients are computed and updated in T_μ small steps, each of which has a size of μ/T_μ.

Given a loss function:

$$L(\theta) = \frac{1}{m_1 + \xi m_2}\Big(\sum_{x_i\in B_t} \mathcal{L}\big(f(x_i,\theta), y_i\big) + \xi \sum_{x_j^{\text{adv}}\in B_t^{\text{adv}}} \Upsilon\big(f(x_j^{\text{adv}},\theta), y_j\big)\Big)$$

where m_1 and m_2 correspondingly are the numbers of examples in B_t and B_t^{adv} at each training step. Algorithm 3.2 presents the vanilla adversarial training.

Algorithm 3.2 Adversarial Training [181]

```
Input: Database $D$, loss function $L$, parameters $\theta$, batch sizes $m_1$ and $m_2$, learning rate $\varrho_t$, parameter $\xi$
1: Initialize $\theta$ randomly
2: for $t \in [T]$ do
3:     Take a random batch $B_t$ with the size $m_1$, and a random batch $B_a$ with the size $m_2$
4:     Craft adversarial examples $B_t^{\text{adv}} = \{x_j^{\text{adv}}\}_{j\in[1,m_2]}$ from corresponding benign examples $x_j \in B_a$
5:     Descent: $\theta \leftarrow \theta - \varrho_t \nabla_\theta L(\theta)$
```

Certified Robustness and DP Recently, some algorithms [68, 69, 77, 183, 184, 70] have been proposed to derive certified robustness, in which each prediction is guaranteed to be consistent under the perturbation α, if a robustness condition is held. Given a benign example x, we focus on achieving a robustness condition to $l_p(\mu)$-norm attacks, as follows:

$$\forall \alpha \in l_p(\mu) : f_k(x+\alpha) > \max_{i:i\neq k} f_i(x+\alpha) \tag{3.35}$$

where $k = y(x)$, indicating that a small perturbation α in the input does not change the predicted label $y(x)$. To achieve the robustness condition in Equation (3.35), [101] introduce an algorithm, called **PixelDP**. By considering an input x (e.g., images) as databases in DP parlance, and individual features (e.g., pixels) as tuples, PixelDP shows that randomizing the scoring function $f(x)$ to enforce DP on a small number of pixels in an image guarantees robustness of predictions. To randomize $f(x)$, *random noise* σ_r is injected into either input x or an arbitrary hidden layer, resulting in the following (ϵ_r, δ_r)-PixelDP condition:

Lemma 3.8. *(ϵ_r, δ_r)-PixelDP [101]. Given a randomized scoring function $f(x)$ satisfying (ϵ_r, δ_r)-PixelDP w.r.t. a l_p-norm metric, we have:*

$$\forall k, \forall \alpha \in l_p(1) : \mathbb{E} f_k(x) \leq e^{\epsilon_r} \mathbb{E} f_k(x + \alpha) + \delta_r \tag{3.36}$$

where $\mathbb{E} f_k(x)$ is the expected value of $f_k(x)$, ϵ_r is a predefined budget, δ_r is a broken probability.

At the prediction time, a certified robustness check is implemented for each prediction, as follows:

$$\hat{\mathbb{E}}_{lb} f_k(x) > e^{2\epsilon_r} \max_{i:i \neq k} \hat{\mathbb{E}}_{ub} f_i(x) + (1 + e^{\epsilon_r})\delta_r \tag{3.37}$$

where $\hat{\mathbb{E}}_{lb}$ and $\hat{\mathbb{E}}_{ub}$ are the lower and upper bounds of the expected value $\hat{\mathbb{E}} f(x) = \frac{1}{n} \sum_n f(x)_n$, derived from the Monte Carlo estimation with an η-confidence, given n is the number of invocations of $f(x)$ with independent draws in the noise σ_r. Passing the check for a given input guarantees that no perturbation up to $l_p(1)$-norm can change the model's prediction. PixelDP *does not* preserve DP in learning private parameters θ to protect the training data.

Functional Mechanism Functional mechanism [161] achieves ϵ-DP by perturbing the objective function $L_D(\theta)$ and then releasing the model parameter $\overline{\theta}$ minimizing the perturbed objective function $\overline{L}_D(\theta)$ instead of the original θ, given a private training dataset D. The mechanism exploits the polynomial representation of $L_D(\theta)$. The model parameter θ is a vector that contains **d** values $\theta_1, \ldots, \theta_{\mathbf{d}}$. Let $\phi(\theta)$ denote a product of $\theta_1, \ldots, \theta_{\mathbf{d}}$, namely, $\phi(\theta) = \theta_1^{c_1} \cdot \theta_2^{c_2} \cdots \theta_{\mathbf{d}}^{c_{\mathbf{d}}}$ for some $c_1, \ldots, c_{\mathbf{d}} \in \mathbb{N}$. Let $\Phi_j (j \in \mathbb{N})$ denote the set of all products of $\theta_1, \ldots, \theta_{\mathbf{d}}$ with degree j, i.e., $\Phi_j = \left\{ \theta_1^{c_1} \cdot \theta_2^{c_2} \cdots \theta_{\mathbf{d}}^{c_{\mathbf{d}}} \middle| \sum_{a=1}^{\mathbf{d}} c_a = j \right\}$. By the Stone-Weierstrass Theorem [185], any continuous and differentiable $L(x_i, \theta)$ can always be written as a polynomial of $\theta_1, \ldots, \theta_{\mathbf{d}}$, for some $J \in [0, \infty]$, i.e., $L(x_i, \theta) = \sum_{j=0}^{J} \sum_{\phi \in \Phi_j} \lambda_{\phi x_i} \phi(\theta)$ where $\lambda_{\phi x_i} \in \mathbb{R}$ denotes the coefficient of $\phi(\theta)$ in the polynomial.

For instance, the polynomial expression of the loss function in the linear regression is as follows: $L(x_i, \theta) = (y_i - x_i^\top \theta)^2 = y_i^2 - \sum_{j=1}^{d} (2y_i x_{ij})\theta_j + \sum_{1 \leq j,a \leq d} (x_{ij} x_{ia})\theta_j \theta_a$, where d $(= \mathbf{d})$ is the number of features in x_i. In fact, $L(x_i, \theta)$ only involves monomials in $\Phi_0 = \{1\}, \Phi_1 = \{\theta_1, \ldots, \theta_d\}$, and $\Phi_2 = \{\theta_i \theta_a \big| i, a \in [1, d]\}$. Each $\phi(\theta)$ has its own coefficient, e.g., for θ_j, its polynomial coefficient $\lambda_{\phi_{x_i}} = -2y_i x_{ij}$. Similarly, $L_D(\theta)$ can be expressed as a polynomial of $\theta_1, \ldots, \theta_d$, as

$$L_D(\theta) = \sum_{x_i \in D} L(x_i, \theta) = \sum_{j=0}^{J} \sum_{\phi \in \Phi_j} \sum_{x_i \in D} \lambda_{\phi x_i} \phi(\theta) \tag{3.38}$$

To achieve ϵ-DP, $L_D(\theta)$ is perturbed by injecting Laplace noise $Lap(\frac{\Delta}{\epsilon})$ into its polynomial coefficients λ_ϕ, and then the model parameter $\overline{\theta}$ is derived to minimize the perturbed function $\overline{L}_D(\theta)$, where the global sensitivity $\Delta = 2 \max_x \sum_{j=1}^{J} \sum_{\phi \in \Phi_j} \|\lambda_{\phi x}\|_1$ is derived given any two neighboring datasets. To guarantee that the optimization of $\overline{\theta} = \arg\min_\theta \overline{L}_D(\theta)$ achieves ϵ-DP without accessing the original data, i.e., that may potentially incur additional privacy leakage, grid search-based approaches are applied to learn the ϵ-DP parameters $\overline{\theta}$ with low loss $\overline{L}_D(\theta)$. Although this approach works well in simple tasks, i.e., logistic regression, it may not be optimal in large models, such as DNNs.

3.2.2 Notations and Terminologies

All notations and terminologies used in this work are described in Table 3.1.

3.2.3 Stochastic Batch (StoBatch) Mechanism

StoBatch is presented in Algorithm 3.5. Our DNN (Figure 3.6a) is presented as: $f(x) = g(a(x, \theta_1), \theta_2)$, where $a(x, \theta_1)$ is a feature representation learning model with x as an input,

Table 3.1 Notations and Terminologies.

D and x	Training data with benign examples $x \in [-1,1]^d$
$y = \{y_1, \dots, y_K\}$	One-hot label vector of K categories
$f : \mathbb{R}^d \to \mathbb{R}^K$	Function/model f that maps inputs x to a vector of scores $f(x) = \{f_1(x), \dots, f_K(x)\}$
$y_x \in y$	A single true class label of example x
$y(x) = \max_{k \in K} f_k(x)$	Predicted label for the example x given the function f
$x^{\text{adv}} = x + \alpha$	Adversarial example where α is the perturbation
$l_p(\mu) = \{\alpha \in \mathbb{R}^d : \|\alpha\|_p \leq \mu\}$	The l_p-norm ball of attack radius μ
(ϵ_r, δ_r)	Robustness budget ϵ_r and broken probability δ_r
$\mathbb{E} f_k(x)$	The expected value of $f_k(x)$
$\hat{\mathbb{E}}_{lb}$ and $\hat{\mathbb{E}}_{ub}$	Lower and upper bounds of the expected value $\hat{\mathbb{E}} f(x) = \frac{1}{n} \sum_n f(x)_n$
$a(x, \theta_1)$	Feature representation learning model with x and parameters θ_1
B_t	A batch of benign examples x_i
$\mathcal{R}_{B_t}(\theta_1)$	Data reconstruction function given B_t in $a(x, \theta_1)$
$\mathbf{h}_{1B_t} = \{\theta_1^T x_i\}_{x_i \in B_t}$	The values of all hidden neurons in the hidden layer $\mathbf{h}_1$ of $a(x, \theta_1)$ given the batch B_t
$\widetilde{\mathcal{R}}_{B_t}(\theta_1)$ and $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$	Approximated and perturbed functions of $\mathcal{R}_{B_t}(\theta_1)$
$\overline{x}_i$ and $\widetilde{x}_i$	Perturbed and reconstructed inputs x_i
$\Delta_{\mathcal{R}} = d(\beta + 2)$	Sensitivity of the approximated function $\widetilde{\mathcal{R}}_{B_t}(\theta_1)$
$\overline{\mathbf{h}}_{1B_t}$	Perturbed affine transformation $\mathbf{h}_{1B_t}$
$\overline{x}_j^{\text{adv}} = x_j^{\text{adv}} + \frac{1}{m} Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$	DP adversarial examples crafting from benign example x_j
$\overline{B}_t$ and $\overline{B}_t^{\text{adv}}$	Sets of perturbed inputs $\overline{x}_i$ and DP adversarial examples $\overline{x}_j^{\text{adv}}$
$\mathcal{L}_{\overline{B}_t}(\theta_2)$	Loss function of perturbed benign examples in $\overline{B}_t$, given θ_2
$\Upsilon\big(f(\overline{x}_j^{\text{adv}}, \theta_2), y_j\big)$	Loss function of DP adversarial examples $\overline{x}_j^{\text{adv}}$, given θ_2
$\overline{\mathcal{L}}_{\overline{B}_t}(\theta_2)$	DP loss function for perturbed benign examples $\overline{B}_t$
$\mathcal{L}_{2\overline{B}_t}(\theta_2)$	A part of the loss function $\mathcal{L}_{\overline{B}_t}(\theta_2)$ that needs to be DP
$f(\mathcal{M}_1, \dots, \mathcal{M}_s \| x)$	Composition scoring function given independent randomizing mechanisms $\mathcal{M}_1, \dots, \mathcal{M}_s$
Δ_r^x and Δ_r^h	Sensitivities of x and h, given the perturbation $\alpha \in l_p(1)$
$(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$	Privacy budget to protect the training data D
$(\frac{\kappa\varphi}{\kappa+\varphi})_{max}$	Robustness size guarantee given an input x at the inference time

and g will take the output of $a(x, \theta_1)$ and return the class scores $f(x)$. At a high level, there are four key components: **(1)** DP $a(x, \theta_1)$, which is to preserve DP in learning the feature representation model $a(x, \theta_1)$; **(2)** DP Adversarial Learning, which focuses on preserving DP in adversarial learning, given DP $a(x, \theta_1)$; **(3)** Certified Robustness and Verified Inferring, which are to compute robustness bounds given an input at the inference time; and **(4)** Stochastic batch training (Figure 3.6b). To establish theoretical results in DP preservation and in deriving robustness bounds, let us first present our mechanism in the vanilla iterative batch-by-batch training (**Algorithm 3.3**). The network f (Lines 2-3, Algorithm 3.3) is trained over T training steps. In each step, a disjoint and fixed batch of m perturbed training examples and a disjoint and fixed batch of m DP adversarial examples, derived from D, are used to train our network (Lines 4-12, Algorithm 3.3).

Algorithm 3.3 Adversarial Learning with DP

```
Input: Database $D$, loss function $L$, parameters $\theta$, batch size $m$, learning rate $\varrho_t$, privacy budgets: $\epsilon_1$ and $\epsilon_2$, robustness parameters: $\epsilon_r$, $\Delta_r^x$, and $\Delta_r^h$, adversarial attack size $\mu_a$, the number of invocations $n$, ensemble attacks $A$, parameters $\psi$ and $\xi$, and the size $|\mathbf{h}_\pi|$ of $\mathbf{h}_\pi$
 1: Draw Noise $\chi_1 \leftarrow [Lap(\frac{\Delta_\mathcal{R}}{\epsilon_1})]^d$, $\chi_2 \leftarrow [Lap(\frac{\Delta_\mathcal{R}}{\epsilon_1})]^\beta$, $\chi_3 \leftarrow [Lap(\frac{\Delta_{\mathcal{L}2}}{\epsilon_2})]^{|\mathbf{h}_\pi|}$
 2: Randomly Initialize $\theta = \{\theta_1, \theta_2\}$, $\mathbf{B} = \{B_1, \ldots, B_{N/m}\}$ s.t. $\forall B \in \mathbf{B} : B$ is a batch with the size $m$, $B_1 \cap \ldots \cap B_{N/m} = \emptyset$, and $B_1 \cup \ldots \cup B_{N/m} = D$, $\overline{\mathbf{B}} = \{\overline{B}_1, \ldots, \overline{B}_{N/m}\}$ where $\forall i \in [1, N/m] : \overline{B}_i = \{\overline{x} \leftarrow x + \frac{\chi_1}{m}\}_{x \in B_i}$
 3: Construct a deep network $f$ with hidden layers $\{\mathbf{h}_1 + \frac{2\chi_2}{m}, \ldots, \mathbf{h}_\pi\}$, where $\mathbf{h}_\pi$ is the last hidden layer
 4: for $t \in [T]$ do
 5:     Take a batch $\overline{B}_i \in \overline{\mathbf{B}}$ where $i = t\%(N/m)$, $\overline{B}_t \leftarrow \overline{B}_i$
 6:     Ensemble DP Adversarial Examples:
 7:     Draw Random Perturbation Value $\mu_t \in (0, 1]$
 8:     Take a batch $\overline{B}_{i+1} \in \overline{\mathbf{B}}$, Assign $\overline{B}_t^{\text{adv}} \leftarrow \emptyset$
 9:     for $l \in A$ do
10:         Take the next batch $\overline{B}_a \subset \overline{B}_{i+1}$ with the size $m/|A|$
11:         $\forall \overline{x}_j \in \overline{B}_a$: Craft $\overline{x}_j^{\text{adv}}$ by using attack algorithm $A[l]$ with $l_\infty(\mu_t)$, $\overline{B}_t^{\text{adv}} \leftarrow \overline{B}_t^{\text{adv}} \cup \overline{x}_j^{\text{adv}}$
12:     Descent: $\theta_1 \leftarrow \theta_1 - \varrho_t \nabla_{\theta_1} \overline{\mathcal{R}}_{\overline{B}_t \cup \overline{B}_t^{\text{adv}}}(\theta_1)$; $\theta_2 \leftarrow \theta_2 - \varrho_t \nabla_{\theta_2} \overline{L}_{\overline{B}_t \cup \overline{B}_t^{\text{adv}}}(\theta_2)$ with the noise $\frac{\chi_3}{m}$
    Output: $\epsilon = (\epsilon_1 + \epsilon_1/\gamma_\mathbf{x} + \epsilon_1/\gamma + \epsilon_2)$-DP parameters $\theta = \{\theta_1, \theta_2\}$, robust model with an $\epsilon_r$ budget
```

3.2.4 DP Feature Representation Learning

Our idea is to use auto-encoder to simultaneously learn DP parameters θ_1 and ensure that the output of $a(x, \theta_1)$ is DP, since: (1) It is easier to train, given its small size; and (2) It can be reused for different predictive models. A typical data reconstruction function (cross-entropy), given a batch B_t at the training step t of the input x_i, is as follows: $\mathcal{R}_{B_t}(\theta_1) = \sum_{x_i \in B_t} \sum_{j=1}^{d} \big[x_{ij} \log(1 + e^{-\theta_{1j} h_i}) + (1 - x_{ij}) \log(1 + e^{\theta_{1j} h_i})\big]$, where $h_i = \theta_1^T x_i$, the hidden layer $\mathbf{h}_1$ of $a(x, \theta_1)$ given the batch B_t is denoted as $\mathbf{h}_{1B_t} = \{\theta_1^T x_i\}_{x_i \in B_t}$, and $\widetilde{x}_i = \theta_1 h_i$ is the reconstruction of x_i.

To preserve ϵ_1-DP in learning θ_1 where ϵ_1 is a privacy budget, we first derive the 1st-order polynomial approximation of $\mathcal{R}_{B_t}(\theta_1)$ by applying Taylor Expansion [160], denoted as $\widetilde{\mathcal{R}}_{B_t}(\theta_1)$. Then, *Functional Mechanism* [161] **(revisited in Subsection 3.2.1)** is adapted to inject noise into coefficients of the approximated function $\widetilde{\mathcal{R}}_{B_t}(\theta_1) = \sum_{x_i \in B_t} \sum_{j=1}^{d} \sum_{l=1}^{2} \sum_{r=0}^{1} \frac{\mathbf{F}_{lj}^{(r)}(0)}{r!} \big(\theta_{1j} h_i\big)^r$, where $\mathbf{F}_{1j}(z) = x_{ij} \log(1 + e^{-z})$, $\mathbf{F}_{2j}(z) = (1 - x_{ij}) \log(1 + e^{z})$, we have that: $\widetilde{\mathcal{R}}_{B_t}(\theta_1) = \sum_{x_i \in B_t} \sum_{j=1}^{d} \Big[\log 2 + \theta_{1j}\big(\frac{1}{2} - x_{ij}\big) h_i\Big]$. In $\widetilde{\mathcal{R}}_{B_t}(\theta_1)$, parameters θ_{1j} derived from the function optimization need to be ϵ_1-DP. To achieve that, Laplace noise $\frac{1}{m} Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$ is injected into coefficients $\big(\frac{1}{2} - x_{ij}\big) h_i$, where $\Delta_{\mathcal{R}}$ is the sensitivity of $\widetilde{\mathcal{R}}_{B_t}(\theta_1)$, as follows:

$$\begin{aligned} \widetilde{\mathcal{R}}_{B_t}(\theta_1) &= \sum_{x_i \in B_t} \sum_{j=1}^{d} \Big[\theta_{1j}\Big(\big(\frac{1}{2} - x_{ij}\big) h_i + \frac{1}{m} Lap\big(\frac{\Delta_{\mathcal{R}}}{\epsilon_1}\big)\Big)\Big] \\ &= \sum_{x_i \in B_t} \Big[\sum_{j=1}^{d} \big(\frac{1}{2} \theta_{1j} \overline{h}_i\big) - x_i \widetilde{x}_i\Big] \end{aligned} \tag{3.39}$$

To ensure that the computation of $\widetilde{x}_i$ does not access the original data, we further inject Laplace noise $\frac{1}{m} Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$ into x_i. This can be done as a preprocessing step for all the benign examples in D to construct a set of *disjoint* batches $\overline{\mathbf{B}}$ of perturbed benign examples (Lines 2 and 5, Algorithm 3.3). The perturbed function now becomes:

$$\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1) = \sum_{\overline{x}_i \in \overline{B}_t} \Big[\sum_{j=1}^{d} (\frac{1}{2}\theta_{1j}\overline{h}_i) - \overline{x}_i \widetilde{x}_i \Big] \tag{3.40}$$

where $\overline{x}_i = x_i + \frac{1}{m}Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$, $h_i = \theta_1^T \overline{x}_i$, $\overline{h}_i = h_i + \frac{2}{m}Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$, and $\widetilde{x}_i = \theta_1 \overline{h}_i$. Let us denote β as the number of neurons in $\mathbf{h}_1$, and h_i is bounded in $[-1, 1]$, the global sensitivity $\Delta_{\mathcal{R}}$ is as follows:

Lemma 3.9. *The global sensitivity of* $\widetilde{\mathcal{R}}$ *over any two neighboring batches,* B_t *and* B'_t*, is:* $\Delta_{\mathcal{R}} \leq d(\beta + 2)$.

Proof of Lemma 3.9

Proof. Assume that B_t and B'_t differ in the last tuple, x_m (x'_m). Then,

$$\begin{aligned}
\Delta_{\mathcal{R}} &= \sum_{j=1}^{d} \Big[\| \sum_{x_i \in B_t} \frac{1}{2} h_i - \sum_{x'_i \in B'_t} \frac{1}{2} h'_i \|_1 + \| \sum_{x_i \in B_t} x_{ij} - \sum_{x'_i \in B'_t} x'_{ij} \|_1 \Big] \\
&\leq 2 \max_{x_i} \sum_{j=1}^{d} (\| \frac{1}{2} h_i \|_1 + \| x_{ij} \|_1) \\
&\leq d(\beta + 2)
\end{aligned}$$

□

By setting $\Delta_{\mathcal{R}} = d(\beta + 2)$, we show that the output of $a(\cdot)$, which is the perturbed affine transformation $\overline{\mathbf{h}}_{1\overline{B}_t} = \{\theta_1^T \overline{x}_i + \frac{2}{m}Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})\}_{\overline{x}_i \in \overline{B}_t}$, is (ϵ_1/γ)-DP, given $\gamma = \frac{2\Delta_{\mathcal{R}}}{m\|\overline{\theta}_1\|_{1,1}}$ and $\|\overline{\theta}_1\|_{1,1}$ is the maximum 1-norm of θ_1's columns [186]. This is important to tighten the privacy budget consumption in computing the remaining hidden layers $g(a(x, \theta_1), \theta_2)$.

In fact, without using additional information from the original data, the computation of $g(a(x, \theta_1), \theta_2)$ is also (ϵ_1/γ)-DP.

Similarly, the perturbation of each benign example x turns $\overline{B}_t = \{\overline{x}_i \leftarrow x_i + \frac{1}{m}Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})\}_{x_i \in B_t}$ into a $(\epsilon_1/\gamma_{\mathbf{x}})$-DP batch, with $\gamma_{\mathbf{x}} = \Delta_{\mathcal{R}}/m$. We do not use the post-processing property of DP to estimate the DP guarantee of $\overline{\mathbf{h}}_{1\overline{B}_t}$ based upon the DP guarantee of $\overline{B}_t$, since $\epsilon_1/\gamma < \epsilon_1/\gamma_{\mathbf{x}}$ in practice. So, the (ϵ_1/γ)-DP $\overline{\mathbf{h}}_{1\overline{B}_t}$ provides a more rigorous DP protection to the computation of $g(\cdot)$ and to the output layer.

Lemma 3.10. *The computation of the batch $\overline{B}_t$ as the input layer is $(\epsilon_1/\gamma_{\mathbf{x}})$-DP, and the computation of the affine transformation $\overline{\mathbf{h}}_{1\overline{B}_t}$ is (ϵ_1/γ)-DP.*

Proof of Lemma 3.10

Proof. Regarding the computation of $\mathbf{h}_{1\overline{B}_t} = \{\overline{\theta}_1^T \overline{x}_i\}_{\overline{x}_i \in \overline{B}_t}$, we can see that $h_i = \overline{\theta}_1^T \overline{x}_i$ is a linear function of x. The sensitivity of a function h is defined as the maximum change in output, that can be generated by a change in the input [101]. Therefore, the global sensitivity of $\mathbf{h}_1$ can be computed as follows:

$$\Delta_{\mathbf{h}_1} = \frac{\|\sum_{\overline{x}_i \in \overline{B}_t} \overline{\theta}_1^T \overline{x}_i - \sum_{\overline{x}'_i \in \overline{B}'_t} \overline{\theta}_1^T \overline{x}'_i\|_1}{\|\sum_{\overline{x}_i \in \overline{B}_t} \overline{x}_i - \sum_{\overline{x}'_i \in \overline{B}'_t} \overline{x}'_i\|_1} \leq \max_{x_i \in B_t} \frac{\|\overline{\theta}_1^T \overline{x}_i\|_1}{\|\overline{x}_i\|_1} \leq \|\overline{\theta}_1^T\|_{1,1}$$

following matrix norms [186]: $\|\overline{\theta}_1^T\|_{1,1}$ is the maximum 1-norm of θ_1's columns. By injecting Laplace noise $Lap(\frac{\Delta_{\mathbf{h}_1}}{\epsilon_1})$ into $\mathbf{h}_{1B_t}$, i.e., $\overline{\mathbf{h}}_{1\overline{B}_t} = \{\overline{\theta}_1^T \overline{x}_i + Lap(\frac{\Delta_{\mathbf{h}_1}}{\epsilon_1})\}_{\overline{x}_i \in \overline{B}_t}$, we can preserve ϵ_1-DP in the computation of $\overline{\mathbf{h}}_{1\overline{B}_t}$. Let us set $\Delta_{\mathbf{h}_1} = \|\overline{\theta}_1^T\|_{1,1}$, $\gamma = \frac{2\Delta_{\mathcal{R}}}{m\Delta_{\mathbf{h}_1}}$, and χ_2 drawn as a Laplace noise $[Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})]^\beta$, in our mechanism, the perturbed affine transformation $\overline{\mathbf{h}}_{1\overline{B}_t}$ is presented as:

$$\overline{\mathbf{h}}_{1\overline{B}_t} = \{\overline{\theta}_1^T \overline{x}_i + \frac{2\chi_2}{m}\}_{\overline{x}_i \in \overline{B}_t} = \{\overline{\theta}_1^T \overline{x}_i + \frac{2}{m}[Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})]^{\beta}\}_{\overline{x}_i \in \overline{B}_t}$$
$$= \{\overline{\theta}_1^T \overline{x}_i + [Lap(\frac{\gamma\Delta_{\mathbf{h}_1}}{\epsilon_1})]^{\beta}\}_{\overline{x}_i \in \overline{B}_t} = \{\overline{\theta}_1^T \overline{x}_i + [Lap(\frac{\Delta_{\mathbf{h}_1}}{\epsilon_1/\gamma})]^{\beta}\}_{\overline{x}_i \in \overline{B}_t}$$

This results in an (ϵ_1/γ)-DP affine transformation $\overline{\mathbf{h}}_{1B_t} = \{\overline{\theta}_1^T \overline{x}_i + [Lap(\frac{\Delta_{\mathbf{h}_1}}{\epsilon_1/\gamma})]^{\beta}\}_{\overline{x}_i \in \overline{B}_t}$.

Similarly, the perturbed inputs $\overline{B}_t = \{\overline{x}_i\}_{\overline{x}_i \in \overline{B}_t} = \{x_i + \frac{\chi_1}{m}\}_{x_i \in B_t} = \{x_i + [Lap(\frac{\Delta_{\mathbf{x}}}{\epsilon_1/\gamma_{\mathbf{x}}})]^d\}_{x_i \in B_t}$, where $\Delta_{\mathbf{x}}$ is the sensitivity measuring the maximum change in the input layer that can be generated by a change in the batch B_t and $\gamma_{\mathbf{x}} = \frac{\Delta_{\mathcal{R}}}{m\Delta_{\mathbf{x}}}$. Following [101], $\Delta_{\mathbf{x}}$ can be computed as follows: $\Delta_{\mathbf{x}} = \frac{\|\sum_{x_i \in B_t} \overline{x}_i - \sum_{x'_i \in B'_t} \overline{x}'_i\|_1}{\|\sum_{x_i \in B_t} \overline{x}_i - \sum_{x'_i \in B'_t} \overline{x}'_i\|_1} = 1$. As a result, the computation of $\overline{B}_t$ is $(\epsilon_1/\gamma_{\mathbf{x}})$-DP. Consequently, Lemma 3.10 does hold. □

Departing from the vanilla Functional Mechanism, in which only *grid search*-based approaches can be applied to find DP-preserving θ_1 with a low loss $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$, our following Theorem 3.2 shows that *gradient descent*-based optimizing $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ is $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP in learning θ_1 given an $(\epsilon_1/\gamma_{\mathbf{x}})$-DP $\overline{B}_t$ batch. In fact, in addition to $h_i, \overline{h}_i, \widetilde{x}_i$, based on Lemma 3.10, we further show that the computation of gradients, i.e., $\forall j \in [1, d] : \frac{\delta \overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)}{\delta \theta_{1j}} = \sum_{i=1}^m \overline{h}_i(\frac{1}{2} - \overline{x}_{ij})$, and descent operations given the $(\epsilon_1/\gamma_{\mathbf{x}})$-DP $\overline{B}_t$ batch are $(\epsilon_1/\gamma_{\mathbf{x}})$-DP, without incurring any additional information from the original data. As a result, gradient descent-based approaches can be applied to optimize $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ in Algorithm 3.3, since all the computations on top of $\overline{B}_t$ are DP, without using any additional information from the original data.

Theorem 3.2. *The gradient descent-based optimization of* $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ *preserves* $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-*DP in learning* θ_1.

Proof of Theorem 3.2

Proof. Given χ_1 drawn as a Laplace noise $[Lap(\frac{\Delta_\mathcal{R}}{\epsilon_1})]^d$ and χ_2 drawn as a Laplace noise $[Lap(\frac{\Delta_\mathcal{R}}{\epsilon_1})]^\beta$, the perturbation of the coefficient $\phi \in \Phi = \{\frac{1}{2}h_i, x_i\}$, denoted as $\overline{\phi}$, can be rewritten as follows:

$$\begin{aligned} \mathit{for}\ \phi \in \{x_i\} : \overline{\phi} &= \sum_{x_i \in B} (\phi_{x_i} + \frac{\chi_1}{m}) = \sum_{x_i \in B} \phi_{x_i} + \chi_1 \\ &= \sum_{x_i \in B} \phi_{x_i} + [Lap(\frac{\Delta_\mathcal{R}}{\epsilon_1})]^d \end{aligned} \tag{3.41}$$

$$\begin{aligned} \mathit{for}\ \phi \in \{\frac{1}{2}h_i\} : \overline{\phi} &= \sum_{x_i \in B} \frac{1}{2}(h_i + \frac{2\chi_2}{m}) = \sum_{x_i \in B} (\phi_{x_i} + \frac{\chi_2}{m}) \\ &= \sum_{x_i \in B} \phi_{x_i} + \chi_2 = \sum_{x_i \in B} \phi_{x_i} + [Lap(\frac{\Delta_\mathcal{R}}{\epsilon_1})]^\beta \end{aligned} \tag{3.42}$$

we have

$$Pr\big(\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)\big) = \prod_{j=1}^{d} \prod_{\phi \in \Phi} \exp\big(- \frac{\epsilon_1 \|\sum_{x_i \in B_t} \phi_{x_i} - \overline{\phi}\|_1}{\Delta_\mathcal{R}}\big)$$

$\Delta_\mathcal{R}$ is set to $d(\beta + 2)$, we have that:

$$\begin{aligned} \frac{Pr\big(\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{B}'_t}(\theta_1)\big)} &= \frac{\prod_{j=1}^{d} \prod_{\phi \in \Phi} \exp\big(- \frac{\epsilon_1 \|\sum_{x_i \in B_t} \phi_{x_i} - \overline{\phi}\|_1}{\Delta_\mathcal{R}}\big)}{\prod_{j=1}^{d} \prod_{\phi \in \Phi} \exp\big(- \frac{\epsilon_1 \|\sum_{x'_i \in B'_t} \phi_{x'_i} - \overline{\phi}\|_1}{\Delta_\mathcal{R}}\big)} \\ &\leq \prod_{j=1}^{d} \prod_{\phi \in \Phi} \exp(\frac{\epsilon_1}{\Delta_\mathcal{R}} \Big\| \sum_{x_i \in B_t} \phi_{x_i} - \sum_{x'_i \in B'_t} \phi_{x'_i} \Big\|_1) \\ &\leq \prod_{j=1}^{d} \prod_{\phi \in \Phi} \exp(\frac{\epsilon_1}{\Delta_\mathcal{R}} 2 \max_{x_i \in B_t} \big\|\phi_{x_i}\big\|_1) \leq \exp(\frac{\epsilon_1 d(\beta + 2)}{\Delta_\mathcal{R}}) = \exp(\epsilon_1) \end{aligned} \tag{3.43}$$

Consequently, the computation of $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ preserves ϵ_1-DP in Algorithm 3.3 (**Result 1**). To show that gradient descent-based optimizers can be used to optimize the objective function $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ in learning private parameters θ_1, we prove that all the computations on top of the perturbed data $\overline{B}_t$, including h_i, $\overline{h}_i$, $\widetilde{x}_i$, gradients and descent, are DP without incurring any additional information from the original data, as follows.

First, by following the post-processing property in DP [100], it is clear that the computations of $\mathbf{h}_{1\overline{B}_t} = \{h_i\}_{\overline{x}_i \in \overline{B}_t} = \theta_1^T\{\overline{x}_i\}_{\overline{x}_i \in \overline{B}_t}$ is $(\epsilon_1/\gamma_{\mathbf{x}})$-DP. As in Lemma 3.10, we also have that $\overline{\mathbf{h}}_{1\overline{B}_t} = \{h_i + \frac{2\chi_2}{m}\}_{\overline{x}_i \in \overline{B}_t}$ is (ϵ_1/γ)-DP. Given this, it is obvious that $\widetilde{\mathbf{x}}_i = \{\widetilde{x}_i\}_{\overline{x}_i \in \overline{B}_t} = \theta_1\{\overline{h}_i\}_{\overline{x}_i \in \overline{B}_t}$ is (ϵ_1/γ)-DP, i.e., the post-processing property in DP. In addition, the computations of $\mathbf{h}_{1\overline{B}_t}$, $\overline{\mathbf{h}}_{1\overline{B}_t}$, and $\widetilde{\mathbf{x}}_i$ do not access the original data B_t. Therefore, they do not incur any additional information from the private data, except the privacy loss measured by $(\epsilon_1/\gamma_{\mathbf{x}})$-DP, since the computations of $\overline{\mathbf{h}}_{1\overline{B}_t}$ and $\widetilde{\mathbf{x}}_i$ are based on the $(\epsilon_1/\gamma_{\mathbf{x}})$-DP $\mathbf{h}_{1\overline{B}_t}$. (**Result 2**)

Second, the gradient of a particular parameter θ_{1j}, with $\forall j \in [1, d]$, can be computed as follows:

$$\forall j \in [1, d] : \nabla_{\theta_{1j}} \overline{\mathcal{R}}_{\overline{B}_t}(\theta_1) = \frac{\delta \overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)}{\delta \theta_{1j}} = \sum_{i=1}^{m} \overline{h}_i(\frac{1}{2} - \overline{x}_{ij}) \tag{3.44}$$

$$= \sum_{i=1}^{m}(h_i + \frac{2\chi_2}{m})(\frac{1}{2} - \overline{x}_{ij}) \tag{3.45}$$

$$= \Big[\sum_{i=1}^{m} h_i(\frac{1}{2} - \overline{x}_{ij})\Big] + \chi_2 - \Big[\frac{2\chi_2}{m}\sum_{i=1}^{m} \overline{x}_{ij}\Big] \tag{3.46}$$

In Equation (3.46), we have that $\sum_{i=1}^{m} \overline{x}_{ij} = (\sum_{i=1}^{m} x_{ij}) + Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$ (Equation (3.41)), which is $(\epsilon_1/\gamma_{\mathbf{x}})$-DP. Therefore, the term $\frac{2\chi_2}{m}\sum_{i=1}^{m} \overline{x}_{ij}$ also is $(\epsilon_1/\gamma_{\mathbf{x}})$-DP (the post-processing property in DP). (**Result 3**)

Regarding the term $\sum_{i=1}^{m} h_i(\frac{1}{2} - \overline{x}_{ij})$ in Equation (3.46), its global sensitivity given two arbitrary neighboring batches, denoted as Δ_g, can be bounded as follows: $\Delta_g \leq 2\max_{\overline{x}_i} \|h_i(\frac{1}{2} - \overline{x}_{ij})\|_1 = 3\beta$. As a result, we have that:

$$\big[\sum_{i=1}^{m} h_i(\frac{1}{2} - \overline{x}_{ij})\big] + \chi_2 = \big[\sum_{i=1}^{m} h_i(\frac{1}{2} - \overline{x}_{ij})\big] + [Lap(\frac{\Delta_g}{\epsilon_1/\frac{\Delta_{\mathcal{R}}}{\Delta_g}})]^{\beta} \tag{3.47}$$

which is $(\epsilon_1/\frac{\Delta_{\mathcal{R}}}{\Delta_g})$-DP. (**Result 4**)

From Results 3 and 4, the computation of gradients $\nabla_{\theta_{1j}} \overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ is $(\epsilon_1/\frac{\Delta_{\mathcal{R}}}{\Delta_g} + \epsilon_1/\gamma_{\mathbf{x}})$-DP, since: **(1)** The computations of the two terms in Equation (3.46) can be treated as two independent DP-preserving mechanisms applied on the perturbed batch $\overline{B}_t$; and **(2)** This is true for every dimension $j \in [1, d]$, each of which $\nabla_{\theta_{1j}}$ is independently computed and bounded. It is important to note that this result is different from the traditional DPSGD [62], in which the parameter gradients are jointly clipped by a l_2-norm constant bound, such that Gaussian noise can be injected to achieve DP. In addition, as in Equation (3.44), the computation of $\nabla_{\theta_{1j}} \overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ only uses $(\epsilon_1/\gamma_{\mathbf{x}})$-DP $\overline{B}_t = \{\overline{x}_i\}_{\overline{x}_i \in \overline{B}_t}$ and (ϵ_1/γ)-DP $\overline{\mathbf{h}}_{1\overline{B}_t}$, without accessing the original data. Basically, $\overline{\mathbf{h}}_{1\overline{B}_t}$ is computed on top of $\overline{B}_t$, without touching any benign example. Therefore, it does not incur any additional information from the private data, except the privacy loss $(\epsilon_1/\frac{\Delta_{\mathcal{R}}}{\Delta_g} + \epsilon_1/\gamma_{\mathbf{x}})$-DP. In practice, we observed that $\epsilon_1/\gamma_{\mathbf{x}} \gg \epsilon_1/\frac{\Delta_{\mathcal{R}}}{\Delta_g} \approxeq \epsilon_1 \times 1e-3$, which is tiny. We can simply consider that the computation of gradients $\nabla_{\theta_{1j}} \overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ is $(\epsilon_1/\gamma_{\mathbf{x}})$-DP without affecting the general DP protection. In addition to the gradient computation, the descent operations are simply post-processing steps without consuming any further privacy budget. (**Result 5**)

From Results 1, 2, and 5, we have shown that all the computations on top of $(\epsilon_1/\gamma_{\mathbf{x}})$-DP $\overline{B}_t$, including parameter gradients and gradient descents, clearly are DP without accessing the original data; therefore, they do not incur any additional information from the private data (the post-processing property in DP). As a result, gradient descent-based approaches can be applied to optimize $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ in Algorithm 3.3. The total privacy budget to learn the perturbed optimal parameters $\overline{\theta}_1$ in Algorithm 3.3 is $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP, where the $\epsilon_1/\gamma_{\mathbf{x}}$ is counted for the perturbation on the batch of benign examples B_t.

Consequently, Theorem 3.2 does hold.

□

3.2.5 Adversarial Learning with Differential Privacy

To integrate adversarial learning, we first draft DP adversarial examples $\overline{x}_j^{\text{adv}}$ using perturbed benign examples $\overline{x}_j$, with an ensemble of attack algorithms A and a random perturbation budget $\mu_t \in (0, 1]$, at each step t (Lines 6-11, Algorithm 3.3). This will significantly enhances the robustness of our models under different types of adversarial examples with an unknown adversarial attack size μ.

$$\overline{x}_j^{\text{adv}} = \overline{x}_j + \mu \cdot \text{sign}\Big(\nabla_{\overline{x}_j}\mathcal{L}\big(f(\overline{x}_j, \theta), y(\overline{x}_j)\big)\Big) \tag{3.48}$$

with $y(\overline{x}_j)$ is the class prediction result of $f(\overline{x}_j)$ to avoid label leaking of x_j during the adversarial example crafting. Given a set of DP adversarial examples $\overline{B}_t^{\text{adv}}$, training the auto-encoder with $\overline{B}_t^{\text{adv}}$ preserves $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP.

Theorem 3.3. *The gradient descent-based optimization of $\overline{\mathcal{R}}_{\overline{B}_t^{adv}}(\theta_1)$ preserves $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP in learning θ_1.*

The proof of Theorem 3.3 is in Result 4 of the proof in Page 117. It can be extended to iterative attacks as: $\overline{x}_{j,0}^{\text{adv}} = \overline{x}_j$,

$$\overline{x}_{j,\text{t}+1}^{\text{adv}} = \overline{x}_{j,\text{t}}^{\text{adv}} + \frac{\mu}{T_\mu} \cdot \text{sign}\Big(\nabla_{\overline{x}_{j,\text{t}}^{\text{adv}}}\mathcal{L}\big(f(\overline{x}_{j,\text{t}}^{\text{adv}}, \theta), y(\overline{x}_{j,\text{t}}^{\text{adv}})\big)\Big) \tag{3.49}$$

where $y(\overline{x}_{j,\text{t}}^{\text{adv}})$ is the prediction of $f(\overline{x}_{j,\text{t}}^{\text{adv}}, \theta)$, t $\in [0, T_\mu - 1]$.

Second, we propose a novel DP adversarial objective function $L_{B_t}(\theta_2)$, in which the loss function $\mathcal{L}$ for benign examples is combined with an additional loss function Υ for DP adversarial examples, to optimize the parameters θ_2. The objective function $L_{B_t}(\theta_2)$ is defined as follows:

$$L_{\overline{B}_t \cup \overline{B}_t^{\text{adv}}}(\theta_2) = \frac{1}{m(1+\xi)}\Big(\sum_{\overline{x}_i \in \overline{B}_t} \mathcal{L}\big(f(\overline{x}_i, \theta_2), y_i\big) + \xi \sum_{\overline{x}_j^{\text{adv}} \in \overline{B}_t^{\text{adv}}} \Upsilon\big(f(\overline{x}_j^{\text{adv}}, \theta_2), y_j\big)\Big) \quad (3.50)$$

where ξ is a hyper-parameter. For the sake of clarity, in Equation (3.50), we denote y_i and y_j as the true class labels y_{x_i} and y_{x_j} of examples x_i and x_j. $\overline{x}_j^{\text{adv}}$ and x_j share the same label y_{x_j}.

Now we are ready to preserve DP in objective functions $\mathcal{L}\big(f(\overline{x}_i, \theta_2), y_i\big)$ and $\Upsilon\big(f(\overline{x}_j^{\text{adv}}, \theta_2), y_j\big)$ in order to achieve DP in learning θ_2. Since the objective functions use the true class labels y_i and y_j, we need to protect the labels at the output layer. Let us first present our approach to preserve DP in the objective function $\mathcal{L}$ for benign examples. Given $\mathbf{h}_{\pi i}$ computed from the $\overline{x}_i$ through the network with W_π is the parameter at the last hidden layer $\mathbf{h}_\pi$. Cross-entropy function is approximated as: $\mathcal{L}_{\overline{B}_t}\big(\theta_2\big) \approxeq \sum_{k=1}^{K} \sum_{\overline{x}_i} \big[\mathbf{h}_{\pi i} W_{\pi k} - (\mathbf{h}_{\pi i} W_{\pi k}) y_{ik} - \frac{1}{2}|\mathbf{h}_{\pi i} W_{\pi k}| + \frac{1}{8}(\mathbf{h}_{\pi i} W_{\pi k})^2\big] \approxeq \mathcal{L}_{1\overline{B}_t}\big(\theta_2\big) - \mathcal{L}_{2\overline{B}_t}\big(\theta_2\big)$, where $\mathcal{L}_{1\overline{B}_t}\big(\theta_2\big) = \sum_{k=1}^{K} \sum_{\overline{x}_i} \big[\mathbf{h}_{\pi i} W_{\pi k} - \frac{1}{2}|\mathbf{h}_{\pi i} W_{\pi k}| + \frac{1}{8}(\mathbf{h}_{\pi i} W_{\pi k})^2\big]$, and $\mathcal{L}_{2\overline{B}_t}\big(\theta_2\big) = \sum_{k=1}^{K} \sum_{\overline{x}_i} (\mathbf{h}_{\pi i} y_{ik}) W_{\pi k}$.

Based on the *post-processing property of DP* [100], $\mathbf{h}_{\pi \overline{B}_t} = \{\mathbf{h}_{\pi i}\}_{\overline{x}_i \in \overline{B}_t}$ is (ϵ_1/γ)-DP, since the computation of $\overline{\mathbf{h}}_{1\overline{B}_t}$ is (ϵ_1/γ)-DP (Lemma 3.10). Hence, the optimization of $\mathcal{L}_{1\overline{B}_t}\big(\theta_2\big)$ does not disclose any information from the training data, and $\frac{Pr(\mathcal{L}_{1\overline{B}_t}(\theta_2))}{Pr(\mathcal{L}_{1\overline{B}'_t}(\theta_2))} = \frac{Pr(\mathbf{h}_{\pi \overline{B}_t})}{Pr(\mathbf{h}_{\pi \overline{B}'_t})} \leq e^{\epsilon_1/\gamma}$, given neighboring batches $\overline{B}_t$ and $\overline{B}'_t$. Thus, we only need to preserve ϵ_2-

DP in the function $\mathcal{L}_{2\overline{B}_t}(\theta_2)$, which accesses the ground-truth label y_{ik}. Given coefficients $\mathbf{h}_{\pi i} y_{ik}$, the sensitivity $\Delta_{\mathcal{L}2}$ of $\mathcal{L}_{2\overline{B}_t}(\theta_2)$ is computed as:

Lemma 3.11. *Let $\overline{B}_t$ and $\overline{B}'_t$ be neighboring batches of benign examples, we have the following inequality: $\Delta_{\mathcal{L}2} \leq 2|\mathbf{h}_\pi|$, where $|\mathbf{h}_\pi|$ is the number of hidden neurons in $\mathbf{h}_\pi$.*

Proof of Lemma 3.11

Proof. Assume that $\overline{B}_t$ and $\overline{B}'_t$ differ in the last tuple, and $\overline{x}_m$ ($\overline{x}'_m$) be the last tuple in $\overline{B}_t$ ($\overline{B}'_t$), we have that

$$\Delta_{\mathcal{L}2} = \sum_{k=1}^{K} \Big\| \sum_{\overline{x}_i \in \overline{B}_t} (\mathbf{h}_{\pi i} y_{ik}) - \sum_{\overline{x}'_i \in \overline{B}'_t} (\mathbf{h}'_{\pi i} y'_{ik}) \Big\|_1 = \sum_{k=1}^{K} \big\| \mathbf{h}_{\pi m} y_{mk} - \mathbf{h}'_{\pi m} y'_{mk} \big\|_1$$

Since y_{mk} and y'_{mk} are one-hot encoding, we have that $\Delta_{\mathcal{L}2} \leq 2 \max_{\overline{x}_i} \|\mathbf{h}_{\pi i}\|_1$. Given $\mathbf{h}_{\pi i} \in [-1, 1]$, we have

$$\Delta_{\mathcal{L}2} \leq 2|\mathbf{h}_\pi| \tag{3.51}$$

Lemma 3.11 does hold. □

The sensitivity of our objective function is notably smaller than the state-of-the-art bound [72], which is crucial to improve our model utility. The perturbed functions become: $\overline{\mathcal{L}}_{\overline{B}_t}\big(\theta_2\big) = \mathcal{L}_{1\overline{B}_t}(\theta_2) - \overline{\mathcal{L}}_{2\overline{B}_t}(\theta_2)$, where $\overline{\mathcal{L}}_{2\overline{B}_t}(\theta_2) = \sum_{k=1}^{K} \sum_{\overline{x}_i} \big(\mathbf{h}_{\pi i} y_{ik} + \frac{1}{m} Lap(\frac{\Delta_{\mathcal{L}2}}{\epsilon_2})\big) W_{\pi k}$.

Theorem 3.4. *Algorithm 3.3 preserves $(\epsilon_1/\gamma + \epsilon_2)$-DP in the gradient descent-based optimization of $\overline{\mathcal{L}}_{\overline{B}}\big(\theta_2\big)$.*

Proof of Theorem 3.4

Proof. Let $\overline{B}_t$ and $\overline{B}'_t$ be neighboring batches of benign examples, and χ_3 drawn as Laplace noise $[Lap(\frac{\Delta_{\mathcal{L}2}}{\epsilon_2})]^{|\mathbf{h}_\pi|}$, the perturbations of the coefficients $\mathbf{h}_{\pi i} y_{ik}$ can be rewritten as:

$$\overline{\mathbf{h}}_{\pi i}\overline{y}_{ik} = \sum_{\overline{x}_i}(\mathbf{h}_{\pi i} y_{ik} + \frac{\chi_3}{m}) = \sum_{\overline{x}_i}(\mathbf{h}_{\pi i} y_{ik}) + [Lap(\frac{\Delta_{\mathcal{L}2}}{\epsilon_2})]^{|\mathbf{h}_\pi|}$$

Since all the coefficients are perturbed, and given $\Delta_{\mathcal{L}2} = 2|\mathbf{h}_\pi|$, we have that

$$\begin{aligned}
\frac{Pr(\mathcal{L}_{\overline{B}_t}(\theta_2))}{Pr(\mathcal{L}_{\overline{B}'_t}(\theta_2))} &= \frac{Pr(\mathcal{L}_{1\overline{B}_t}(\theta_2))}{Pr(\mathcal{L}_{1\overline{B}'_t}(\theta_2))} \times \frac{Pr(\overline{\mathcal{L}}_{2\overline{B}_t}(\theta_2))}{Pr(\overline{\mathcal{L}}_{2\overline{B}'_t}(\theta_2))} \\
&\leq e^{\epsilon_1/\gamma} \sum_{k=1}^{K} \frac{\exp(-\frac{\epsilon_2 \|\sum_{\overline{x}_i} \mathbf{h}_{\pi i} y_{ik} - \overline{\mathbf{h}}_{\pi i}\overline{y}_{ik}\|_1}{\Delta_{\mathcal{L}2}})}{\exp(-\frac{\epsilon_2 \|\sum_{\overline{x}'_i} \mathbf{h}_{\pi i} y_{ik} - \overline{\mathbf{h}}_{\pi i}\overline{y}_{ik}\|_1}{\Delta_{\mathcal{L}2}})} \\
&\leq e^{\epsilon_1/\gamma} \sum_{k=1}^{K} \exp(\frac{\epsilon_2}{\Delta_{\mathcal{L}2}} \Big\| \sum_{\overline{x}_i} \mathbf{h}_{\pi i} y_{ik} - \sum_{\overline{x}'_i} \mathbf{h}_{\pi i} y_{ik} \Big\|_1) \\
&\leq e^{\epsilon_1/\gamma} \exp(\frac{\epsilon_2}{\Delta_{\mathcal{L}2}} 2 \max_{\overline{x}_i} \|\mathbf{h}_{\pi i}\|_1) = e^{\epsilon_1/\gamma + \epsilon_2}
\end{aligned}$$

The computation of $\overline{\mathcal{L}}_{2\overline{B}_t}(\theta_2)$ preserves $(\epsilon_1/\gamma + \epsilon_2)$-differential privacy. Similar to Theorem 3.2, the gradient descent-based optimization of $\overline{\mathcal{L}}_{2\overline{B}_t}(\theta_2)$ does not access additional information from the original input $x_i \in B_t$. It only reads the (ϵ_1/γ)-DP $\overline{\mathbf{h}}_{1\overline{B}_t} = \{h_i + \frac{2\chi_2}{m}\}_{\overline{x}_i \in \overline{B}_t}$. Consequently, the optimal perturbed parameters $\overline{\theta}_2$ derived from $\overline{\mathcal{L}}_{2\overline{B}_t}(\theta_2)$ are $(\epsilon_1/\gamma + \epsilon_2)$-DP. □

We apply the same technique to preserve $(\epsilon_1/\gamma + \epsilon_2)$-DP in the optimization of the function $\Upsilon\big(f(\overline{x}_j^{\text{adv}}, \theta_2), y_j\big)$ over the DP adversarial examples $\overline{x}_j^{\text{adv}} \in \overline{B}_t^{\text{adv}}$. As the perturbed functions $\overline{\mathcal{L}}$ and $\overline{\Upsilon}$ are always optimized given two disjoint batches $\overline{B}_t$ and $\overline{B}_t^{\text{adv}}$, the privacy budget used to preserve DP in the adversarial objective function $L_{B_t}(\theta_2)$ is $(\epsilon_1/\gamma + \epsilon_2)$, following the *parallel composition* property [100]. The total budget to learn private parameters $\overline{\theta} = \{\overline{\theta}_1, \overline{\theta}_2\} = \arg\min_{\{\theta_1,\theta_2\}}(\overline{\mathcal{R}}_{\overline{B}_t \cup \overline{B}_t^{\text{adv}}}(\theta_1) + \overline{L}_{\overline{B}_t \cup \overline{B}_t^{\text{adv}}}(\theta_2))$ is $\epsilon = (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$ (Line 12, Algorithm 3.3).

DP at the Dataset Level Our mechanism achieves DP at the batch level $\overline{B}_t \cup \overline{B}_t^{\text{adv}}$ given a specific training step t. By constructing *disjoint* and *fixed* batches from D, we leverage both parallel composition and post-processing properties of DP to extend the result to ϵ-DP in learning $\{\overline{\theta}_1, \overline{\theta}_2\}$ on D across T training steps. There are three key properties in our model: **(1)** It only reads perturbed inputs $\overline{B}_t$ and perturbed coefficients $\overline{\mathbf{h}}_1$, which are DP across T training steps with *a single draw of Laplace noise* (i.e., no further privacy leakage); **(2)** Given N/m disjoint batches in each epoch, $\forall \overline{x}$, $\overline{x}$ is included in *one and only one* batch, denoted $B_x \in \overline{\mathbf{B}}$. As a result, the DP guarantee to $\overline{x}$ in D is equivalent to the DP guarantee to $\overline{x}$ in B_x; since the optimization using any other batches does not affect the DP guarantee of $\overline{x}$, even the objective function given B_x can be slightly different from the objective function given any other batches in $\overline{\mathbf{B}}$; and **(3)** All the batches are fixed across T training steps to prevent additional privacy leakage, caused by generating new and overlapping batches (which are considered overlapping datasets in the parlance of DP) in the typical training.

Theorem 3.5. *Algorithm 3.3 achieves $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$-DP parameters $\overline{\theta} = \{\overline{\theta}_1, \overline{\theta}_2\}$ on the private training data D across T gradient descent-based training steps.*

Proofs of Theorem 3.3 and Theorem 3.5

Proof. First, we optimize for a single draw of noise during training (Line 3, Algorithm 3.3) and all the batches of perturbed benign examples are disjoint and fixed across epochs. As a result, the computation of $\overline{x}_i$ is equivalent to a data preprocessing step with DP, which does not incur any additional privacy budget consumption over T training steps (the post-processing property of DP) (**Result 1**). That is different from repeatedly applying a DP mechanism on either the same or overlapping datasets causing the accumulation of the privacy budget.

Now, we show that our algorithm achieves DP at the dataset level D. Let us consider the computation of the first hidden layer, given any two neighboring datasets D and D' differing at most one tuple $\overline{x}_e \in D$ and $\overline{x}'_e \in D'$. For any $O = \prod_{i=1}^{N/m} o_i \in \prod_{i=1}^{N/m} \overline{\mathbf{h}}_{1\overline{B}_i} (\in \mathbb{R}^{\beta \times m})$, we have that

$$\frac{P\big(\overline{\mathbf{h}}_{1D} = O\big)}{P\big(\overline{\mathbf{h}}_{1D'} = O\big)} = \frac{P(\overline{\mathbf{h}}_{1\overline{B}_1} = o_1) \dots P(\overline{\mathbf{h}}_{1\overline{B}_{N/m}} = o_{N/m})}{P(\overline{\mathbf{h}}_{1\overline{B}'_1} = o_1) \dots P(\overline{\mathbf{h}}_{1\overline{B}'_{N/m}} = o_{N/m})} \tag{3.52}$$

By having disjoint and fixed batches, we have that:

$$\exists! \tilde{B} \in \overline{\mathbf{B}} \; s.t. \; x_e \in \tilde{B} \; and \; \exists! \tilde{B}' \in \overline{\mathbf{B}}' \; s.t. \; x'_e \in \tilde{B}' \tag{3.53}$$

From Equations (3.52), (3.53), and Lemma 3.10, we have that

$$\forall \overline{B} \in \mathbf{B}, \overline{B} \neq \tilde{B} : \overline{B} = \overline{B}' \Rightarrow \frac{P\big(\overline{\mathbf{h}}_{1\overline{B}} = o\big)}{P\big(\overline{\mathbf{h}}_{1\overline{B}'} = o\big)} = 1 \tag{3.54}$$

$$\textit{Equations (3.53) and (3.54)} \Rightarrow \frac{P\big(\overline{\mathbf{h}}_{1D} = O\big)}{P\big(\overline{\mathbf{h}}_{1D'} = O\big)} = \frac{P\big(\overline{\mathbf{h}}_{1\tilde{B}} = \tilde{o}\big)}{P\big(\overline{\mathbf{h}}_{1\tilde{B}'} = \tilde{o}\big)} \leq e^{\epsilon_1/\gamma} \tag{3.55}$$

As a result, the computation of $\overline{\mathbf{h}}_{1D}$ is (ϵ_1/γ)-DP given the data D, since the Equation (3.55) does hold for any tuple $x_e \in D$. That is consistent with the parallel composition property of DP, in which batches can be considered disjoint datasets given $\overline{\mathbf{h}}_{1\overline{B}}$ as a DP mechanism [100].

This does hold across epochs, since batches $\overline{\mathbf{B}}$ are disjoint and fixed among epochs. At each training step $t \in [1, T]$, the computation of $\overline{\mathbf{h}}_{1\overline{B}_t}$ does not access the original data. It only reads the perturbed batch of inputs $\overline{B}_t$, which is $(\epsilon_1/\gamma_{\mathbf{x}})$-DP (Lemma 3.10).

Following the post-processing property in DP [100], the computation of $\overline{\mathbf{h}}_{1\overline{B}_t}$ does not incur any additional information from the original data across T training steps. (**Result 2**)

Similarly, we show that the optimization of the function $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ is $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP across T training steps. As in Theorem 3.2 and Proof 3.2.4, we have that $Pr\big(\overline{\mathcal{R}}_{\overline{B}}(\theta_1)\big) = \prod_{j=1}^{d}\prod_{\phi\in\Phi}\exp\big(-\frac{\epsilon_1\|\sum_{x_i\in B}\phi_{x_i}-\overline{\phi}\|_1}{\Delta_{\mathcal{R}}}\big)$, where $\overline{B}\in\overline{\mathbf{B}}$. Given any two perturbed neighboring datasets $\overline{D}$ and $\overline{D}'$ differing at most one tuple $\overline{x}_e \in \overline{D}$ and $\overline{x}'_e \in \overline{D}'$:

$$\frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}}(\theta_1)\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'}(\theta_1)\big)} = \frac{Pr\big(\overline{\mathcal{R}}_{\overline{B}_1}(\theta_1)\big)\ldots Pr\big(\overline{\mathcal{R}}_{\overline{B}_{N/m}}(\theta_1)\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{B}'_1}(\theta_1)\big)\ldots Pr\big(\overline{\mathcal{R}}_{\overline{B}'_{N/m}}(\theta_1)\big)} \tag{3.56}$$

From Equations (3.53), (3.56), and Theorem 3.2, we have that

$$\forall\overline{B}\in\mathbf{B}, \overline{B}\neq\tilde{B} : \overline{B} = \overline{B}' \Rightarrow \frac{P\big(\overline{\mathcal{R}}_{\overline{B}}(\theta_1)\big)}{P\big(\overline{\mathcal{R}}_{\overline{B}'}(\theta_1)\big)} = 1 \tag{3.57}$$

$$\textit{Equations (3.56) and (3.57)} \Rightarrow \frac{P\big(\overline{\mathcal{R}}_{\overline{D}}(\theta_1)\big)}{P\big(\overline{\mathcal{R}}_{\overline{D}'}(\theta_1)\big)} = \frac{P\big(\overline{\mathcal{R}}_{\tilde{B}}(\theta_1)\big)}{P\big(\overline{\mathcal{R}}_{\tilde{B}'}(\theta_1)\big)} \leq e^{\epsilon_1} \tag{3.58}$$

As a result, the optimization of $\overline{\mathcal{R}}_{\overline{D}}(\theta_1)$ is $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP given the data $\overline{D}$ (which is $\epsilon_1/\gamma_{\mathbf{x}}$-DP (Lemma 3.10)), since the Equation (3.58) does hold for any tuple $\overline{x}_e \in \overline{D}$. This is consistent with the parallel composition property in DP [100], in which batches can be considered disjoint datasets and the optimization of the function on one batch does not affect the privacy guarantee in any other batch, even the objective function given one batch can be slightly different from the objective function given any other batch in $\overline{\mathbf{B}}$. In addition, $\forall t\in[1,T]$, the optimization of $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ does not use any additional information from the original data D. Consequently, the privacy budget is $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$ across T training steps, following the post-processing property in DP [100] (**Result 3**).

Similarly, we can also prove that optimizing the data reconstruction function $\overline{\mathcal{R}}_{\overline{B}_t^{adv}}(\theta_1)$ given the DP adversarial examples crafted in Equations (3.48) and (3.49), i.e., $\overline{x}_j^{\text{adv}}$, is also $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP given $t \in [1, T]$ on the training data D. First, DP adversarial examples $\overline{x}_j^{\text{adv}}$ are crafted from perturbed benign examples $\overline{x}_j$. As a result, the computation of the batch $\overline{B}_t^{adv}$ of DP adversarial examples is 1) $(\epsilon_1/\gamma_{\mathbf{x}})$-DP (the post-processing property of DP [100]), and 2) does not access the original data $\forall t \in [1, T]$. In addition, the computation of $\overline{\mathbf{h}}_{1\overline{B}_t^{adv}}$ and the optimization of $\overline{\mathcal{R}}_{\overline{B}_t^{adv}}(\theta_1)$ correspondingly are ϵ_1/γ-DP and ϵ_1-DP. In fact, the data reconstruction function $\overline{\mathcal{R}}_{\overline{B}_t^{adv}}$ is presented as follows:

$$\begin{aligned}
\overline{\mathcal{R}}_{\overline{B}_t^{adv}}(\theta_1) &= \sum_{\overline{x}_j^{\text{adv}} \in \overline{B}_t^{adv}} \Big[\sum_{i=1}^{d} \big(\frac{1}{2}\theta_{1i}\overline{h}_j^{\text{adv}}\big) - \overline{x}_j^{\text{adv}}\widetilde{x}_j^{\text{adv}} \Big] \\
&= \sum_{\overline{x}_j^{\text{adv}} \in \overline{B}_t^{adv}} \Big[\sum_{i=1}^{d} \big(\frac{1}{2}\theta_{1i}\overline{h}_j^{\text{adv}}\big) - \overline{x}_j\widetilde{x}_j^{\text{adv}} - \mu \cdot \text{sign}\Big(\nabla_{\overline{x}_j}\mathcal{L}\big(f(\overline{x}_j, \theta), y(\overline{x}_j)\big)\Big)\widetilde{x}_j^{\text{adv}} \Big] \\
&= \sum_{\overline{x}_j^{\text{adv}} \in \overline{B}_t^{adv}} \Big[\sum_{i=1}^{d} \big(\frac{1}{2}\theta_{1i}\overline{h}_j^{\text{adv}}\big) - \overline{x}_j\widetilde{x}_j^{\text{adv}} \Big] - \sum_{\overline{x}_j^{\text{adv}} \in \overline{B}_t^{adv}} \mu \cdot \text{sign}\Big(\nabla_{\overline{x}_j}\mathcal{L}\big(f(\overline{x}_j, \theta), y(\overline{x}_j)\big)\Big)\widetilde{x}_j^{\text{adv}}
\end{aligned} \tag{3.59}$$

where $h_j^{\text{adv}} = \theta_1^T \overline{x}_j^{\text{adv}}, \overline{h}_j^{\text{adv}} = h_j^{\text{adv}} + \frac{2}{m}Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$, and $\widetilde{x}_j^{\text{adv}} = \theta_1 \overline{h}_j^{\text{adv}}$. The right summation component in Equation (3.59) does not disclose any additional information, since the $sign(\cdot)$ function is computed from perturbed benign examples (the post-processing property in DP [100]). Meanwhile, the left summation component has the same form with $\overline{\mathcal{R}}_{\overline{B}_t}(\theta_1)$ in Equation (3.40). Therefore, we can employ the Proof 3.2.4 in Theorem 3.2, by replacing the coefficients $\Phi = \{\frac{1}{2}h_i, x_i\}$ with $\Phi = \{\frac{1}{2}h_j^{\text{adv}}, x_j\}$ to prove that the optimization of $\overline{\mathcal{R}}_{\overline{B}_t^{adv}}(\theta_1)$ is $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP. As a result, Theorem 3.3 does hold. (**Result 4**)

In addition to the Result 4, by applying the same analysis in Result 3, we can further show that the optimization of $\overline{\mathcal{R}}_{D^{adv}}(\theta_1)$ is $(\epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1)$-DP given the DP adversarial examples D^{adv} crafted using the data $\overline{D}$ across T training steps, since batches used to created DP adversarial examples are disjoint and fixed across epochs. It is also straightforward to conduct the same analysis in Result 2, in order to prove that the computation of the first affine transformation $\overline{\mathbf{h}}_{1\overline{B}_t^{adv}} = \{\overline{\theta}_1^T \overline{x}_j^{\text{adv}} + \frac{2}{m} Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})\}_{\overline{x}_j^{\text{adv}} \in \overline{B}_t^{adv}}$ given the batch of DP adversarial examples $\overline{B}_t^{adv}$, is (ϵ_1/γ)-DP with $t \in [1, T]$ training steps. This is also true given the data level D^{adv}. (**Result 5**)

Regarding the output layer, the Algorithm 3.3 preserves $(\epsilon_1/\gamma + \epsilon_2)$-DP in optimizing the adversarial objective function $\overline{L}_{\overline{B}_t \cup \overline{B}_t^{\text{adv}}}(\theta_2)$ (Theorem 3.4). We apply the same technique to preserve $(\epsilon_1/\gamma + \epsilon_2)$-DP across T training steps given disjoint and fixed batches derived from the private training data D. In addition, as our objective functions $\overline{\mathcal{R}}$ and $\overline{L}$ are always optimized given two disjoint batches $\overline{B}_t$ and $\overline{B}_t^{\text{adv}}$, the privacy budget used to preserve DP in these functions is $(\epsilon_1 + \epsilon_1/\gamma + \epsilon_2)$, following the *parallel composition* property in DP [100]. **(Result 6)**

With the **Results 1-6**, all the computations and optimizations in the Algorithm 3.3 are DP following the post-processing property in DP [100], by working on perturbed inputs and perturbed coefficients. The crafting and utilizing processes of DP adversarial examples based on the perturbed benign examples do not disclose any additional information. The optimization of our DP adversarial objective function at the output layer is DP to protect the ground-truth labels. More importantly, the DP guarantee in learning given the whole dataset level $\overline{D}$ is equivalent to the DP guarantee in learning on disjoint and fixed batches across epochs. Consequently, Algorithm 3.3 preserves $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$-DP in learning private parameters $\overline{\theta} = \{\overline{\theta}_1, \overline{\theta}_2\}$ given the training data D across T training steps. Note that the $\epsilon_1/\gamma_{\mathbf{x}}$ is counted for the perturbation on the benign examples. Theorem 3.5 does hold. $\square$

3.2.6 Certified Robustness

Now, we establish the correlation between our mechanism and certified robustness. In the *inference time*, to derive the certified robustness condition against adversarial examples $x+\alpha$, i.e., $\forall\alpha \in l_p(1)$, PixelDP randomizes the function $f(x)$ by injecting *robustness noise* σ_r into either input x or a hidden layer, i.e., $x' = x + Lap(\frac{\Delta_r^x}{\epsilon_r})$ or $h' = h + Lap(\frac{\Delta_r^h}{\epsilon_r})$, where Δ_r^x and Δ_r^h are the sensitivities of x and h, measuring how much x and h can be changed given the perturbation $\alpha \in l_p(1)$ in the input x. Monte Carlo estimation of the expected values $\hat{\mathbb{E}}f(x)$, $\hat{\mathbb{E}}_{lb}f_k(x)$, and $\hat{\mathbb{E}}_{ub}f_k(x)$ are used to derive the robustness condition in Equation (3.37).

On the other hand, in our mechanism, the privacy noise σ_p includes Laplace noise injected into both input x, i.e., $\frac{1}{m}Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$, and its affine transformation h, i.e., $\frac{2}{m}Lap(\frac{\Delta_{\mathcal{R}}}{\epsilon_1})$. Note that the perturbation of $\overline{\mathcal{L}}_{2\overline{B}_t}(\theta_2)$ is equivalent to $\overline{\mathcal{L}}_{2\overline{B}_t}(\theta_2) = \sum_{k=1}^{K}\sum_{\overline{x}_i}(\mathbf{h}_{\pi i}y_{ik}W_{\pi k} + \frac{1}{m}Lap(\frac{\Delta_{\mathcal{L}2}}{\epsilon_2})W_{\pi k})$. This helps us to avoid injecting the noise directly into the coefficients $\mathbf{h}_{\pi i}y_{ik}$. The correlation between our DP preservation and certified robustness lies in the correlation between the privacy noise σ_p and the robustness noise σ_r.

We can derive a robustness bound by projecting the privacy noise σ_p on the scale of the robustness noise σ_r. Given the input x, let $\kappa = \frac{\Delta_{\mathcal{R}}}{m\epsilon_1}/\frac{\Delta_r^x}{\epsilon_r}$, in our mechanism we have that: $\overline{x} = x + Lap(\kappa\Delta_r^x/\epsilon_r)$. By applying a group privacy size κ [100, 101], the scoring function $f(x)$ satisfies ϵ_r-PixelDP given $\alpha \in l_p(\kappa)$, or equivalently is ϵ_r/κ-PixelDP given $\alpha \in l_p(1)$, $\delta_r = 0$. By applying Lemma 3.8, we have

$$\forall k, \forall\alpha \in l_p(\kappa) : \mathbb{E}f_k(x) \le e^{\epsilon_r}\mathbb{E}f_k(x+\alpha),$$
$$or \quad \forall k, \forall\alpha \in l_p(1) : \mathbb{E}f_k(x) \le e^{\frac{\epsilon_r}{\kappa}}\mathbb{E}f_k(x+\alpha)$$

With that, we can achieve a robustness condition against $l_p(\kappa)$-norm attacks, as follows:

$$\hat{\mathbb{E}}_{lb} f_k(x) > e^{2\epsilon_r} \max_{i:i \neq k} \hat{\mathbb{E}}_{ub} f_i(x) \tag{3.60}$$

with the probability $\geq \eta_x$-confidence, derived from the Monte Carlo estimation of $\hat{\mathbb{E}} f(x)$. Our mechanism also perturbs h (Equation (3.40)). Given $\varphi = \frac{2\Delta_{\mathcal{R}}}{m\epsilon_1} / \frac{\Delta_r^h}{\epsilon_r}$, we further have $\overline{h} = h + Lap(\frac{\varphi \Delta_r^h}{\epsilon_r})$. Therefore, the scoring function $f(x)$ also satisfies ϵ_r-PixelDP given the perturbation $\alpha \in l_p(\varphi)$. In addition to the robustness to the $l_p(\kappa)$-norm attacks, we achieve an additional robustness bound in Equation (3.60) against $l_p(\varphi)$-norm attacks. Similar to PixelDP, these robustness conditions can be achieved as randomization processes in the inference time. They can be considered as *two independent and certified defensive mechanisms* applied against two l_p-norm attacks, i.e., $l_p(\kappa)$ and $l_p(\varphi)$.

One challenging question here is: *"What is the general robustness bound, given κ and φ?"* Intuitively, our model is robust to attacks with $\alpha \in l_p(\frac{\kappa\varphi}{\kappa+\varphi})$. We leverage the theory of *sequential composition* in DP [100] to theoretically answer this question. Given S independent mechanisms $\mathcal{M}_1, \ldots, \mathcal{M}_S$, whose privacy guarantees are $\epsilon_1, \ldots, \epsilon_S$-DP with $\alpha \in l_p(1)$. Each mechanism $\mathcal{M}_s$, which takes the input x and outputs the value of $f(x)$ with the Laplace noise only injected to randomize the layer s (i.e., no randomization at any other layers), denoted as $f^s(x)$, is defined as: $\forall s \in [1, S], \mathcal{M}_s f(x) : \mathbb{R}^d \rightarrow f^s(x) \in \mathbb{R}^K$. We aim to derive a generalized robustness of any composition scoring function $f(\mathcal{M}_1, \ldots, \mathcal{M}_s | x) : \prod_{s=1}^{S} \mathcal{M}_s f(x)$ bounded in $[0, 1]$, defined as follows:

$$f(\mathcal{M}_1, \ldots, \mathcal{M}_S | x) : \mathbb{R}^d \rightarrow \prod_{s \in [1,S]} f^s(x) \in \mathbb{R}^K \tag{3.61}$$

Our setting follows the sequential composition in DP [100]. Thus, we can prove that the expected value $\mathbb{E} f(\mathcal{M}_1, \ldots, \mathcal{M}_S | x)$ is insensitive to small perturbations $\alpha \in l_p(1)$ in Lemma 3.12, and we derive our composition of robustness in Theorem 3.6, as follows:

Lemma 3.12. *Given S independent mechanisms $\mathcal{M}_1, \ldots, \mathcal{M}_S$, which are $\epsilon_1, \ldots, \epsilon_S$-DP w.r.t a l_p-norm metric, then the expected output value of any sequential function f of them, i.e., $f(\mathcal{M}_1, \ldots, \mathcal{M}_S|x) \in [0,1]$, satisfies:*

$$\forall \alpha \in l_p(1) : \mathbb{E}f(\mathcal{M}_1, \ldots, \mathcal{M}_S|x) \leq e^{(\sum_{s=1}^{S} \epsilon_s)} \mathbb{E}f(\mathcal{M}_1, \ldots, \mathcal{M}_S|x+\alpha)$$

Proof of Lemma 3.12

Proof. Thanks to the sequential composition theory in DP [100], $f(\mathcal{M}_1, \ldots, \mathcal{M}_S|x)$ is $(\sum_s \epsilon_s)$-DP, since for any $O = \prod_{s=1}^{S} o_s \in \prod_{s=1}^{S} f^s(x)(\in \mathbb{R}^K)$, we have that

$$\frac{P\big(f(\mathcal{M}_1, \ldots, \mathcal{M}_S|x) = O\big)}{P\big(f(\mathcal{M}_1, \ldots, \mathcal{M}_S|x+\alpha) = O\big)} = \frac{P(\mathcal{M}_1 f(x) = o_1) \ldots P(\mathcal{M}_S f(x) = o_S)}{P(\mathcal{M}_1 f(x+\alpha) = o_1) \ldots P(\mathcal{M}_S f(x+\alpha) = o_S)}$$
$$\leq \prod_{s=1}^{S} \text{exp}(\epsilon_s) = e^{(\sum_{s=1}^{S} \epsilon_s)}$$

As a result, we have

$$P\big(f(\mathcal{M}_1, \ldots, \mathcal{M}_S|x)\big) \leq e^{(\sum_i \epsilon_i)} P\big(f(\mathcal{M}_1, \ldots, \mathcal{M}_S|x+\alpha)\big)$$

The sequential composition of the expected output is as:

$$\begin{aligned}\mathbb{E}f(\mathcal{M}_1,\ldots,\mathcal{M}_S|x) &= \int_0^1 P\big(f(\mathcal{M}_1,\ldots,\mathcal{M}_S|x) > t\big)dt \\ &\leq e^{(\sum_s \epsilon_s)} \int_0^1 P\big(f(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) > t\big)dt \\ &= e^{(\sum_s \epsilon_s)} \mathbb{E}f(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha)\end{aligned}$$

Lemma 3.12 does hold. □

Theorem 3.6. *(Composition of Robustness) Given S independent mechanisms $\mathcal{M}_1,\ldots,\mathcal{M}_S$. Given any sequential function $f(\mathcal{M}_1,\ldots,\mathcal{M}_S|x)$, and let $\hat{\mathbb{E}}_{lb}$ and $\hat{\mathbb{E}}_{ub}$ are lower and upper bounds with an η-confidence, for the Monte Carlo estimation of $\hat{\mathbb{E}}f(\mathcal{M}_1,\ldots,\mathcal{M}_S|x) = \frac{1}{n}\sum_n f(\mathcal{M}_1,\ldots,\mathcal{M}_S|x)_n = \frac{1}{n}\sum_n(\prod_{s=1}^S f^s(x)_n)$.*

$$\forall x, \textit{if } \exists k \in K : \hat{\mathbb{E}}_{lb} f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x) > e^{2(\sum_{s=1}^S \epsilon_s)} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub} f_i(\mathcal{M}_1,\ldots,\mathcal{M}_S|x), \quad (3.62)$$

then the predicted label $k = \arg\max_k \hat{\mathbb{E}}f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x)$, is robust to adversarial examples $x+\alpha$, $\forall\alpha \in l_p(1)$, with probability $\geq \eta$, by satisfying: $\hat{\mathbb{E}}f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) > \max_{i:i\neq k}\hat{\mathbb{E}}f_i(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha)$, which is the targeted robustness condition in Equation (3.35).

Proof of Theorem 3.6

Proof. $\forall\alpha \in l_p(1)$, from Lemma 3.12, with probability $\geq \eta$, we have that

$$\hat{\mathbb{E}}f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) \geq \frac{\hat{\mathbb{E}}f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x)}{e^{(\sum_{s=1}^{s}\epsilon_s)}} \geq \frac{\hat{\mathbb{E}}_{lb}f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x)}{e^{(\sum_{s=1}^{S}\epsilon_s)}} \tag{3.63}$$

In addition, we also have

$$\begin{aligned} &\forall i \neq k : \hat{\mathbb{E}}f_{i:i\neq k}(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) \leq e^{(\sum_{s=1}^{S}\epsilon_s)}\hat{\mathbb{E}}f_{i:i\neq k}(\mathcal{M}_1,\ldots,\mathcal{M}_S|x) \\ &\Rightarrow \forall i \neq k : \hat{\mathbb{E}}f_i(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) \leq e^{(\sum_{s=1}^{S}\epsilon_s)} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub}f_i(\mathcal{M}_1,\ldots,\mathcal{M}_S|x) \end{aligned} \tag{3.64}$$

Using the hypothesis (Equation (3.62)) and the first inequality (Equation (3.63)), we have that

$$\begin{aligned} \hat{\mathbb{E}}f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) &> \frac{e^{2(\sum_{s=1}^{S}\epsilon_s)}\max_{i:i\neq k}\hat{\mathbb{E}}_{ub}f_i(\mathcal{M}_1,\ldots,\mathcal{M}_S|x)}{e^{(\sum_{s=1}^{S}\epsilon_s)}} \\ &> e^{(\sum_{s=1}^{S}\epsilon_s)} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub}f_i(\mathcal{M}_1,\ldots,\mathcal{M}_S|x) \end{aligned}$$

Now, we apply the third inequality (Equation (3.64)), we have that

$$\begin{aligned} &\forall i \neq k : \hat{\mathbb{E}}f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) > \hat{\mathbb{E}}f_i(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) \\ &\Leftrightarrow \hat{\mathbb{E}}f_k(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) > \max_{i:i\neq k} \hat{\mathbb{E}}f_i(\mathcal{M}_1,\ldots,\mathcal{M}_S|x+\alpha) \end{aligned}$$

The Theorem 3.6 does hold. □

There is no η_s-confidence for each mechanism s, since we do not estimate the expected value $\hat{\mathbb{E}}f^s(x)$ independently. To apply the composition of robustness in our mechanism, the noise injections into the input x and its affine transformation h can be considered as two mechanisms $\mathcal{M}_x$ and $\mathcal{M}_h$, sequentially applied as $(\mathcal{M}_h(x), \mathcal{M}_x(x))$. When $\mathcal{M}_h(x)$ is applied by invoking $f(x)$ with independent draws in the noise χ_2, the noise χ_1 injected into x is fixed; and vice-versa. By applying group privacy [100] with sizes κ and φ, the scoring functions $f^x(x)$ and $f^h(x)$, given $\mathcal{M}_x$ and $\mathcal{M}_h$, are ϵ_r/κ-DP and ϵ_r/φ-DP with $\alpha \in l_p(1)$. With Theorem 3.6, we have a generalized bound as follows:

Corollary 3.1. *(StoBatch Robustness). Given: $\forall x$, if $\exists k \in K : \hat{\mathbb{E}}_{lb} f_k(\mathcal{M}_h, \mathcal{M}_x|x) > e^{2\epsilon_r} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub} f_i(\mathcal{M}_h, \mathcal{M}_x|x)$ (i.e., Equation (3.62)), then the predicted label k of our function $f(\mathcal{M}_h, \mathcal{M}_x|x)$ is robust to perturbations $\alpha \in l_p(\frac{\kappa\varphi}{\kappa+\varphi})$ with the probability $\geq \eta$, by satisfying $\forall \alpha \in l_p(\frac{\kappa\varphi}{\kappa+\varphi}) : \hat{\mathbb{E}} f_k(\mathcal{M}_h, \mathcal{M}_x|x+\alpha) > \max_{i:i\neq k} \hat{\mathbb{E}} f_i(\mathcal{M}_h, \mathcal{M}_x|x+\alpha)$*

Proof of Corollary 3.1

Proof. $\forall \alpha \in l_p(1)$, by applying Theorem 3.6, we have

$$\begin{aligned}\hat{\mathbb{E}}_{lb} f_k(\mathcal{M}_h, \mathcal{M}_x|x) &> e^{2(\frac{\epsilon_r}{\kappa}+\frac{\epsilon_r}{\varphi})} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub} f_i(\mathcal{M}_h, \mathcal{M}_x|x) \\ &> e^{2(\frac{\kappa+\varphi}{\kappa\varphi})\epsilon_r} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub} f_i(\mathcal{M}_h, \mathcal{M}_x|x) = e^{2(\epsilon_r/\frac{\kappa\varphi}{\kappa+\varphi})} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub} f_i(\mathcal{M}_h, \mathcal{M}_x|x)\end{aligned}$$

Furthermore, by applying group privacy, we have that

$$\forall \alpha \in l_p(\frac{\kappa\varphi}{\kappa+\varphi}) : \hat{\mathbb{E}}_{lb} f_k(\mathcal{M}_h, \mathcal{M}_x|x) > e^{2\epsilon_r} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub} f_i(\mathcal{M}_h, \mathcal{M}_x|x) \tag{3.65}$$

By applying Proof 3.2.6, it is straight to have

$$\forall \alpha \in l_p(\frac{\kappa\varphi}{\kappa+\varphi}) : \hat{\mathbb{E}} f_k(\mathcal{M}_h, \mathcal{M}_x | x+\alpha) > \max_{i:i\neq k} \hat{\mathbb{E}} f_k(\mathcal{M}_h, \mathcal{M}_x | x+\alpha)$$

with probability $\geq \eta$. Corollary 3.1 does hold. □

Compared with state-of-the-art robustness analysis [70, 101], in which either the input space or the latent space are randomized, the advantage of our robustness bound is the composition of different levels of robustness in both input and latent spaces.

3.2.7 Verified Inference

At the inference time, we implement a *verified inference* (Algorithm 3.4, Subsection 3.2.3) to return a *robustness size guarantee* for each example x, i.e., the maximal value of $\frac{\kappa\varphi}{\kappa+\varphi}$, for which the robustness condition in Corollary 3.1 holds. Maximizing $\frac{\kappa\varphi}{\kappa+\varphi}$ is equivalent to maximizing the robustness epsilon ϵ_r, which is the only parameter controlling the size of $\frac{\kappa\varphi}{\kappa+\varphi}$; since, all the other hyper-parameters, i.e., $\Delta_{\mathcal{R}}$, m, ϵ_1, ϵ_2, θ_1, θ_2, Δ_r^x, and Δ_r^h are fixed given a well-trained model $f(x)$:

$$(\frac{\kappa\varphi}{\kappa+\varphi})_{max} = \max_{\epsilon_r} \frac{\Delta_{\mathcal{R}}\epsilon_r}{m\epsilon_1(\Delta_r^x + \Delta_r^h/2)}$$
$$\text{s.t. } \hat{\mathbb{E}}_{lb} f_k(x) > e^{2\epsilon_r} \max_{i:i\neq k} \hat{\mathbb{E}}_{ub} f_i(x) \text{ (i.e., Equation (3.62))} \quad (3.66)$$

The prediction on an example x is robust to attacks up to $(\frac{\kappa\varphi}{\kappa+\varphi})_{max}$. The failure probability 1-η can be made arbitrarily small by increasing the number of invocations of $f(x)$, with independent draws in the noise. Similar to [101], Hoeffding's inequality is applied to *bound* the approximation error in $\hat{\mathbb{E}} f_k(x)$ and to *search* for the robustness bound $(\frac{\kappa\varphi}{\kappa+\varphi})_{max}$. We use the following sensitivity bounds $\Delta_r^h = \beta\|\theta_1\|_\infty$ where $\|\theta_1\|_\infty$ is the maximum 1-norm of θ_1's rows, and $\Delta_r^x = \mu d$ for l_∞ attacks. In the Monte Carlo Estimation of $\hat{\mathbb{E}}f(x)$, we also propose a new method to draw independent noise to control the *distribution shifts* between training and inferring, in order to improve the verified inference effectiveness, without affecting the DP protection and the robustness bounds. Details of this method is described in the following subsection.

Algorithm 3.4 Verified Inferring

```
Input: (an input $x$, attack size $\mu_a$)
1: Compute robustness size $(\frac{\kappa\varphi}{\kappa+\varphi})_{max}$ in Equation (3.66) of $x$
2: if $(\frac{\kappa\varphi}{\kappa+\varphi})_{max} \geq \mu_a$ then
3:     Return $isRobust(x) = True$, label $k$, $(\frac{\kappa\varphi}{\kappa+\varphi})_{max}$
4: else
5:     Return $isRobust(x) = False$, label $k$, $(\frac{\kappa\varphi}{\kappa+\varphi})_{max}$
```

Effective Monte Carlo Estimation of $\hat{\mathbb{E}}f(x)$ Recall that the Monte Carlo estimation is applied to estimate the expected value $\hat{\mathbb{E}}f(x) = \frac{1}{n}\sum_n f(x)_n$, where n is the number of invocations of $f(x)$ with independent draws in the noise, i.e., $\frac{1}{m}Lap(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1})$ and $\frac{2}{m}Lap(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1})$ in our case. When ϵ_1 is small (indicating a strong privacy protection), it causes a *notably large distribution shift between training and inference, given independent draws of the Laplace noise*.

In fact, let us denote a single draw in the noise as $\chi_1 = \frac{1}{m}Lap(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1})$ used to train the function $f(x)$, the model converges to the point that the noise χ_1 and $2\chi_2$ need to be correspondingly added into x and h in order to make correct predictions. χ_1 can be approximated as $Lap(\chi_1, \varrho)$, where $\varrho \to 0$. It is clear that independent draws of the noise $\frac{1}{m}Lap(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1})$ have *distribution shifts* with the fixed noise $\chi_1 \approxeq Lap(\chi_1, \varrho)$. These distribution shifts can also be large, when noise is large. We have experienced that these distribution shifts in having independent draws of noise to estimate $\hat{\mathbb{E}}f(x)$ can notably degrade the inference accuracy of the scoring function, when privacy budget ϵ_1 is small resulting in a large amount of noise injected to provide strong privacy guarantees.

To address this, one solution is to increase the number of invocations of $f(x)$, i.e., n, to a huge number per prediction. However, this is impractical in real-world scenarios. We propose a novel way to draw independent noise following the distribution of $\chi_1 + \frac{1}{m}Lap(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1}/\psi)$ for the input x and $2\chi_2 + \frac{2}{m}Lap(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1}/\psi)$ for the affine transformation h, where ψ is a hyper-parameter to control the distribution shifts. This approach works well and does not affect the DP bounds and the certified robustness condition, since: **(1)** Our mechanism achieves both DP and certified robustness in the training process; and **(2)** It is clear that $\hat{\mathbb{E}}f(x) = \frac{1}{n}\sum_n f(x)_n = \frac{1}{n}\sum_n g\big(a(x + \chi_1 + \frac{1}{m}Lap_n(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1}/\psi), \theta_1) + 2\chi_2 + \frac{2}{m}Lap_n(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1}/\psi), \theta_2\big)$, where $Lap_n(0, \frac{\Delta_{\mathcal{R}}}{\epsilon_1}/\psi)$ is the n-th draw of the noise. When $n \to \infty$, $\hat{\mathbb{E}}f(x)$ will converge to $\frac{1}{n}\sum_n g\big(a(x + \chi_1, \theta_1) + 2\chi_2, \theta_2\big)$, which aligns well with the convergence point of the scoring function $f(x)$. Injecting χ_1 and $2\chi_2$ to x and h during the estimation of $\hat{\mathbb{E}}f(x)$ yields better performance, without affecting the DP and the composition robustness bounds.

3.2.8 Distributed Training

In the vanilla iterative batch-by-batch training for DP DNNs, at each step, only one batch of examples can be used to train our model, so that the privacy loss can be computed [74, 73, 78, 79]. Parameters θ_1 and θ_2 are independently updated (Lines 4-12, Algorithm 3.3). This prevents us from applying *practical adversarial training* [187, 80], in which *distributed training using synchronized SGD* on many GPUs (e.g., 128 GPUs) is used to scale adversarial training to large DNNs. Each GPU processes a mini-batch of 32 images (i.e., the total batch size is $128 \times 32 = 4,096$).

To overcome this, a well-applied technique [73] is to fine-tune a limited number of layers, such as a fully connected layer and the output layer, under DP of a pre-trained model, i.e., VGG16, trained over a public and large dataset, e.g., ImageNet, in order to handle simpler tasks on smaller private datasets, e.g., CIFAR-10. Although this approach works well, there are several utility and security concerns: **(1)** Suitable public data may not always be available, especially for highly sensitive data; **(2)** Trojans can be implanted in the pre-trained model for backdoor attacks [188]; and **(3)** Public data can be poisoned [189]. Fine-tuning a limited number of layers may not be secure; while fine-tuning an entire of a large pre-trained model iteratively batch-by-batch is still inefficient.

To address this bottleneck, we leverage the training recipe of [187, 80] to propose a distributed training algorithm, called **StoBatch** (Figure 3.6b), in order to efficiently train large DP DNNs in adversarial learning, without affecting the DP protection (Algorithm 3.5, Subsection 3.2.3). In StoBatch, fixed and disjoint batches $\overline{\mathbf{B}}$ are distributed to $N/(2m)$ local trainers, each of which have two batches $\{\overline{B}_{i1}, \overline{B}_{i2}\}$ randomly picked from $\overline{\mathbf{B}}$ with $i \in [1, N/(2m)]$ (Line 4, Algorithm 3.5). At each training step t, we randomly pick $\mathbb{N}$ local trainers, each of which gets the latest global parameters θ from the parameter server. A local trainer i will compute the gradients $\nabla_i\theta_1$ and $\nabla_i\theta_2$ to optimize the DP objective functions $\overline{\mathcal{R}}$ and $\overline{L}$ using its local batch $\overline{B}_{i1}$ and ensemble DP adversarial examples crafted from $\overline{B}_{i2}$ (Lines 5-14, Algorithm 3.5). The gradients will be sent back to the parameter server for a synchronized SGD (Lines 15-16, Algorithm 3.5), as follows: $\theta_1 \leftarrow \theta_1 - \frac{\varrho_t}{\mathbb{N}} \sum_{i \in [1,\mathbb{N}]} \nabla_i\theta_1, \quad \theta_2 \leftarrow \theta_2 - \frac{\varrho_t}{\mathbb{N}} \sum_{i \in [1,\mathbb{N}]} \nabla_i\theta_2$. This enables us to train large DNNs with our DP adversarial learning, by training from multiple batches simultaneously with more adversarial examples, without affecting the DP guarantee in Theorem 3.5; since the optimization of one batch does not affect the DP protection at any other batch and at the dataset level D across T training steps (**Theorem 3.5**).

In addition, the *average errors of our approximation functions* are always *bounded*, and are *independent* of the number of data instances N in D (details described in the following subsection). This further ensures that our functions can be applied in large datasets.

Our approach can be extended into two different complementary scenarios: **(1)** Distributed training for each local trainer i, in which the batches $\{\overline{B}_{i1}, \overline{B}_{i2}\}$ can be located across $\mathbb{M}$ GPUs to efficiently compute the gradients $\nabla_i\theta_1 = \frac{1}{\mathbb{M}} \sum_{j \in [1,\mathbb{M}]} \nabla_{i,j}\theta_1$ and $\nabla_i\theta_2 =$

Algorithm 3.5 StoBatch Training

Input: Database D, loss function L, parameters θ, batch size m, learning rate ϱ_t, privacy budgets: ϵ_1 and ϵ_2, robustness parameters: ϵ_r, Δ_r^x, and Δ_r^h, adversarial attack size μ_a, the number of invocations n, ensemble attacks A, parameters ψ and ξ, the size $|\mathbf{h}_\pi|$ of $\mathbf{h}_\pi$, a number of $\mathbb{N}$ random local trainers ($\mathbb{N} \leq N/(2m)$)

```
1: **Draw Noise** $\chi_1 \leftarrow [Lap(\frac{\Delta_\mathcal{R}}{\epsilon_1})]^d$, $\chi_2 \leftarrow [Lap(\frac{\Delta_\mathcal{R}}{\epsilon_1})]^\beta$, $\chi_3 \leftarrow [Lap(\frac{\Delta_{\mathcal{L}2}}{\epsilon_2})]^{|\mathbf{h}_\pi|}$
2: **Randomly Initialize** $\theta = \{\theta_1, \theta_2\}$, $\mathbf{B} = \{B_1, \ldots, B_{N/m}\}$ s.t. $\forall B \in \mathbf{B} : B$ is a batch
   with the size $m$, $B_1 \cap \ldots \cap B_{N/m} = \emptyset$, and $B_1 \cup \ldots \cup B_{N/m} = D$, $\overline{\mathbf{B}} = \{\overline{B}_1, \ldots, \overline{B}_{N/m}\}$
   where $\forall i \in [1, N/m] : \overline{B}_i = \{\overline{x} \leftarrow x + \frac{\chi_1}{m}\}_{x \in B_i}$
3: **Construct** a deep network $f$ with **hidden layers** $\{\mathbf{h}_1 + \frac{2\chi_2}{m}, \ldots, \mathbf{h}_\pi\}$, where $\mathbf{h}_\pi$ is the
   last hidden layer
4: **Distribute** fixed and disjoint batches $\overline{\mathbf{B}}$ to $N/(2m)$ local trainers, each of which have
   two batches $\{\overline{B}_{i1}, \overline{B}_{i2}\}$ randomly picked from $\overline{\mathbf{B}}$ with $i \in [1, N/(2m)]$
5: **for** $t \in [T]$ **do**
6:     **Randomly Pick** $\mathbb{N}$ local trainers, each of which **Gets** the latest global parameters
       $\theta$ from the parameter server
7:     **for** $i \in [1, \mathbb{N}]$ **do**
8:         **Assign** $\overline{B}_{t,i} \leftarrow \overline{B}_{i1}$
9:         **Ensemble DP Adversarial Examples:**
10:        **Draw Random Perturbation Value** $\mu_t \in (0, 1]$, **Assign** $\overline{B}_{t,i}^{\text{adv}} \leftarrow \emptyset$
11:        **for** $l \in A$ **do**
12:            **Take** the next batch $\overline{B}_a \subset \overline{B}_{i2}$ with the size $m/|A|$
13:            $\forall \overline{x}_j \in \overline{B}_a$: **Craft** $\overline{x}_j^{\text{adv}}$ by using attack algorithm $A[l]$ with $l_\infty(\mu_t)$, $\overline{B}_{t,i}^{\text{adv}} \leftarrow$
   $\overline{B}_{t,i}^{\text{adv}} \cup \overline{x}_j^{\text{adv}}$
14:        **Compute** $\nabla_i\theta_1 \leftarrow \nabla_{\theta_1}\overline{\mathcal{R}}_{\overline{B}_{t,i} \cup \overline{B}_{t,i}^{\text{adv}}}(\theta_1)$, $\nabla_i\theta_2 \leftarrow \nabla_{\theta_2}\overline{L}_{\overline{B}_{t,i} \cup \overline{B}_{t,i}^{\text{adv}}}(\theta_2)$ with the noise
   $\frac{\chi_3}{m}$
15:        **Send** $\nabla_i\theta_1$ and $\nabla_i\theta_2$ to the parameter server
16:    **Descent:** $\theta_1 \leftarrow \theta_1 - \varrho_t \frac{1}{\mathbb{N}} \sum_{i \in [1,\mathbb{N}]} \nabla_i\theta_1$; $\theta_2 \leftarrow \theta_2 - \varrho_t \frac{1}{\mathbb{N}} \sum_{i \in [1,\mathbb{N}]} \nabla_i\theta_2$, on the
   parameter server
   **Output:** $\epsilon = (\epsilon_1 + \epsilon_1/\gamma_\mathbf{x} + \epsilon_1/\gamma + \epsilon_2)$-DP parameters $\theta = \{\theta_1, \theta_2\}$, robust model with
   an $\epsilon_r$ budget
```

$\frac{1}{\mathbb{M}}\sum_{j\in[1,\mathbb{M}]}\nabla_{i,j}\theta_2$; and **(2)** Federated training, given each local trainer can be considered as an independent party. In this setting, an independent party can further have different sizes of batches. As long as the global sensitivities $\Delta_{\mathcal{R}}$ and $\Delta_{\mathcal{L}2}$ are the same for all the parties, the DP guarantee in Theorem 3.5 does hold given D be the union of all local datasets from all the parties. This can be achieved by nomalizing all the inputs x to be in $[-1,1]^d$. This is a step forward compared with the classical federated learning [190]. We focus on the distributed training setting in this work, and reserve the federated learning scenarios for future exploration.

Approximation Error Bounds To compute how much error our polynomial approximation approaches (i.e., truncated Taylor expansions), $\widetilde{\mathcal{R}}_{B_t}(\theta_1)$ (Equation (3.39)) and $\mathcal{L}_{\overline{B}_t}\big(\theta_2\big)$, incur, we directly apply Lemma 4 in [60], Lemma 3 in [161], and the well-known error bound results in [164]. Note that $\widetilde{\mathcal{R}}_{B_t}(\theta_1)$ is the 1st-order Taylor series and $\mathcal{L}_{\overline{B}_t}\big(\theta_2\big)$ is the 2nd-order Taylor series following the implementation of [191]. Let us closely follow [60, 161, 164] to adapt their results into our scenario, as follows:

Given the truncated function $\widetilde{\mathcal{R}}_{B_t}(\theta_1) = \sum_{x_i\in B_t}\sum_{j=1}^{d}\sum_{l=1}^{2}\sum_{r=0}^{1}\frac{\mathbf{F}_{lj}^{(r)}(0)}{r!}\big(\theta_{1j}h_i\big)^r$, the original Taylor polynomial function $\widehat{\mathcal{R}}_{B_t}(\theta_1) = \sum_{x_i\in B_t}\sum_{j=1}^{d}\sum_{l=1}^{\infty}\sum_{r=0}^{1}\frac{\mathbf{F}_{lj}^{(r)}(0)}{r!}\big(\theta_{1j}h_i\big)^r$, the average error of the approximation is bounded as

$$\frac{1}{|B_t|}|\widehat{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widehat{\mathcal{R}}_{B_t}(\widehat{\theta}_1)| \leq \frac{4e\times d}{(1+e)^2} \tag{3.67}$$
$$\frac{1}{|B_t|}|\widehat{\mathcal{L}}_{B_t}(\widetilde{\theta}_2) - \widehat{\mathcal{L}}_{B_t}(\widehat{\theta}_2)| \leq \frac{e^2+2e-1}{e(1+e)^2}\times K \tag{3.68}$$

where $\widehat{\theta}_1 = \arg\min_{\theta_1}\widehat{\mathcal{R}}_{B_t}(\theta_1)$, $\widetilde{\theta}_1 = \arg\min_{\theta_1}\widetilde{\mathcal{R}}_{B_t}(\theta_1)$, $\widehat{\mathcal{L}}_{B_t}(\theta_2)$ is the original Taylor polynomial function of $\sum_{x_i\in B_t}\mathcal{L}\big(f(\overline{x}_i,\theta_2),y_i\big)$, $\widehat{\theta}_2 = \arg\min_{\theta_2}\widehat{\mathcal{L}}_{B_t}(\theta_2)$, and $\widetilde{\theta}_2 = \arg\min_{\theta_2}\mathcal{L}_{B_t}(\theta_2)$.

Proof. Let $U = \max_{\theta_1}\big(\widehat{\mathcal{R}}_{B_t}(\theta_1) - \widetilde{\mathcal{R}}_{B_t}(\theta_1)\big)$ and $S = \min_{\theta_1}\big(\widehat{\mathcal{R}}_{B_t}(\theta_1) - \widetilde{\mathcal{R}}_{B_t}(\theta_1)\big)$.

We have that $U \geq \widehat{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widetilde{\mathcal{R}}_{B_t}(\widetilde{\theta}_1)$ and $\forall \theta_1^* : S \leq \widehat{\mathcal{R}}_{B_t}(\theta_1^*) - \widetilde{\mathcal{R}}_{B_t}(\theta_1^*)$. Therefore, we have

$$\widehat{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widetilde{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widehat{\mathcal{R}}_{B_t}(\theta_1^*) + \widetilde{\mathcal{R}}_{B_t}(\theta_1^*) \leq U - S \tag{3.69}$$

$$\Leftrightarrow \widehat{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widehat{\mathcal{R}}_{B_t}(\theta_1^*) \leq U - S + \big(\widetilde{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widetilde{\mathcal{R}}_{B_t}(\theta_1^*)\big) \tag{3.70}$$

In addition, $\widetilde{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widetilde{\mathcal{R}}_{B_t}(\theta_1^*) \leq 0$, it is straightforward to have:

$$\widehat{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widehat{\mathcal{R}}_{B_t}(\theta_1^*) \leq U - S \tag{3.71}$$

If $U \geq 0$ and $S \leq 0$ then we have:

$$|\widehat{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widehat{\mathcal{R}}_{B_t}(\theta_1^*)| \leq U - S \tag{3.72}$$

Equation (3.72) holds for every θ_1^*, including $\widehat{\theta}_1$. Equation (3.72) shows that the error incurred by truncating the Taylor series approximate function depends on the maximum and minimum values of $\widehat{\mathcal{R}}_{B_t}(\theta_1) - \widetilde{\mathcal{R}}_{B_t}(\theta_1)$. This is consistent with [60, 161]. To quantify the magnitude of the error, we rewrite $\widehat{\mathcal{R}}_{B_t}(\theta_1) - \widetilde{\mathcal{R}}_{B_t}(\theta_1)$ as:

$$\widehat{\mathcal{R}}_{B_t}(\theta_1) - \widetilde{\mathcal{R}}_{B_t}(\theta_1) = \sum_{j=1}^{d}\big(\widehat{\mathcal{R}}_{B_t}(\theta_{1j}) - \widetilde{\mathcal{R}}_{B_t}(\theta_{1j})\big) \tag{3.73}$$

$$= \sum_{j=1}^{d}\Big(\sum_{i=1}^{|B_t|}\sum_{l=1}^{2}\sum_{r=3}^{\infty}\frac{\mathbf{F}_{lj}^{(r)}(z_{lj})}{r!}\big(g_{lj}(x_i,\theta_{1j}) - z_{lj}\big)^r\Big) \tag{3.74}$$

where $g_{1j}(x_i, \theta_{1j}) = \theta_{1j} h_i$ and $g_{2j}(x_i, \theta_{1j}) = \theta_{1j} h_i$.

By looking into the remainder of Taylor expansion for each j (i.e., following [60, 164]), with $z_j \in [z_{lj} - 1, z_{lj} + 1]$, $\frac{1}{|B_t|}\big(\widehat{\mathcal{R}}_{B_t}(\theta_{1j}) - \widetilde{\mathcal{R}}_{B_t}(\theta_{1j})\big)$ must be in the interval $\Big[\sum_l \frac{\min_{z_j} \mathbf{F}_{lj}^{(2)}(z_j)(z_j - z_{lj})^2}{2!}, \sum_l \frac{\max_{z_j} \mathbf{F}_{lj}^{(2)}(z_j)(z_j - z_{lj})^2}{2!}\Big]$. If $\sum_l \frac{\max_{z_j} \mathbf{F}_{lj}^{(2)}(z_j)(z_j - z_{lj})^2}{2!} \geq 0$ and $\sum_l \frac{\min_{z_j} \mathbf{F}_{lj}^{(2)}(z_j)(z_j - z_{lj})^2}{2!} \leq 0$, then we have that $|\frac{1}{|B_t|}\big(\widehat{\mathcal{R}}_{B_t}(\theta_1) - \widetilde{\mathcal{R}}_{B_t}(\theta_1)\big)| \leq \sum_{j=1}^{d} \sum_l \frac{\max_{z_j} \mathbf{F}_{lj}^{(2)}(z_j)(z_j - z_{lj})^2 - \min_{z_j} \mathbf{F}_{lj}^{(2)}(z_j)(z_j - z_{lj})^2}{2!}$. This can be applied to the case of our auto-encoder, as follows:

For the functions $\mathbf{F}_{1j}(z_j) = x_{ij} \log(1 + e^{-z_j})$ and $\mathbf{F}_{2j}(z_j) = (1 - x_{ij}) \log(1 + e^{z_j})$, we have $\mathbf{F}_{1j}^{(2)}(z_j) = \frac{x_{ij} e^{-z_j}}{(1+e^{-z_j})^2}$ and $\mathbf{F}_{2j}^{(2)}(z_j) = (1 - x_{ij})\frac{e^{z_j}}{(1+e^{z_j})^2}$. It can be verified that $\arg\min_{z_j} \mathbf{F}_{1j}^{(2)}(z_j) = \frac{-e}{(1+e)^2} < 0$, $\arg\max_{z_j} \mathbf{F}_{1j}^{(2)}(z_j) = \frac{e}{(1+e)^2} > 0$, $\arg\min_{z_j} \mathbf{F}_{2j}^{(2)}(z_j) = 0$, and $\arg\max_{z_j} \mathbf{F}_{2j}^{(2)}(z_j) = \frac{2e}{(1+e)^2} > 0$. Thus, the average error of the approximation is at most:

$$\frac{1}{|B_t|}|\widehat{\mathcal{R}}_{B_t}(\widetilde{\theta}_1) - \widehat{\mathcal{R}}_{B_t}(\widehat{\theta}_1)| \leq \Big[\big(\frac{e}{(1+e)^2} - \frac{-e}{(1+e)^2}\big) + \frac{2e}{(1+e)^2}\Big] \times d = \frac{4e \times d}{(1+e)^2} \quad (3.75)$$

Consequently, Equation (3.67) does hold. Similarly, by looking into the remainder of Taylor expansion for each label k, Equation (3.68) can be proved straightforwardly. In fact, by using the 2nd-order Taylor series with K categories, we have that: $\frac{1}{|B_t|}|\widehat{\mathcal{L}}_{B_t}(\widetilde{\theta}_2) - \widehat{\mathcal{L}}_{B_t}(\widehat{\theta}_2)| \leq \frac{e^2+2e-1}{e(1+e)^2} \times K$. □

3.2.9 Experiments

Model Configurations The MNIST database consists of handwritten digits [47]. Each example is a 28 × 28 size gray-level image. The CIFAR-10 dataset consists of color images belonging to 10 classes, i.e., airplanes, dogs, etc. The dataset is split into 50,000 training samples and 10,000 test samples [99]. Tiny Imagenet $(64 \times 64 \times 3)$ has 200 classes. Each class has 500 training images, 50 validation images, and 50 test images. We used the first thirty classes with data augmented, including horizontal flip and random brightness, in the Tiny ImageNet dataset in our experiment. In general, the dataset is split into 45,000 training samples and 1,500 test samples [102, 192]. The experiments were conducted on a server of 4 GPUs, each of which is an NVIDIA TITAN Xp, 12 GB with 3,840 CUDA cores. All the models share the same structure, consisting of 2 and 3 convolutional layers, respectively for MNIST and CIFAR-10 datasets, and a ResNet18 model for the Tiny ImageNet dataset.

Both fully-connected and convolution layers can be applied in the representation learning model $a(x, \theta_1)$. Given convolution layer, the computation of each feature map needs to be DP; since each of them independently reads a local region of input neurons. Therefore, the sensitivity $\Delta_{\mathcal{R}}$ can be considered the maximal sensitivity given any single feature map in the first affine transformation layer. In addition, each hidden neuron can only be used to reconstruct a unit patch of input units. That results in d (Lemma 3.9) being the size of the unit patch connected to each hidden neuron, e.g., $d = 9$ given a 3×3 unit patch, and β is the number of hidden neurons in a feature map.

MNIST: We used two convolutional layers (32 and 64 features). Each hidden neuron connects with a 5x5 unit patch. A fully-connected layer has 256 units. The batch size m was set to 2,499, $\xi = 1$, $\psi = 2$. I-FGSM, MIM, and MadryEtAl were used to draft $l_\infty(\mu)$ adversarial examples in training, with $T_\mu = 10$. Learning rate ϱ_t was set to $1e-4$. Given a predefined total privacy budget ϵ, ϵ_2 is set to be 0.1, and ϵ_1 is computed as: $\epsilon_1 = \frac{\epsilon - \epsilon_2}{(1 + 1/\gamma + 1/\gamma_{\mathbf{x}})}$. This will guarantee that $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2) = \epsilon$. $\Delta_{\mathcal{R}} = (14^2 + 2) \times 25$ and $\Delta_{\mathcal{L}2} = 2 \times 256$. The number of Monte Carlo sampling for certified inference n is set to 2,000.

CIFAR-10: We used three convolutional layers (128, 128, and 256 features). Each hidden neuron connects with a 4x4 unit patch in the first layer, and a 5x5 unit patch in other layers. One fully-connected layer has 256 neurons. The batch size m was set to 1,851, $\xi = 1.5$, $\psi = 10$, and $T_\mu = 3$. The ensemble of attacks A includes I-FGSM, MIM, and MadryEtAl. We use data augmentation, including random crop, random flip, and random contrast. Learning rate ϱ_t was set to $5e-2$. In the CIFAR-10 dataset, ϵ_2 is set to $(1 + r/3.0)$ and $\epsilon_1 = (1 + 2r/3.0)/(1 + 1/\gamma + 1/\gamma_{\mathbf{x}})$, where $r \geq 0$ is a ratio to control the total privacy budget ϵ in our experiment. For instance, given $r = 0$, we have that $\epsilon = (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2) = 2$. $\Delta_{\mathcal{R}} = 3 \times (14^2 + 2) \times 16$ and $\Delta_{\mathcal{L}2} = 2 \times 256$. $\mathbb{N}$ and $\mathbb{M}$ are set to 1 and 4 in the distributed training. The number of Monte Carlo sampling for certified inference n is set to 1,000.

Tiny ImageNet: We used a ResNet-18 model. Each hidden neuron connects with a 7x7 unit patch in the first layer, and 3x3 unit patch in other layers. The batch size m was set to 4,500, $\xi = 1.5$, $\psi = 10$, and $T_\mu = 10$. The ensemble of attacks A includes I-FGSM, MIM, and MadryEtAl. Learning rate ϱ_t was set to $1e - 2$. In the Tiny ImageNet dataset, ϵ_2 is set to 1 and $\epsilon_1 = (1 + r)/(1 + 1/\gamma + 1/\gamma_{\mathbf{x}})$, where $r \geq 0$ is a ratio to control the total privacy budget ϵ in our experiment. $\Delta_{\mathcal{R}} = 3 \times (32^2 + 2) \times 49$ and $\Delta_{\mathcal{L}2} = 2 \times 256$. $\mathbb{N}$ and $\mathbb{M}$ are set to 1 and 20 in the distributed training. The number of Monte Carlo sampling for certified inference n is set to 1,000.

Experimental Results In this subsection, we will show and explain the experimental results on MNIST and CIFAR datasets, and with the scenario of under strong attack.

Results on the MNIST Dataset Figure 3.7 illustrates the conventional accuracy of each model as a function of the privacy budget ϵ on the MNIST dataset under $l_\infty(\mu_a)$-norm attacks, with $\mu_a = 0.2$ (a pretty strong attack). It is clear that our StoBatch outperforms AdLM, DP-SGD, SecureSGD, and SecureSGD-AGM, in all cases, with $p < 1.32e - 4$. On average, we register a 22.36% improvement over SecureSGD ($p < 1.32e - 4$), a 46.84% improvement over SecureSGD-AGM ($p < 1.83e - 6$), a 56.21% improvement over AdLM ($p < 2.05e - 10$), and a 77.26% improvement over DP-SGD ($p < 5.20e - 14$), given our StoBatch mechanism. AdLM and DP-SGD achieve the worst conventional accuracies. There is no guarantee provided in AdLM and DP-SGD. Thus, the accuracy of the AdLM and DPSGD algorithms seem to show no effect against adversarial examples, when the privacy budget is varied. This is in contrast to our StoBatch model, the SecureSGD model, and the SecureSGD-AGM model, whose accuracies are proportional to the privacy budget.

When the privacy budget $\epsilon = 0.2$ (a tight DP protection), there are significant drops, in terms of conventional accuracy, given the baseline approaches. By contrast, our StoBatch mechanism only shows a small degradation in the conventional accuracy (6.89%, from 89.59% to 82.7%), compared with a 37% drop in SecureSGD (from 78.64% to 41.64%), and a 32.89% drop in SecureSGD-AGM (from 44.1% to 11.2%) on average, when the privacy budget ϵ goes from 2.0 to 0.2. At $\epsilon = 0.2$, our StoBatch mechanism achieves 82.7%, compared with 11.2% and 41.64% correspondingly for SecureSGD-AGM and SecureSGD. This is an important result, showing the ability to offer tight DP protections under adversarial example attacks in our model, compared with existing algorithms.

Figure 3.9 presents the conventional accuracy of each model as a function of the attack size μ_a on the MNIST dataset, under a strong DP guarantee, $\epsilon = 0.2$. Our StoBatch mechanism outperforms the baseline approaches in all cases. On average, our StoBatch model improves 44.91% over SecureSGD ($p < 7.43e - 31$), a 61.13% over SecureSGD-AGM ($p < 2.56e - 22$), a 52.21% over AdLM ($p < 2.81e - 23$), and a 62.20% over DP-SGD ($p < 2.57e - 22$). More importantly, our StoBatch model is resistant to different adversarial example algorithms with different attack sizes. When $\mu_a \geq 0.2$, AdLM, DP-SGD, SecureSGD, and SecureSGD-AGM become defenseless. We further register significantly drops in terms of accuracy, when μ_a is increased from 0.05 (a weak attack) to 0.6 (a strong attack), i.e., 19.87% on average given our StoBatch, across all attacks, compared with 27.76% (AdLM), 29.79% (DP-SGD), 34.14% (SecureSGD-AGM), and 17.07% (SecureSGD).

Figure 3.11 demonstrates the certified accuracy as a function of μ_a. The privacy budget is set to 1.0, offering a reasonable privacy protection. In PixelDP, the construction attack bound ϵ_r is set to 0.1, which is a pretty reasonable defense. With (small perturbation) $\mu_a \leq 0.2$, PixelDP achieves better certified accuracies under all attacks; since PixelDP does not preserve DP to protect the training data, compared with other models. Meanwhile, our StoBatch model outperforms all the other models when $\mu_a \geq 0.3$, indicating a stronger defense to more aggressive attacks. More importantly, our StoBatch has a consistent certified accuracy to different attacks given different attack sizes, compared with baseline approaches. In fact, when μ_a is increased from 0.05 to 0.6, our StoBatch shows a small drop (11.88% on average, from $84.29\%(\mu_a = 0.05)$ to $72.41\%(\mu_a = 0.6)$), compared with a huge drop of the PixelDP, i.e., from $94.19\%(\mu_a = 0.05)$ to $9.08\%(\mu_a = 0.6)$ on average under I-FGSM, MIM, and MadryEtAl attacks, and to $77.47\%(\mu_a = 0.6)$ under FGSM attack. Similarly, we also register significant drops in terms of certified accuracy for SecureSGD (78.74%, from 86.74% to 7.99%) and SecureSGD-AGM (81.97%, from 87.23% to 5.26%) on average. This is promising.

Results on the CIFAR-10 Dataset Results on the CIFAR-10 dataset further strengthen our observations. In Figure 3.8, our StoBatch clearly outperforms baseline models in all cases ($p < 6.17e - 9$), especially when the privacy budget is small ($\epsilon < 4$), yielding strong privacy protections. On average conventional accuracy, our StoBatch mechanism has an improvement of 10.42% over SecureSGD ($p < 2.59e - 7$), an improvement of 14.08% over SecureSGD-AGM ($p < 5.03e - 9$), an improvement of 29.22% over AdLM ($p < 5.28e - 26$), and a 14.62% improvement over DP-SGD ($p < 4.31e - 9$). When the privacy budget is increased from 2 to 10, the conventional accuracy of our StoBatch model increases from 42.02% to 46.76%, showing a 4.74% improvement on average. However, the conventional accuracy of our model under adversarial example attacks is still low, i.e., 44.22% on average given the privacy budget at 2.0. This opens a long-term research avenue to achieve better robustness under strong privacy guarantees in adversarial learning.

The accuracy of our model is consistent given different attacks with different adversarial perturbations μ_a under a rigorous DP protection ($\epsilon = 2.0$), compared with baseline approaches (Figure 3.10). In fact, when the attack size μ_a increases from 0.05 to 0.5, the conventional accuracies of the baseline approaches are remarkably reduced, i.e., a drop of 25.26% on average given the most effective baseline approach, SecureSGD. Meanwhile, there is a much smaller degradation (4.79% on average) in terms of the conventional accuracy observed in our StoBatch model. Our model also achieves better accuracies compared with baseline approaches in all cases ($p < 8.2e - 10$). Figure 3.12 further shows that our StoBatch model is more accurate than baseline approaches (i.e., ϵ_r is set to 0.1 in PixelDP) in terms of certified accuracy in all cases, with a tight privacy budget set to 2.0 ($p < 2.04e - 18$). We register an improvement of 21.01% in our StoBatch model given the certified accuracy over SecureSGD model, which is the most effective baseline approach ($p < 2.04e - 18$).

Scalability under Strong Iterative Attacks First, we scale our model in terms of *adversarial training* in the CIFAR-10 dataset, in which the number of iterative attack steps is increased from $T_\mu = 3$ to T_μ *= 200 in training*, and up to T_a *= 2,000 in testing*. Note that the traditional iterative batch-by-batch DP adversarial training (Algorithm 3.3) is nearly infeasible in this setting, taking over 30 days for one training with 600 epochs. Thanks to the parallel and distributed training, our StoBatch only takes $\approxeq$ 3 days to finish the training. More importantly, our StoBatch achieves consistent conventional and certified accuracies under strong iterative attacks with $T_a = 1,000$, compared with the best baseline, i.e., SecureSGD (Figure 3.13). Across attack sizes $\mu_a \in \{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ and steps $T_a \in \{100, 500, 1000, 2000\}$, on average, our StoBatch achieves 44.87±1.8% and 42.18±1.8% in conventional and certified accuracies, compared with 29.47±12.5% and 20±6.1% of SecureSGD ($p < 1.05e - 9$).

We achieve a similar improvement over the **Tiny ImageNet**, i.e., following [192], with a ResNet18 model, i.e., *a larger dataset on a larger network* (Figure 3.14). On average, across attack sizes $\mu_a \in \{0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$ and steps $T_a \in \{100, 500, 1000, 2000\}$, our StoBatch achieves 29.78±4.8% and 28.31±1.58% in conventional and certified accuracies, compared with 8.99±5.95% and 8.72±5.5% of SecureSGD ($p < 1.55e - 42$).

Key observations **(1)** Incorporating ensemble adversarial learning into DP preservation, tightened sensitivity bounds, a random perturbation size μ_t at each training step, and composition robustness bounds in both input and latent spaces does enhance the consistency, robustness, and accuracy of DP model against different attacks with different levels of perturbations. These are key advantages of our mechanism; **(2)** As a result, our StoBatch model outperforms baseline algorithms, in terms of conventional and certified accuracies in most of the cases. It is clear that existing DP-preserving approaches have not been designed to withstand against adversarial examples; and **(3)** Our StoBatch training can help us to scale our mechanism to larger DP DNNs and datasets with distributed adversarial learning, without affecting the model accuracies and DP protections.

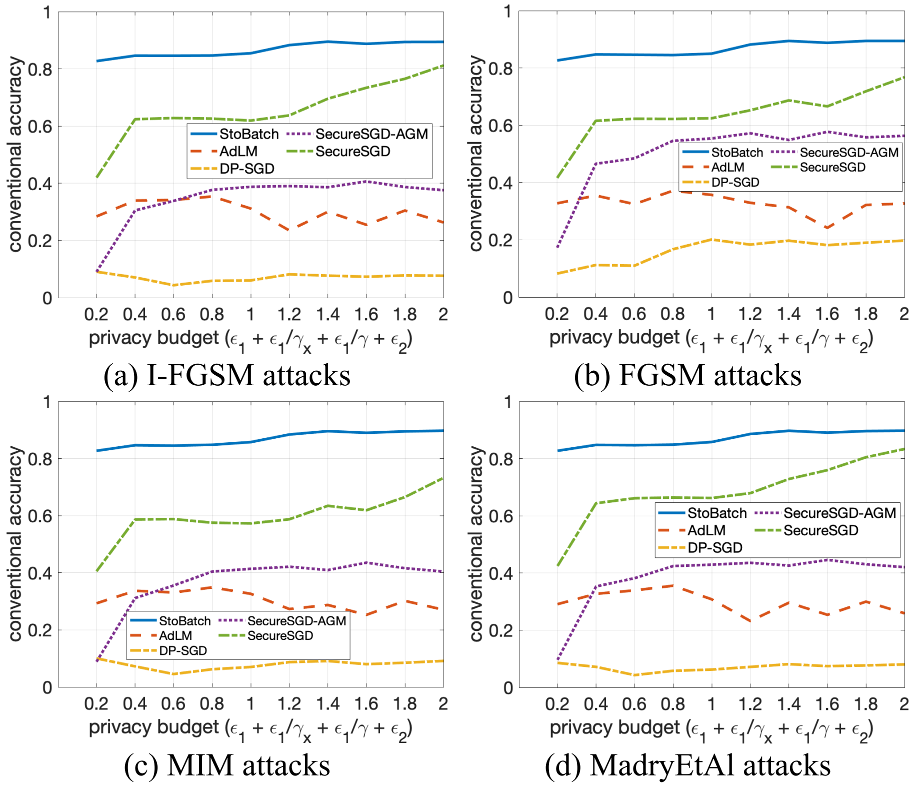
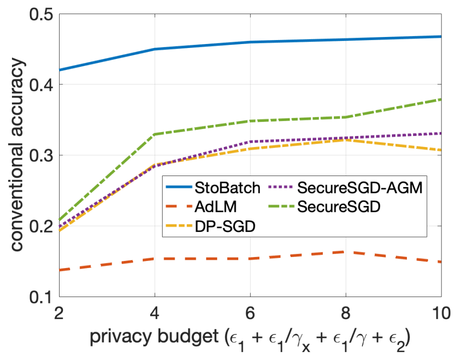
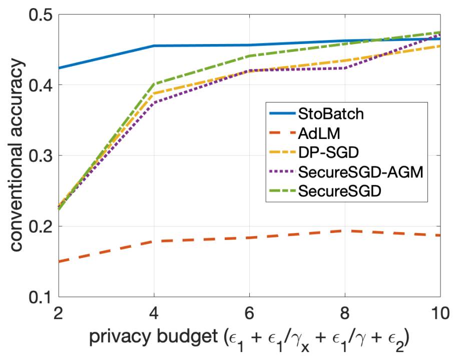


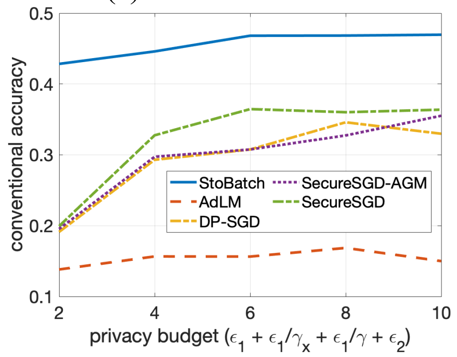
Figure 3.7 Conventional accuracy on the MNIST dataset given ϵ, under $l_\infty(\mu_a = 0.2)$ and $T_a = 10$.



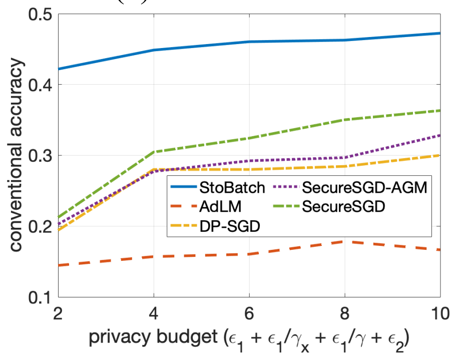
(a) I-FGSM attacks



(b) FGSM attacks

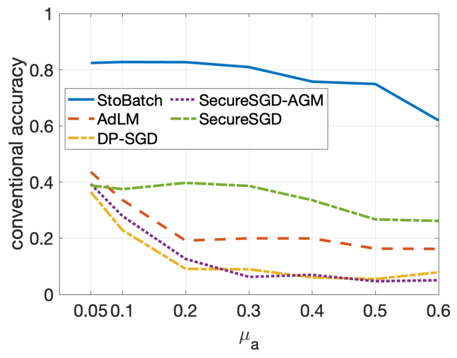


(c) MIM attacks

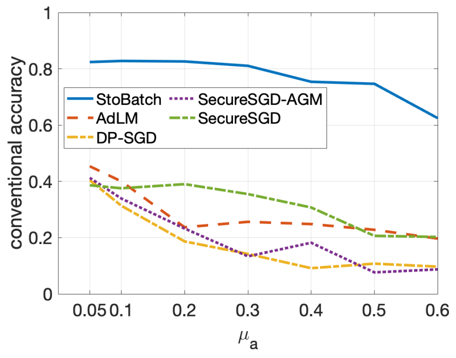


(d) MadryEtAl attacks

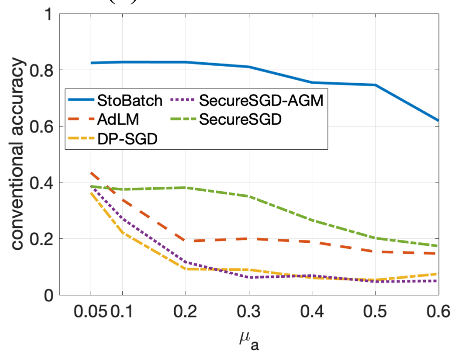
Figure 3.8 Conventional accuracy on the CIFAR-10 dataset given ϵ, under $l_\infty(\mu_a = 0.2)$ and $T_a = 3$.



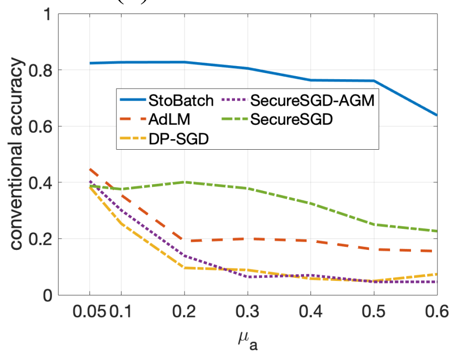
(a) I-FGSM attacks



(b) FGSM attacks

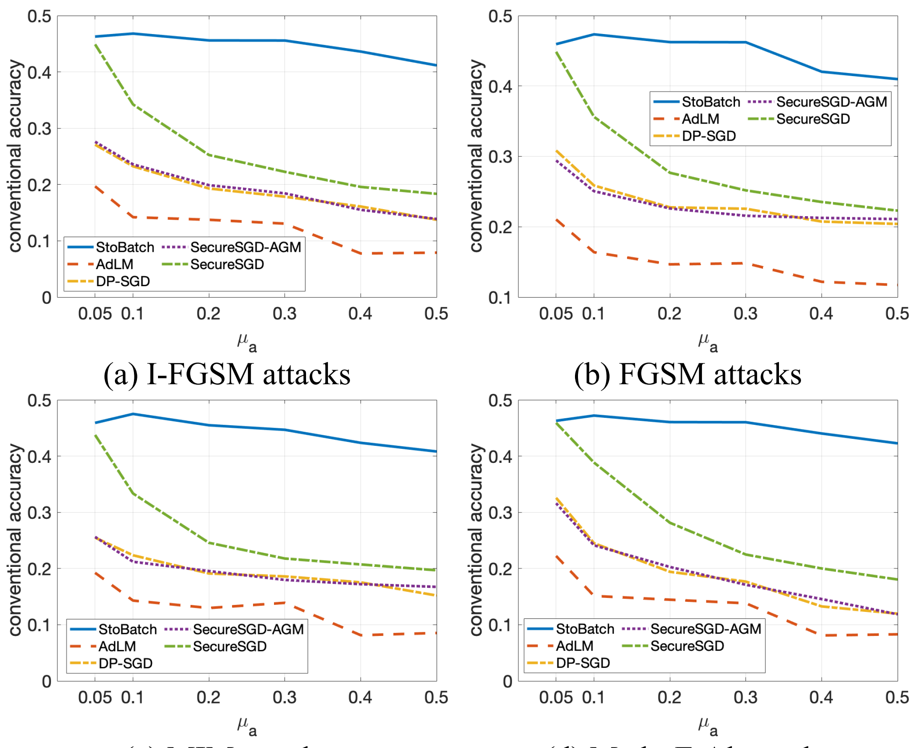


(c) MIM attacks



(d) MadryEtAl attacks

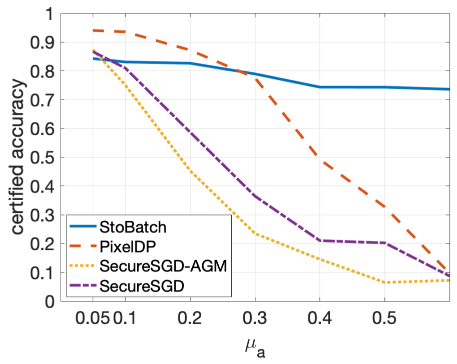
Figure 3.9 Conventional accuracy on the MNIST dataset given μ_a ($\epsilon = 0.2$, tight DP protection) and $T_a = 10$.



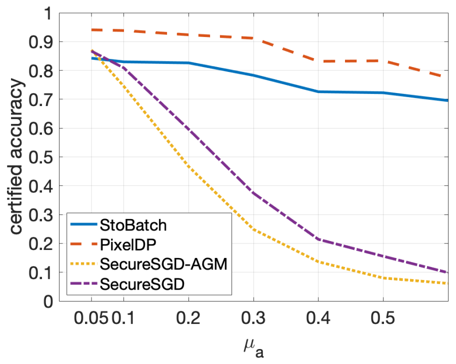
(a) I-FGSM attacks (b) FGSM attacks

(c) MIM attacks (d) MadryEtAl attacks

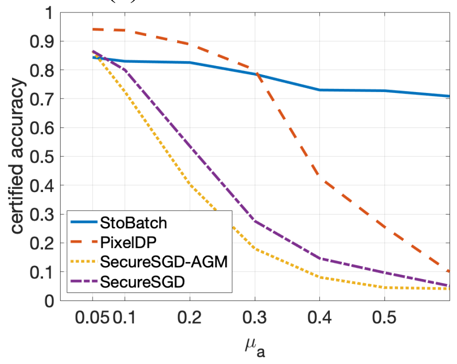
Figure 3.10 Conventional accuracy on the CIFAR-10 dataset given μ_a ($\epsilon = 2$, tight DP protection) and $T_a = 3$.



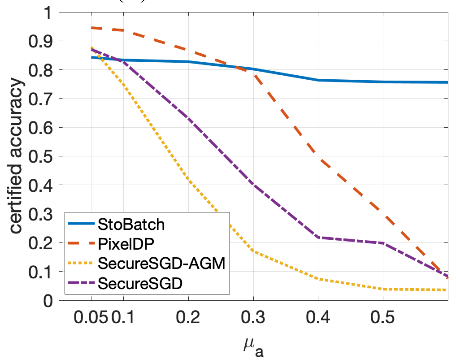
(a) I-FGSM attacks



(b) FGSM attacks

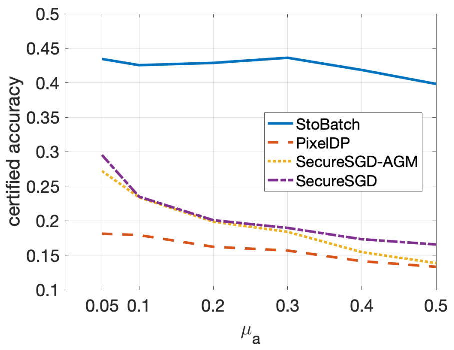


(c) MIM attacks

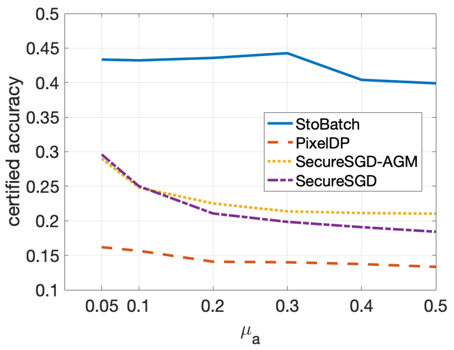


(d) MadryEtAl attacks

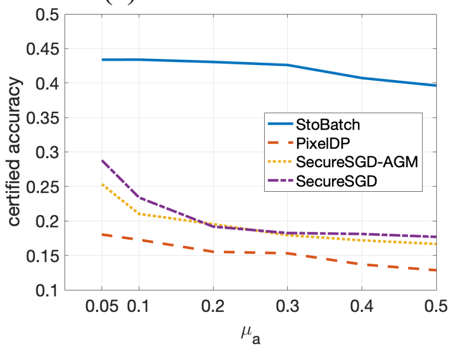
Figure 3.11 Certified accuracy on the MNIST dataset. ϵ is set to 1.0 (tight DP protection) and $T_a = 10$.



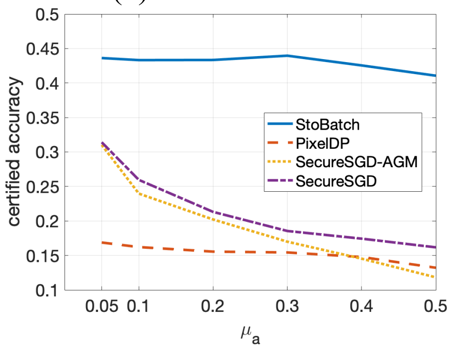
(a) I-FGSM attacks



(b) FGSM attacks

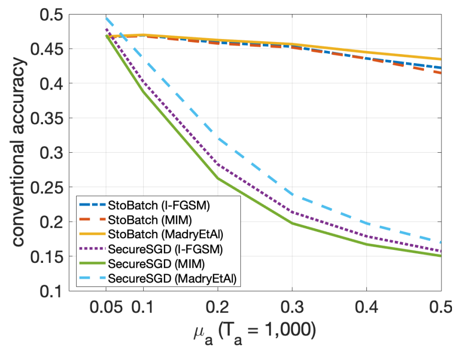


(c) MIM attacks

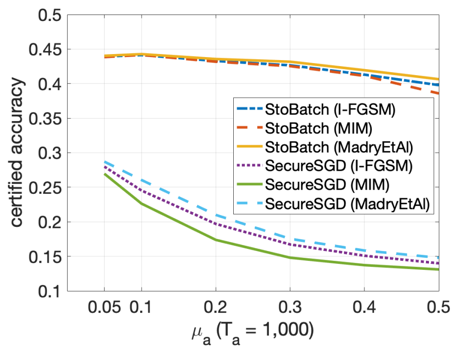


(d) MadryEtAl attacks

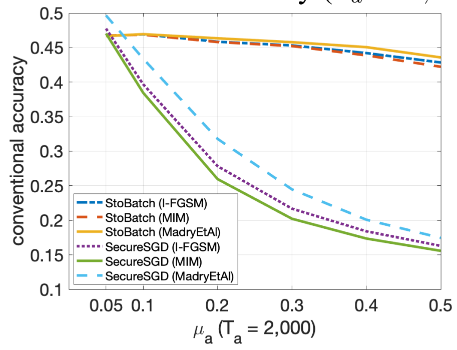
Figure 3.12 Certified accuracy on the CIFAR-10 dataset. ϵ is set to 2 (tight DP protection) and and $T_a = 3$.



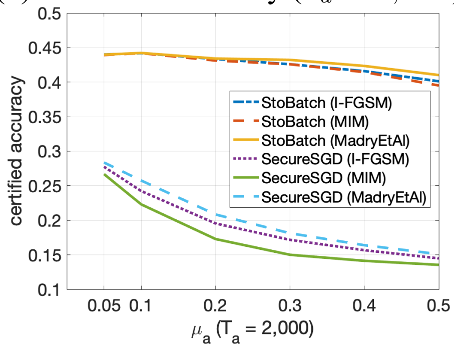
(a) Conventional Accuracy ($T_a = 1,000$)



(b) Certified Accuracy ($T_a = 1,000$)

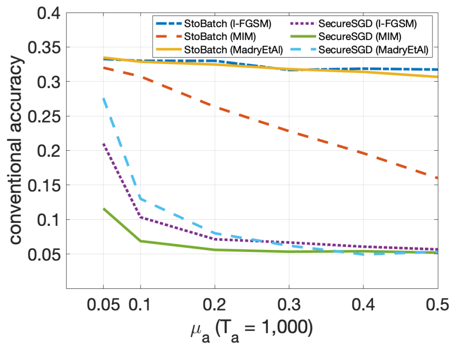


(c) Conventional Accuracy ($T_a = 2,000$)

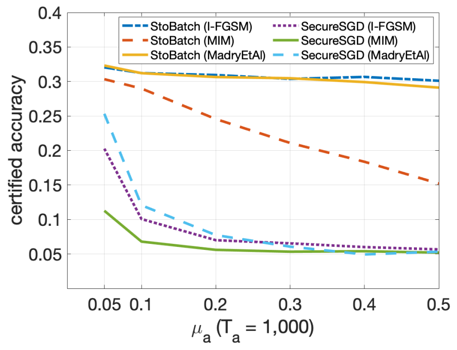


(d) Certified Accuracy ($T_a = 2,000$)

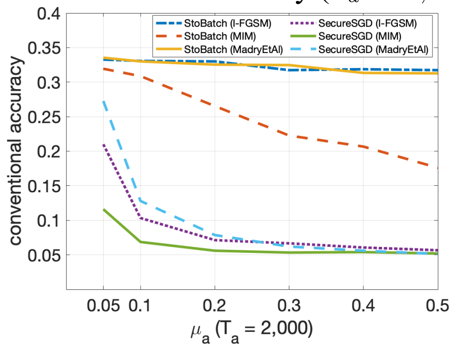
Figure 3.13 Accuracy on the CIFAR-10 dataset, under Strong Iterative Attacks ($T_a = 1,000; 2,000$). ϵ is set to 2 (tight DP protection).



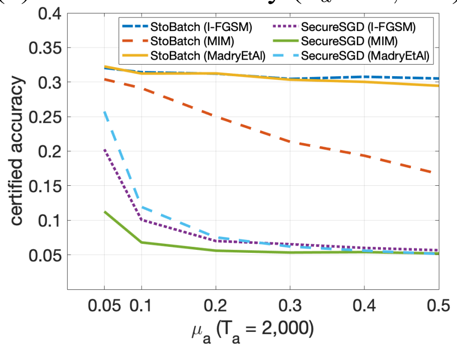
(a) Conventional Accuracy ($T_a = 1,000$)



(b) Certified Accuracy ($T_a = 1,000$)



(c) Conventional Accuracy ($T_a = 2,000$)



(d) Certified Accuracy ($T_a = 2,000$)

Figure 3.14 Accuracy on the Tiny ImageNet dataset, under Strong Iterative Attacks ($T_a = 1,000; 2,000$). ϵ is set to 5.

3.2.10 Conclusion

In this part of the dissertation, we established a connection among DP preservation to protect the training data, adversarial learning, and certified robustness. A sequential composition robustness was introduced to generalize robustness given any sequential and bounded function of independent defensive mechanisms in both input and latent spaces. We addressed the trade-off among model utility, privacy loss, and robustness by tightening the global sensitivity bounds. We further developed a stochastic batch training mechanism to bypass the vanilla iterative batch-by-batch training in DP DNNs. The average errors of our approximation functions are always bounded by constant values. Last but not least, a new Monte Carlo Estimation was proposed to stabilize the estimation of the robustness bounds. Rigorous experiments conducted on benchmark datasets shown that our mechanism significantly enhances the robustness and scalability of DP DNNs. In future work, we will test our algorithms and models in the Baidu Fedcube platform [193]. In addition, we will evaluate our robustness bounds against synergistic attacks, in which adversarial examples can be combined with other attacks, such as Trojans [194, 188], to create more lethal and stealthier threats [195].

3.3 Consistently Bounded Differential Privacy in Lifelong Learning

3.3.1 Background

Let us first revisit L2M with A-gem and DP. In L2M, we learn a sequence of tasks $\mathbf{T} = \{t_1, \ldots, t_m\}$ one by one, such that the learning of each new task will not forget the models learned for the previous tasks. Let D_i is the dataset of the i-th task. Each tuple contains data $x \in [-1, 1]^d$ and a ground-truth label $y \in \mathbb{Z}_K$, which is a one-hot vector of K categorical outcomes $y = \{y_1, \ldots, y_K\}$. A single true class label $y_x \in y$ given x is assigned to only one of the K categories. All the training sets D_i are non-overlapping; that is, an arbitrary input (x, y) belongs to only one D_i, i.e., $\exists! i \in [1, m] : (x, y) \in D_i$ ($x \in D_i$ for simplicity). On input x and parameters θ, a model outputs class scores $f : \mathbb{R}^d \rightarrow \mathbb{R}^K$ that map inputs

x to a vector of scores $f(x) = \{f_1(x), \dots, f_K(x)\}$ s.t. $\forall k \in [1, K] : f_k(x) \in [0, 1]$ and $\sum_{k=1}^{K} f_k(x) = 1$. The class with the highest score is selected as the predicted label for x, denoted as $y(x) = \max_{k \in K} f_k(x)$. A loss function $L(f(\theta, x), y)$ presents the penalty for mismatching between the predicted values $f(\theta, x)$ and original values y.

Lifelong Learning Given the current task τ $(\leq m)$, let us denote $\mathbb{T}_\tau = \{t_1, \dots, t_{\tau-1}\}$ is a set of tasks that have been learnt. Although there are different L2M settings, i.e., episodic memory [196, 197, 89, 90, 91, 92, 198] and generative memory [93, 94, 95], we leverage one of the state-of-the-art algorithms, i.e., A-gem [54], to demonstrate our privacy preserving mechanism, without loss of the generality of our study. A-gem avoids catastrophic forgetting by storing an episodic memory M_i for each task $t_i \in \mathbb{T}_\tau$. When minimizing the loss on the current task τ, a typical approach is to treat the losses on the episodic memories of tasks $i < \tau$, given by $L(f(\theta, M_i)) = \frac{1}{|M_i|} \sum_{x \in M_i} L(f(\theta, x), y)$, as inequality constraints. In A-gem, the L2M objective function is:

$$\begin{aligned} \theta^\tau &= \arg\min_\theta L\big(f(\theta^{\tau-1}, D_\tau)\big) \\ \textbf{s.t. } & L\big(f(\theta^\tau, \mathbb{M}_\tau)\big) \leq L\big(f(\theta^{\tau-1}, \mathbb{M}_\tau)\big) \end{aligned} \tag{3.76}$$

where $\mathbb{M}_\tau = \cup_{i<\tau} M_i$ is the episodic memory with $\mathbb{M}_1 = \emptyset$, $L\big(f(\theta^{\tau-1}, \mathbb{M}_\tau)\big) = \sum_{i=1}^{\tau-1} L\big(f(\theta^{\tau-1}, M_i)\big)/(\tau - 1)$, $\theta^{\tau-1}$ is the values of model parameters θ learned after training the task $t_{\tau-1}$, indicating that the model will not forget previously learned tasks $\{t_1, \dots, t_{\tau-1}\}$, given the memory replaying constraint $L\big(f(\theta^\tau, \mathbb{M}_\tau)\big) \leq L\big(f(\theta^{\tau-1}, \mathbb{M}_\tau)\big)$.

At each training step, A-gem [54] has access to only D_τ and $\mathbb{M}_\tau$ to compute the *projected gradient* $\tilde{g}$ (i.e., by addressing the constraint in Equation (3.76)), as follows:

$$\tilde{g} = g - \frac{g^\top g_{ref}}{g_{ref}^\top g_{ref}} g_{ref} \tag{3.77}$$

where g is the *updated gradient* computed on a batch sampled from D_τ, g_{ref} is an *episodic gradient* computed on a batch sampled from $\mathbb{M}_\tau$, and $\tilde{g}$ is used to update the model parameters θ in Equation (3.76).

Differential Privacy guarantees that the released statistical results, computed from the underlying sensitive data, will be insensitive to the presence or absence of one record in a dataset. Let us briefly revisit the definition of DP, as follows:

Definition 3.3. *(ϵ, δ)-DP [55]. A randomized algorithm A is (ϵ, δ)-DP, if for any two neighboring databases D and D' differing at most one tuple, and $\forall O \subseteq Range(A)$, we have:*

$$Pr[A(D) = O] \leq e^{\epsilon} Pr[A(D') = O] + \delta \tag{3.78}$$

where ϵ controls the amount by which the distributions induced by D and D' may differ, and δ is a broken probability. A smaller ϵ enforces a stronger privacy guarantee.

DP also applies to general metrics $\rho(D, D') \leq 1$, where ρ can be a l_p-norms [165]. DP-preserving algorithms in deep learning can be categorized into three lines: **(1)** Introducing noise into parameter gradients [62, 166, 59, 73, 74, 76] for streaming data [199] and Q-learning [97]; **(2)** Injecting noise into objective functions [60, 71, 72, 96]; and **(3)** Injecting noise into labels [167]. In [200], local DP is used to maintain up-to-date data statistics over time. These existing mechanisms have not been designed to preserve DP in L2M. That is different from our goal in this study.

3.3.2 Privacy Risk & Problem Statement

In this subsection, we focus on analyzing the unknown privacy risk in L2M and introduce a new concept of Lifelong DP.

Privacy Risk Analysis One benefit of L2M is that end-users can use an L2M model after training each task τ, instead of waiting for the model to be trained on all the tasks. Thus, in practice, the adversary can observe the model parameters $\theta^1, \ldots, \theta^m$ after training each task $t_1, \ldots, t_m$. Note that the adversary does not observe any information about the (black-box) training algorithm. Another key property in an L2M model is the episodic memory, which is kept to be read at each training step incurring privacy leakage. Therefore, the training data D and episodic memory M need to be protected together across tasks. Finally, in L2M, at each training step for any task t_i ($i \in [1, m]$), we only have access to D_i and $\mathbb{M}_i$, without a complete view of the cumulative dataset of all the tasks $\cup_{i\in[1,m]} D_i$ and $\mathbb{M}_m = \cup_{i\in[1,m-1]} M_i$. This is fundamentally different from the traditional definition of a database in both DP (**Def. 3.3**) and in a model trained on a single task. To cope with this, we propose a new definition of lifelong neighboring databases, as follows:

Definition 3.4. *Lifelong Neighboring Databases. Given any two lifelong databases* $\mathsf{data}_m = \{\mathcal{D}, \mathcal{M}\}$ *and* $\mathsf{data}'_m = \{\mathcal{D}', \mathcal{M}'\}$*, where* $\mathcal{D} = \{D_1, \ldots, D_m\}$*,* $\mathcal{D}' = \{D'_1, \ldots, D'_m\}$*,* $\mathcal{M} = \{\mathbb{M}_1, \ldots, \mathbb{M}_m\}$*,* $\mathcal{M}' = \{\mathbb{M}'_1, \ldots, \mathbb{M}'_m\}$*,* $\mathbb{M}_i = \cup_{j\in[1,i-1]} M_j$*, and* $\mathbb{M}'_i = \cup_{j\in[1,i-1]} M'_j$*.* data_m *and* data'_m *are called lifelong neighboring databases if,* $\forall i \in [1, m]$*:* ***(1)*** D_i *and* D'_i *differ at most one tuple; and* ***(2)*** M_i *and* M'_i *differ at most one tuple.*

A Naive Mechanism Given the aforementioned properties and **Def. 3.4**, to preserve DP in L2M, one can employ the well-applied moment accountant in [62] to train the model f by injecting Gaussian noise into parameter gradients g and g_{ref} in Equation (3.77), with a privacy budget ϵ_{D_τ} on each dataset D_τ and $\epsilon_{\mathbb{M}_\tau}$ on the episodic memory $\mathbb{M}_\tau$. The post-processing property in DP [100] can be applied to guarantee that $\tilde{g}$, computed from the perturbed g and g_{ref}, is also DP. Let us denote this mechanism as A, and A_τ is used to denote A applied on the task τ. A naive approach is to repeatedly apply A on the sequence of tasks **T**, as follows:

$$\theta^{\tau} = \arg\min_{\theta} A_{\tau} L\big(f(\theta^{\tau-1}, D_{\tau})\big)$$
$$\textbf{s.t. } L\big(f(\theta^{\tau}, \mathbb{M}_{\tau})\big) \leq L\big(f(\theta^{\tau-1}, \mathbb{M}_{\tau})\big) \tag{3.79}$$

Since training data is non-overlapping among tasks, the parallel composition property in DP [61] can be applied to estimate the total privacy budget consumed across all the tasks:

$$Pr[A(\mathsf{data}_m) = \{\theta^i\}_{i\in[1,m]}] \leq e^{\epsilon} Pr[A(\mathsf{data}'_m) = \{\theta^i\}_{i\in[1,m]}] + \delta \tag{3.80}$$

where $\epsilon = \max_{i\in[1,m]}(\epsilon_{D_i} + \epsilon_{\mathbb{M}_i})$, and $\forall i, j \in [1, m] : \delta$ is the same for ϵ_{D_i} and $\epsilon_{\mathbb{M}_j}$.

$A(\mathsf{data}_m)$ indicates that the model is trained from scratch with the mechanism A, given randomly initiated parameters θ^0, i.e., $A(\theta^0, \mathsf{data}_m)$. Intuitively, we can achieve the traditional DP guarantee in L2M, as the participation of a particular data tuple in each dataset D_{τ} is protected under the released (ϵ, δ)-DP $\{\theta^i\}_{i\in[1,m]}$. However, this approach introduces unknown privacy risks in each task and in the whole training process, as discussed next.

At each task, the parallel composition property is not sufficient to ensure that the privacy budget will not be accumulated across tasks (**Theorem 3.7**). Together with lacking of a complete view of the cumulative data of all the tasks $\cup_{i\in[1,m]} D_i$ and $\mathbb{M}_m = \cup_{i\in[1,m-1]} M_i$, observing the intermediate parameters $\{\theta^i\}_{i<\tau}$ turns the mechanism A_{τ} into a list of adaptive DP mechanisms $A_1, \ldots, A_{\tau}$ sequentially applied on tasks $t_1, \ldots, t_{\tau}$, where $A_i : (\prod_{j=1}^{i-1} \mathcal{R}_j) \times D_i \rightarrow \mathcal{R}_i$. This is an instance of adaptive composition, which we can model by using the output of all the previous mechanisms $\{\theta^i\}_{i<\tau}$ as the auxiliary input of the A_{τ} mechanism. Therefore, given an outcome $\theta^{\tau} \in \mathcal{R}_{\tau}$, the privacy loss $c(\cdot)$ at θ^{τ} can be measured as follows: $c(\theta^{\tau}; A_{\tau}, \{\theta^i\}_{i<\tau}, \mathsf{data}_{\tau}, \mathsf{data}'_{\tau}) = \log \frac{Pr[A_{\tau}(\{\theta^i\}_{i<\tau}, \mathsf{data}_{\tau}) = \theta^{\tau}]}{Pr[A_{\tau}(\{\theta^i\}_{i<\tau}, \mathsf{data}'_{\tau}) = \theta^{\tau}]}$

The following theorem shows that: $\forall \tau \in [1, m]$, the privacy loss is the sum of the privacy loss consumed in previous tasks.

Theorem 3.7. *Let the privacy loss be defined as above. Then, we have that:* $\forall \tau > 1$: $c(\theta^\tau; A_\tau, \{\theta^i\}_{i<\tau}, \mathsf{data}_\tau, \mathsf{data}'_\tau) = \sum_{i=1}^{\tau} c(\theta^i; A_i, \{\theta^j\}_{j<i}, \mathsf{data}_i, \mathsf{data}'_i)$.

Proof of Theorem 3.7

Proof. Let us denote $A_{1:i}$ as $A_1, \ldots, A_i$, we have:

$$\begin{aligned}
&c(\theta^\tau; A_\tau, \{\theta^i\}_{i<\tau}, \mathsf{data}_\tau, \mathsf{data}'_\tau) = \log \frac{Pr[A_\tau(\{\theta^i\}_{i<\tau}, \mathsf{data}_\tau) = \theta^\tau]}{Pr[A_\tau(\{\theta^i\}_{i<\tau}, \mathsf{data}'_\tau) = \theta^\tau]} \\
&= \log \prod_{i=1}^{\tau} \frac{Pr[A_i(\theta^{i-1}, \mathsf{data}_i) = \theta^i | A_{1:i-1}(\{\theta^j\}_{j<i-1}, \mathsf{data}_{1:i-1}) = \theta^{1:i-1}]}{Pr[A_i(\theta^{i-1}, \mathsf{data}'_i) = \theta^i | A_{1:i-1}(\{\theta^j\}_{j<i-1}, \mathsf{data}'_{1:i-1}) = \theta^{1:i-1}]} \\
&= \sum_{i=1}^{\tau} \log \frac{Pr[A_i(\theta^{i-1}, \mathsf{data}_i) = \theta^i | A_{1:i-1}(\{\theta^j\}_{j<i-1}, \mathsf{data}_{1:i-1}) = \theta^{1:i-1}]}{Pr[A_i(\theta^{i-1}, \mathsf{data}'_i) = \theta^i | A_{1:i-1}(\{\theta^j\}_{j<i-1}, \mathsf{data}'_{1:i-1}) = \theta^{1:i-1}]} \\
&= \sum_{i=1}^{\tau} c(\theta^i; A_i, \{\theta^j\}_{j<i}, \mathsf{data}_i, \mathsf{data}'_i)
\end{aligned}$$

Consequently, Theorem 3.7 does hold. □

As a result of the Theorem 3.7, the privacy budget at each task τ cannot be simply bounded by $\max_{\tau\in[1,m]}(\epsilon_{D_\tau} + \epsilon_{\mathbb{M}_\tau})$, given δ (Equation (3.80)). This problem might be addressed by replacing the max function in Equation (3.80) with a summation function: $\epsilon = \sum_{\tau\in[1,m]}(\epsilon_{D_\tau} + \epsilon_{\mathbb{M}_\tau})$, to compute the upper bound of the privacy budget for an entire of the continual learning process. However, the challenging issues in bounding the privacy risk is still the same, centering around *the growing number of tasks* m, *information disclosure via the episodic memory*, and *the heterogeneity among tasks* for the following reasons. **(1)** The larger the number of tasks, the larger the privacy budget will be (proportionally) consumed by the $\sum$ function. **(2)** More critically, memorizing previous tasks will further disclose information about the data in the past, since g_{ref} (Equation (3.77)) is computed using a batch randomly sampled from the episodic memory consisting of data from previous tasks, i.e., $\mathbb{M}_\tau = \cup_{i<\tau} M_i$. Continuing to access the episodic memory intensifies the privacy risk over time. **(3)** The growing of the episodic memory $\mathbb{M}$ by adding new tuples $M_{\tau-1}$ selected from $D_{\tau-1}$ after training on each task $\tau - 1$ makes it more challenging to bound the privacy budget ϵ (Equation (3.80)). In fact, given that $\forall i \in [1, \tau-1] : M_i$ and M'_i differs at most one tuple, $\mathbb{M}_\tau$ and $\mathbb{M}'_\tau$ will differ at most $\tau - 1$ tuples causing additional privacy leakage. Also, the data sampling probability to compute g_{ref} is affected by the increasing size of $\mathbb{M}$ [62]. **(4)** Different tasks may require different numbers of training steps due to the difference in terms of the number of tuples in each task; thus, affecting the privacy budget ϵ. **(5)** The order of training tasks also affect the privacy budget, since computing g_{ref} by using data in the episodic memory from one task may be more than other tasks. Therefore, bounding the DP budget in L2M is non-trivial.

Lifelong Differential Privacy To address these challenges, we propose a new definition of (ϵ, δ)-**Lifelong DP** to guarantee that an adversary cannot infer whether a data tuple is in the lifelong training dataset $\mathtt{data}_m$, given the released parameters $\{\theta^i\}_{i\in[1,m]}$ learned from a growing stream of an infinite number of new tasks, denoted $\forall m \in [1, \infty)$, under a consistently bounded DP budget (ϵ, δ) (**Equation (3.81)**). A consistently bounded DP means having only one fixed value of the privacy budget (ϵ, δ), regardless the number of tasks m. In other words, if there exists an $i \leq m$ and an $\epsilon' < \epsilon$, such that releasing $\{\theta^j\}_{j\in[1,i]}$ given training dataset $\mathtt{data}_i$ is (ϵ', δ)-DP, then (ϵ, δ) is NOT a consistently bounded DP budget, since it weakens the previously existed protection (ϵ', δ) at the task i (**Equation (3.82)**). A consistently bounded DP is significant in practice, by enabling us to keep training an L2M model and releasing its parameters, without intensifying the end-to-end privacy budget consumption. Our Lifelong DP can be formulated as:

Definition 3.5. (ϵ, δ)*-Lifelong DP. Given a lifelong database* $\mathsf{data}_m = \{\mathcal{D}, \mathcal{M}\}$*, where* $\mathcal{D} = \{D_1, \ldots, D_m\}$ *and* $\mathcal{M} = \{M_1, \ldots, M_m\}$*, a randomized algorithm* A *achieves* (ϵ, δ)*-Lifelong DP, if for any of two lifelong neighboring databases* $(\mathsf{data}_m, \mathsf{data}'_m)$*,* $\forall m \in [1, \infty)$ *we have that*

$$
\begin{aligned}
& Pr\big[A(\mathsf{data}_m) = \{\theta^i\}_{i\in[1,m]}\big] \\
& \qquad \leq e^{\epsilon} Pr\big[A(\mathsf{data}'_m) = \{\theta^i\}_{i\in[1,m]}\big] + \delta \qquad (3.81) \\
& \nexists(\epsilon' < \epsilon, i \leq m) : Pr\big[A(\mathsf{data}_i) = \{\theta^j\}_{j\in[1,i]}\big] \\
& \qquad \leq e^{\epsilon'} Pr\big[A(\mathsf{data}'_i) = \{\theta^j\}_{j\in[1,i]}\big] + \delta \qquad (3.82)
\end{aligned}
$$

3.3.3 Preserving Lifelong DP

To preserve Lifelong DP, we address the following problems: **(1)** The privacy loss accumulation across tasks; **(2)** The overlapping between the episodic memory $\mathcal{M}$ and the training data $\mathcal{D}$; and **(3)** The data sampling process for computing g_{ref} given the growing $\mathcal{M}$. Our network is a multi-layer neural network stacked on top of a feature representation learning model. Then, we design a new Lifelong DP preservation algorithm, called **L2DP-ML (Algorithm 3.6)**, in computing the gradients g, g_{ref}, and $\tilde{g}$ (**Equations (3.83) and (3.87)**). To overcome an expensive computation cost, we develop a scalable and heterogeneous algorithm through a streaming batch training **(Algorithm 3.7)**, to efficiently learn Lifelong DP parameters (**Theorem 3.8**).

Network Design In our **Algorithm 3.6**, a DNN is designed as $f(x) = \mathcal{G}(a(x, \theta_1), \theta_2)$, where $a(x, \theta_1)$ is a feature representation learning model, i.e., an auto-encoder, with x as an input, and a typical multi-layer neural network $\mathcal{G}(\cdot, \theta_2)$, i.e., a CNN, taking the output of $a(x, \theta_1)$ and returning the class scores $f(x)$. Given a dataset D_τ, the objective functions of $a(\cdot)$ and $\mathcal{G}(\cdot)$ can be the classical cross-entropy error functions for data reconstruction $\mathcal{R}_{D_\tau}(\theta_1)$ at the input layer and for classification $\mathcal{L}_{D_\tau}(\theta_2)$ at the output layer.

This network design allows us to: **(1)** Tighten the sensitivity of our model, since it is easy to train $a(\cdot)$ using less sensitive objective functions, given its small sizes; **(2)** Reduce the privacy budget consumption, since the computations of $\mathcal{G}(\cdot)$ automatically is DP when the output of $a(x, \theta_1)$ is DP; and **(3)** Provide a better re-usability, given that $a(\cdot)$ can be reused and shared for different predictive models. For instance, $\mathcal{R}_{D_\tau}(\theta_1)$ can be presented as follows:

$$\mathcal{R}_{D_\tau}(\theta_1) = \sum_{x_r \in D_\tau} \sum_{s=1}^{d} \Big[x_{rs} \log(1 + e^{-\theta_{1s} h_r}) + (1 - x_{rs}) \log(1 + e^{\theta_{1s} h_r}) \Big]$$

where the transformation of x_r is $h_r = \theta_1^\top x_r$, the hidden layer $\mathbf{h}_1$ of $a(x, \theta_1)$ given D_τ is $\mathbf{h}_{1D_\tau} = \{\theta_1^\top x_r\}_{x_r \in D_\tau}$, and $\widetilde{x}_r = \theta_1 h_r$ is the reconstruction of x_r. Our L2M objective function is defined as:

$$\begin{aligned} \{\theta_1^\tau, \theta_2^\tau\} &= \arg\min_{\theta_1, \theta_2} [\mathcal{R}_{D_\tau}(\theta_1^{\tau-1}) + \mathcal{L}_{D_\tau}(\theta_2^{\tau-1})] \\ &\text{s.t. } \mathcal{R}_{\mathbb{M}_\tau}(\theta_1^\tau) \leq \mathcal{R}_{\mathbb{M}_\tau}(\theta_1^{\tau-1}) \text{ and } \mathcal{L}_{\mathbb{M}_\tau}(\theta_2^\tau) \leq \mathcal{L}_{\mathbb{M}_\tau}(\theta_2^{\tau-1}) \end{aligned} \tag{3.83}$$

where $\{\theta_1, \theta_2\}$ are the model parameters; while, $\{\theta_1^\tau, \theta_2^\tau\}$ are used to indicate the values of $\{\theta_1, \theta_2\}$ after learning task τ.

Algorithm 3.6 Lifelong DP - Machine Learning (L2DP-ML)

```
Input: T=$\{t_i\}_{i\in[1,m]}$, $\{D_i\}_{i\in[1,m]}$, $\epsilon_1, \epsilon_2$
 1: Draw Noise $\chi_1 \leftarrow [Lap(\frac{\Delta_{\widetilde{\mathcal{R}}}}{\epsilon_1})]^d$, $\chi_2 \leftarrow [Lap(\frac{\Delta_{\widetilde{\mathcal{R}}}}{\epsilon_1})]^\beta$, $\chi_3 \leftarrow [Lap(\frac{\Delta_{\widetilde{\mathcal{L}}}}{\epsilon_2})]^{|\mathbf{h}_\pi|}$
 2: Randomly Initialize: $\theta^0 = \{\theta_1^0, \theta_2^0\}$, $\mathbb{M}_1 = \emptyset$, $\forall \tau \in \mathbf{T}: \overline{D}_\tau = \{\overline{x}_r \leftarrow x_r + \frac{\chi_1}{|D_\tau|}\}_{x_r\in D_\tau}$,
    hidden layers $\{\mathbf{h}_1 + \frac{2\chi_2}{|D_\tau|}, \ldots, \mathbf{h}_\pi\}$
 3: for $\tau \in [1, m]$ do
 4:     if $\tau == 1$ then
 5:         Compute $g \leftarrow \{\nabla_{\theta_1}\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1}), \nabla_{\theta_2}\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})\}$ with the noise $\frac{\chi_3}{|D_\tau|}$
 6:     else
 7:         $\mathbb{M}_\tau \leftarrow \mathbb{M}_{\tau-1} \cup \{\overline{D}_{\tau-1}\}$
 8:         Randomly Pick a dataset $\overline{D}_{ref} \in \mathbb{M}_\tau$
 9:         Compute Gradients:
10:         $g \leftarrow \{\nabla_{\theta_1}\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1}), \nabla_{\theta_2}\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})\}$ with the noise $\frac{\chi_3}{|D_\tau|}$
11:         $g_{ref} \leftarrow \{\nabla_{\theta_1}\overline{\mathcal{R}}_{\overline{D}_{ref}}(\theta_1^{\tau-1}), \nabla_{\theta_2}\overline{\mathcal{L}}_{\overline{D}_{ref}}(\theta_2^{\tau-1})\}$ with the noise $\frac{\chi_3}{|D_{ref}|}$
12:         $\tilde{g} \leftarrow g - \frac{g^\top g_{ref}}{g_{ref}^\top g_{ref}} g_{ref}$
13:     Descent: $\{\theta_1^\tau, \theta_2^\tau\} \leftarrow \{\theta_1^{\tau-1}, \theta_2^{\tau-1}\} - \varrho\tilde{g}$ # learning rate $\varrho$
14:     Release: $\{\theta_1^\tau, \theta_2^\tau\}$
    Output: $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$-Lifelong DP parameters $\{\theta^i\}_{i\in[1,m]} = \{\theta_1^i, \theta_2^i\}_{i\in[1,m]}$
```

3.3.4 Gradient Update g

To compute the gradient update g for $\{\theta_1^\tau, \theta_2^\tau\}$ (Equation (3.83)) on the current task τ, we first derive polynomial forms of $\mathcal{R}_{D_\tau}(\theta_1)$ and $\mathcal{L}_{D_\tau}(\theta_2)$, by applying the 1st and 2nd orders of Taylor Expansion [160] as:

$$\widetilde{\mathcal{R}}_{D_\tau}(\theta_1) = \sum_{x_r\in D_\tau}\sum_{s=1}^{d}\Big[\theta_{1s}\big(\frac{1}{2} - x_{rs}\big)h_r\Big] \tag{3.84}$$

$$\widetilde{\mathcal{L}}_{D_\tau}(\theta_2) = \sum_{k=1}^{K}\sum_{x_r\in D_\tau}\Big[\mathbf{h}_{\pi r}W_{\pi k} - (\mathbf{h}_{\pi r}W_{\pi k})y_{rk} - \frac{1}{2}|\mathbf{h}_{\pi r}W_{\pi k}| + \frac{1}{8}(\mathbf{h}_{\pi r}W_{\pi k})^2\Big] \tag{3.85}$$

where $\mathbf{h}_{\pi r}$ computed from the x_r through the network with W_π is the parameter at the last hidden layer $\mathbf{h}_\pi$. Laplace noise is injected into polynomial coefficients of the function $\widetilde{\mathcal{R}}_{D_\tau}(\theta_1)$, which are the input x and the first transformation $\mathbf{h}_1$. As in [96], the global sensitivity $\Delta_{\widetilde{\mathcal{R}}}$ is bounded as: $\Delta_{\widetilde{\mathcal{R}}} \leq d(|\mathbf{h}_1| + 2)$, with $|\mathbf{h}_1|$ is the number of neurons in $\mathbf{h}_1$.

The perturbed $\widetilde{\mathcal{R}}$ function becomes:

$$\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1) = \sum_{\overline{x}_r \in \overline{D}_\tau} \Big[\sum_{s=1}^{d} (\frac{1}{2}\theta_{1s}\overline{h}_r) - \overline{x}_r \widetilde{x}_r \Big] \tag{3.86}$$

where $\overline{x}_r = x_r + \frac{1}{|\mathcal{D}_\tau|}Lap(\frac{\Delta_{\widetilde{\mathcal{R}}}}{\epsilon_1})$, $h_r = \theta_1^\top \overline{x}_r$, $\overline{h}_r = h_r + \frac{2}{|\mathcal{D}_\tau|}Lap(\frac{\Delta_{\widetilde{\mathcal{R}}}}{\epsilon_1})$, $\widetilde{x}_r = \theta_1 \overline{h}_r$, h_r is clipped to $[-1, 1]$, and ϵ_1 is a privacy budget.

More importantly, the perturbation of each example x turns the original data D_τ into $\overline{D}_\tau = \{\overline{x}_r \leftarrow x_r + \frac{1}{|\mathcal{D}_\tau|}Lap(\frac{\Delta_{\widetilde{\mathcal{R}}}}{\epsilon_1})\}_{x_r \in D_\tau}$, which is a $(\epsilon_1/\gamma_{\mathbf{x}})$-DP dataset with $\gamma_{\mathbf{x}} = \Delta_{\widetilde{\mathcal{R}}}/|D_\tau|$ (Algorithm 3.6, line 2). Based upon this result, all the computations on top of the $(\epsilon_1/\gamma_{\mathbf{x}})$-DP dataset $\overline{D}_\tau$, including h_r, $\overline{h}_r$, $\widetilde{x}_r$, and the computation of gradients, i.e., $\forall s \in [1, d] : \nabla_{\theta_{1s}} \overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1) = \frac{\delta \overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1)}{\delta \theta_{1s}} = \sum_{r=1}^{|\mathcal{D}_\tau|} \overline{h}_r(\frac{1}{2} - \overline{x}_{rs})$ are shown to be $(\epsilon_1/\gamma_{\mathbf{x}})$-DP, without incurring or accessing any additional information from the original data D_τ. As a result, the total privacy budget used to perturb $\widetilde{\mathcal{R}}$ is $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}})$, by having $\frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1)\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_\tau}(\theta_1)\big)} \times \frac{Pr\big(\overline{D}_\tau\big)}{Pr\big(\overline{D}'_\tau\big)} \leq (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}})$. Details are available in our proof of Theorem 3.8 (Subsection 3.3.5).

A similar approach is applied to perturb the objective function $\widetilde{\mathcal{L}}_{D_\tau}(\theta_2)$ at the output layer with a privacy budget ϵ_2. The perturbed function of $\widetilde{\mathcal{L}}$ is denoted as $\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2)$. As in Lemma 3 [96], we further have that the output of $a(\cdot)$, which is the perturbed transformation $\overline{\mathbf{h}}_{1\overline{D}_\tau} = \{\overline{\theta}_1^\top \overline{x}_r + \frac{2}{|\mathcal{D}_\tau|}Lap(\frac{\Delta_{\widetilde{\mathcal{R}}}}{\epsilon_1})\}_{\overline{x}_r \in \overline{D}_\tau}$, is (ϵ_1/γ)-DP, given $\gamma = \frac{2\Delta_{\widetilde{\mathcal{R}}}}{|\mathcal{D}_\tau| \|\overline{\theta}_1\|_{1,1}}$ and $\|\overline{\theta}_1\|_{1,1}$ is the maximum 1-norm of θ_1's columns [186]. As a result, the computations of all the hidden layers of $\mathcal{G}(a(\cdot), \theta_2)$ are (ϵ_1/γ)-DP, since the input of $\mathcal{G}(a(\cdot), \theta_2)$ is (ϵ_1/γ)-DP $\overline{\mathbf{h}}_{1\overline{D}_\tau}$, i.e., the post-processing property of DP [100] (Algorithm 3.6, line 2). This helps us to **(1)** avoid extra privacy budget consumption in computing $g(a(\cdot), \theta_2)$; **(2)** significantly tighten the sensitivity of the function $\overline{\mathcal{L}}_{\overline{D}_\tau}$ (i.e., $\Delta_{\widetilde{\mathcal{L}}} \leq 2|\mathbf{h}_\pi|$); and **(3)** achieve DP gradient update $\nabla_{\theta_2} \overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2)$ for θ_2. The total privacy budget used to perturb $\widetilde{\mathcal{L}}$ is $(\epsilon_1/\gamma + \epsilon_2)$, by having $Pr\big(\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2)\big)/Pr\big(\overline{\mathcal{L}}_{\overline{D}'_\tau}(\theta_2)\big) \leq (\epsilon_1/\gamma + \epsilon_2)$. Consequently, the total privacy budget in computing the gradient updates g, i.e., $\{\nabla_{\theta_1}\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1}), \nabla_{\theta_2}\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})\}$, for the current task τ is $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$-DP (Algorithm 3.6, lines 5 and 10).

3.3.5 Episodic and Projected Gradients g_{ref} and $\tilde{g}$

Now we are ready to present our approach in achieving Lifelong DP, by configuring the episodic memory $\mathbb{M}_\tau$ as a *fixed* and *disjoint* set of datasets $\{\overline{D}_1, \dots, \overline{D}_{\tau-1}\}$ (Algorithm 3.6, line 7); such that, at each training step, the computation of gradient updates g_{ref} (Equation (3.77)) for θ_1 and θ_2, i.e., $\nabla_{\theta_1}\overline{\mathcal{R}}_{\overline{D}_{ref}}(\theta_1)$ and $\nabla_{\theta_2}\overline{\mathcal{L}}_{\overline{D}_{ref}}(\theta_2)$, using a randomly picked dataset $\overline{D}_{ref} \in \mathbb{M}_\tau$ (Algorithm 3.6, lines 8 and 11), is $(\epsilon_1+\epsilon_1/\gamma_{\mathbf{x}}+\epsilon_1/\gamma+\epsilon_2)$-DP, without incurring any additional privacy budget consumption for the dataset D_{ref}. Then Equation (3.77) can be used to compute the *projected gradient* $\tilde{g}$ from g and g_{ref}. Based on the post-processing property of DP [100], the projected gradient $\tilde{g}$ is also $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$-DP. Hence, the L2M objective function (Equation (3.83)) can be reformulated as:

$$\begin{aligned}
&\{\theta_1^\tau, \theta_2^\tau\} = \arg\min_{\theta_1,\theta_2}[\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1}) + \overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})] \\
&\text{s.t. } \overline{\mathcal{R}}_{\mathbb{M}_\tau}(\theta_1^{\tau-1}) \leq \overline{\mathcal{R}}_{\mathbb{M}_\tau}(\theta_1^{\tau-1}), \overline{\mathcal{L}}_{\mathbb{M}_\tau}(\theta_2^{\tau-1}) \leq \overline{\mathcal{L}}_{\mathbb{M}_\tau}(\theta_2^{\tau-1}) \\
&\text{where } \mathbb{M}_\tau = \{\overline{D}_1, \dots, \overline{D}_{\tau-1}\}
\end{aligned} \tag{3.87}$$

By using the perturbed functions $\overline{\mathcal{R}}$ and $\overline{\mathcal{L}}$, the constrained optimization of Equation (3.87) can be addressed similarly to Equation (3.77), when the projected gradient $\tilde{g}$ is computed as: $\tilde{g} = g - (g^\top g_{ref})/(g_{ref}^\top g_{ref})g_{ref}$, where g is the gradient update on the current task τ, and g_{ref} is computed using a dataset $\overline{D}_{ref}$ randomly selected from the episodic memory $\mathbb{M}_\tau$.

Theorem 3.8 shows that Algorithm 3.6 achieves (ϵ, δ)-Lifelong DP in learning $\{\theta^i\}_{i\in[1,m]} = \{\theta_1^i, \theta_2^i\}_{i\in[1,m]}$, where $\epsilon = (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$ and $\delta = 0$. There are three key properties in our algorithm: **(1)** For every x in the whole training set $\overline{\mathcal{D}} = \{\overline{D}_i\}_{i \in [1,m]}$, x is included in *one and only one* dataset, denoted $\overline{D}_x \in \overline{\mathcal{D}}$. As a result, the DP guarantee to x in $\overline{\mathcal{D}} = \{\overline{D}_i\}_{i \in [1,m]}$ is equivalent to the DP guarantee to x in $\overline{D}_x$; **(2)** Given the episodic memory as a *fixed* and *disjoint* set of datasets across **T** training tasks, we can prevent additional privacy leakage, caused by: (i) Differing at most $i-1$ tuples between neighboring $\mathbb{M}_i$ and $\mathbb{M}'_i$ for all $i \in (1, m]$; and (ii) Generating new and overlapping sets of data samples for computing the episodic gradient (which are considered overlapping datasets in the parlance of DP) in the typical training. Therefore, the optimization on one task does not affect the DP protection of any other tasks, even the objective function given one task can be slightly different from the objective function given any other tasks; and **(3)** Together with (1) and (2), by having one and only one privacy budget for every task, we can simultaneously achieve Equations (3.81) and (3.82) in Lifelong DP (**Def. 3.5**).

Theorem 3.8. *Algorithm 3.6 achieves* $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$*-Lifelong DP in learning* $\{\theta_1^i, \theta_2^i\}_{i \in [1,m]}$.

Proof of Theorem 3.8

Proof. $\forall \tau \in$ **T**, let $\overline{D}_\tau$ and $\overline{D}'_\tau$ be neighboring datasets differing at most one tuple $x_e \in \overline{D}_\tau$ and $x'_e \in \overline{D}'_\tau$, and any two neighboring episodic memories $\mathbb{M}_\tau$ and $\mathbb{M}'_\tau$. Let us denote Algorithm 3.6 as the mechanism A in Definition 3.5. We first show that Algorithm 3.6 achieves typical DP protection. $\forall \tau$ and D_{ref}, we have that

$$Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}_\tau) = \theta^\tau\big] = Pr\big(\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1})\big) Pr\big(\overline{D}_\tau\big) Pr\big(\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})\big) \\ \times Pr\big(\overline{\mathcal{R}}_{\overline{D}_{ref}}(\theta_1^{\tau-1})\big) Pr\big(\overline{D}_{ref}\big) Pr\big(\overline{\mathcal{L}}_{\overline{D}_{ref}}(\theta_2^{\tau-1})\big)$$

Therefore, we further have

$$\frac{Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}_\tau) = \theta^\tau\big]}{Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}'_\tau) = \theta^\tau\big]} = \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_\tau}(\theta_1^{\tau-1})\big)} \frac{Pr\big(\overline{D}_\tau\big)}{Pr\big(\overline{D}'_\tau\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}'_\tau}(\theta_2^{\tau-1})\big)}$$
$$\times \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_{ref}}(\theta_1^{\tau-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_{ref}}(\theta_1^{\tau-1})\big)} \frac{Pr\big(\overline{D}_{ref}\big)}{Pr\big(\overline{D}'_{ref}\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}_{ref}}(\theta_2^{\tau-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}'_{ref}}(\theta_2^{\tau-1})\big)} \quad (3.88)$$

In addition, we also have that:

$$\exists! \overline{D}_\tau \in \overline{\mathcal{D}} \; s.t. \; x_e \in \overline{D}_\tau \; and \; \exists! \overline{D}'_\tau \in \overline{\mathcal{D}}' \; s.t. \; x'_e \in \overline{D}'_\tau \quad (3.89)$$

where $\overline{\mathcal{D}} = \{\overline{D}_1, \ldots, \overline{D}_m\}$.

Together with Equation (3.89), by having disjoint and fixed datasets in the episodic memory, we have that:

$$(x_e \in \overline{D}_\tau \; or \; x_e \in \overline{D}_{ref}), \; but \; (x_e \in \overline{D}_\tau \; and \; x_e \in \overline{D}_{ref}) \quad (3.90)$$

Without loss of the generality, we can assume that $x_e \in \overline{D}_\tau$: Equations (3.88) - (3.90) $\Rightarrow$

$$\frac{Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}_\tau) = \theta^\tau\big]}{Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}'_\tau) = \theta^\tau\big]} = \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_\tau}(\theta_1^{\tau-1})\big)} \frac{Pr\big(\overline{D}_\tau\big)}{Pr\big(\overline{D}'_\tau\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}'_\tau}(\theta_2^{\tau-1})\big)} \quad (3.91)$$
$$\leq (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2) \quad (3.92)$$

This is also true when $x_e \in \overline{D}_{ref}$ *and* $x_e \notin \overline{D}_\tau$.

As a result, we have

$$\forall \tau \in [1, m] : \frac{Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}_\tau) = \theta^\tau\big]}{Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}'_\tau) = \theta^\tau\big]} \leq (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2) \tag{3.93}$$

After one training step, $\overline{D}_\tau$ will be placed into the episodic memory $\mathbb{M}_\tau$ to create the memory $\mathbb{M}_{\tau+1}$. In the next training task, $\overline{D}_\tau$ can be randomly selected to compute the episodic gradient g_{ref}. This computation does not incur any additional privacy budget consumption for the dataset $\overline{D}_\tau$, by applying the Theorem 4 in [96], which allows us to *compute gradients across an unlimited number of training steps* using $\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1})$ and $\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})$. Therefore, if the same privacy budget is used for all the training tasks in **T**, we will have only one privacy loss for every tuple in all the tasks. The optimization in one task does not affect the DP guarantee of any other tasks. Consequently, we have

$$\begin{aligned} &\nexists \epsilon' < (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2), \exists i \leq m \\ &\text{s.t. } Pr\big[A(\{\theta^j\}_{j<i}, \mathsf{data}_i) = \theta^i\big] \leq e^{\epsilon'} Pr\big[A(\{\theta^j\}_{j<i}, \mathsf{data}'_i) = \theta^i\big] \end{aligned} \tag{3.94}$$

Equation (3.94) can be further used to prove the Lifelong DP protection. Given data_m where $M_t = \overline{D}_t$ in Algorithm 3.6, we have that

$$Pr\big[A(\mathsf{data}_m) = \{\theta^i\}_{i\in[1,m]}\big] = \prod_{i=1}^{m} Pr\big[A(\{\theta^j\}_{j<i}, \mathsf{data}_i) = \theta^i\big] \tag{3.95}$$

Therefore, we have

$$\frac{Pr\big[A(\mathsf{data}_m) = \{\theta^i\}_{i\in[1,m]}\big]}{Pr\big[A(\mathsf{data}'_m) = \{\theta^i\}_{i\in[1,m]}\big]} = \prod_{i=1}^{m} \frac{Pr\big[A(\{\theta^j\}_{j<i}, \mathsf{data}_i) = \theta^i\big]}{Pr\big[A(\{\theta^j\}_{j<i}, \mathsf{data}'_i) = \theta^i\big]}$$
$$= \prod_{i=1}^{m} \Big[\frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_i}(\theta_1^{i-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_i}(\theta_1^{i-1})\big)} \frac{Pr\big(\overline{D}_i\big)}{Pr\big(\overline{D}'_i\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}_i}(\theta_2^{i-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}'_i}(\theta_2^{i-1})\big)}$$
$$\times \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}^i_{ref}}(\theta_1^{i-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}^{i'}_{ref}}(\theta_1^{i-1})\big)} \frac{Pr\big(\overline{D}^i_{ref}\big)}{Pr\big(\overline{D}^{i'}_{ref}\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}^i_{ref}}(\theta_2^{i-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}^{i'}_{ref}}(\theta_2^{i-1})\big)} \Big] \quad (3.96)$$

where $\mathsf{data}'_m = \{\overline{\mathcal{D}}, \{M'_i\}_{i\in[1,m]}\}$, and $M'_i = \overline{D}'_i$ in Algorithm 3.6.

Since all the datasets are non-overlapping, i.e., $\cap_{i\in[1,m]} D_i = \emptyset$, given an arbitrary tuple x_e, we have that

$$\exists! \overline{D}_\tau \in \overline{\mathcal{D}} \textit{ s.t. } x_e \in \overline{D}_\tau \textit{ and } \exists! \overline{D}'_\tau \in \overline{\mathcal{D}}' \textit{ s.t. } x'_e \in \overline{D}'_\tau \quad (3.97)$$

Thus, the optimization of $\{\theta_1^i, \theta_2^i\} = \arg\min_{\theta_1,\theta_2}[\overline{\mathcal{R}}_{\overline{D}_i}(\theta_1^{i-1}) + \overline{\mathcal{L}}_{\overline{D}_i}(\theta_2^{i-1})]$ for any other task i different from τ does not affect the privacy protection of x_e in $\overline{\mathcal{D}}$. From Equations (3.96) and (3.97), we have

$$\frac{Pr\big[A(\mathsf{data}_m) = \{\theta^i\}_{i\in[1,m]}\big]}{Pr\big[A(\mathsf{data}'_m) = \{\theta^i\}_{i\in[1,m]}\big]} = \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{\tau-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_\tau}(\theta_1^{\tau-1})\big)} \frac{Pr\big(\overline{D}_\tau\big)}{Pr\big(\overline{D}'_\tau\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{\tau-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}'_\tau}(\theta_2^{\tau-1})\big)}$$
$$\times \prod_{i=1}^{m} \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}^i_{ref}}(\theta_1^{i-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}^{i'}_{ref}}(\theta_1^{i-1})\big)} \frac{Pr\big(\overline{D}^i_{ref}\big)}{Pr\big(\overline{D}^{i'}_{ref}\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}^i_{ref}}(\theta_2^{i-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}^{i'}_{ref}}(\theta_2^{i-1})\big)} \quad (3.98)$$

The worse privacy leakage case to x_e is that $\overline{D}_\tau$ is used in every $\overline{D}^i_{ref}$, i.e., $\tau = 1$ and $\forall i \in [2, m] : \overline{D}^i_{ref} = \overline{D}_\tau$, with $\overline{D}^1_{ref} = \emptyset$. Meanwhile, the least privacy leakage case to x_e

is that $\overline{D}_\tau$ is not used in any $\overline{D}^i_{ref}$, i.e., $\forall i \in [2, m] : \overline{D}^i_{ref} \neq \overline{D}_\tau$, with $\overline{D}^1_{ref} = \emptyset$. In order to bound the privacy loss, we consider the worse case; therefore, from Equation (3.98), we further have that

$$\frac{Pr\big[A(\mathsf{data}_m) = \{\theta^i\}_{i\in[1,m]}\big]}{Pr\big[A(\mathsf{data}'_m) = \{\theta^i\}_{i\in[1,m]}\big]} \leq \prod_{i=1}^{m} \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{i-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_\tau}(\theta_1^{i-1})\big)} \frac{Pr\big(\overline{D}_\tau\big)}{Pr\big(\overline{D}'_\tau\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{i-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}'_\tau}(\theta_2^{i-1})\big)} \tag{3.99}$$

Equation (3.99) is equivalent to the continuously training of our model by optimizing $\overline{\mathcal{R}}$ and $\overline{\mathcal{L}}$ with $\overline{D}_\tau$ used as both the current task and the episodic memory, across m steps. By following the Theorem 4 in [96], the privacy budget is not accumulated across training steps. Therefore, we have that

$$\begin{aligned}
\forall m \in [1, \infty) : & \frac{Pr\big[A(\mathsf{data}_m) = \{\theta^i\}_{i\in[1,m]}\big]}{Pr\big[A(\mathsf{data}'_m) = \{\theta^i\}_{i\in[1,m]}\big]} \\
& \leq \prod_{i=1}^{m} \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1^{i-1})\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_\tau}(\theta_1^{i-1})\big)} \frac{Pr\big(\overline{D}_\tau\big)}{Pr\big(\overline{D}'_\tau\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2^{i-1})\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}'_\tau}(\theta_2^{i-1})\big)} \\
& = \frac{Pr\big(\overline{\mathcal{R}}_{\overline{D}_\tau}(\theta_1)\big)}{Pr\big(\overline{\mathcal{R}}_{\overline{D}'_\tau}(\theta_1)\big)} \frac{Pr\big(\overline{D}_\tau\big)}{Pr\big(\overline{D}'_\tau\big)} \frac{Pr\big(\overline{\mathcal{L}}_{\overline{D}_\tau}(\theta_2)\big)}{Pr\big(\overline{\mathcal{L}}_{\overline{D}'_\tau}(\theta_2)\big)} \\
& \leq (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)
\end{aligned} \tag{3.100}$$

In the least privacy leakage case, we have that

$$\begin{aligned}\forall \tau \leq m :& \frac{Pr\big[A(\mathsf{data}_\tau) = \{\theta^i\}_{i\in[1,\tau]}\big]}{Pr\big[A(\mathsf{data}'_\tau) = \{\theta^i\}_{i\in[1,\tau]}\big]} \\ &\geq \frac{Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}_\tau) = \theta^\tau\big]}{Pr\big[A(\{\theta^i\}_{i<\tau}, \mathsf{data}'_\tau) = \theta^\tau\big]} \\ &\geq (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)\end{aligned} \tag{3.101}$$

As a result, we have that

$$\nexists(\epsilon' < \epsilon, \tau \leq m) : Pr\big[A(\mathsf{data}_\tau) = \{\theta^i\}_{i\in[1,\tau]}\big] \leq e^{\epsilon'} Pr\big[A(\mathsf{data}'_\tau) = \{\theta^i\}_{i\in[1,\tau]}\big] \tag{3.102}$$

where $\epsilon = (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$.

From Equations (3.100) and (3.102), we have that Algorithm 3.6 achieves $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$-Lifelong DP in learning $\{\theta^i\}_{i\in[1,m]} = \{\theta^i_1, \theta^i_2\}_{i\in[1,m]}$. Consequently, Theorem 3.8 does hold. □

3.3.6 Scalable and Heterogeneous Training

Although computing the gradients given the whole dataset $\overline{D}_\tau$ achieves Lifelong DP, it has some shortcomings: **(1)** consumes a large computational memory to store the episodic memory; **(2)** computational efficiency is low, since we need to use the whole dataset $\overline{D}_\tau$ and $\overline{D}_{ref}$ to compute the gradient update and the episodic gradient at each step; This results in a slow convergence speed and poor utility.

Scalability To address this, we propose a streaming batch training (**Algorithm 3.7**), in which a batch of data is used to train the model at each training step, by the following steps. **(1)** Slitting the private training data $\overline{D}_\tau$ ($\forall \tau \in \mathbf{T}$) into *disjoint* and *fixed* batches (Algorithm

Algorithm 3.7 L2DP-ML with Streaming Batch Training

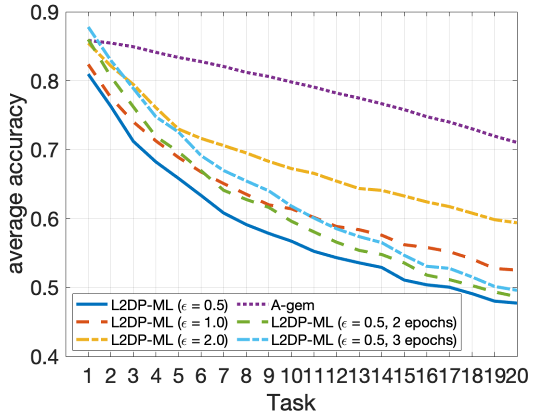
```
Input: T=$\{t_i\}_{i\in[1,m]}$, $\{D_i\}_{i\in[1,m]}$, batch size $\lambda$, privacy budgets: $\epsilon_1$ and $\epsilon_2$, learning rate $\varrho$
 1: Draw Noise $\chi_1 \leftarrow [Lap(\frac{\Delta_{\widetilde{\mathcal{R}}}}{\epsilon_1})]^d$, $\chi_2 \leftarrow [Lap(\frac{\Delta_{\widetilde{\mathcal{R}}}}{\epsilon_1})]^\beta$, $\chi_3 \leftarrow [Lap(\frac{\Delta_{\widetilde{\mathcal{L}}}}{\epsilon_2})]^{|\mathbf{h}_\pi|}$
 2: Randomly Initialize $\theta = \{\theta_1, \theta_2\}$, $\mathbb{M}_1 = \emptyset$, $\forall \tau \in \mathbf{T}$ : $\overline{D}_\tau = \{\overline{x}_r \leftarrow x_r + \frac{\chi_1}{\lambda}\}_{x_r \in D_\tau}$, hidden
    layers $\{\mathbf{h}_1 + \frac{2\chi_2}{\lambda}, \ldots, \mathbf{h}_\pi\}$, where $\mathbf{h}_\pi$ is the last hidden layer
 3: for $\tau \in \mathbf{T}$ do
 4:     $\mathbf{B} = \{B_1, \ldots, B_n\}$ s.t. $\forall B \in \mathbf{B}$ : $B$ is a random batch with the size $s$, $B_1 \cap \ldots \cap B_n = \emptyset$,
        and $B_1 \cup \ldots \cup B_n = \overline{D}_\tau$
 5:     for $B \in \mathbf{B}$ do
 6:         if $\tau == 0$ then
 7:             Compute Gradients:
 8:             $g \leftarrow \{\nabla_{\theta_1}\overline{\mathcal{R}}_B(\theta_1^{\tau-1}), \nabla_{\theta_2}\overline{\mathcal{L}}_B(\theta_2^{\tau-1})\}$ with the noise $\frac{\chi_3}{\lambda}$
 9:             Descent: $\{\theta_1^\tau, \theta_2^\tau\} \leftarrow \{\theta_1^{\tau-1}, \theta_2^{\tau-1}\} - \varrho g$
10:         else
11:             Select a batch $B_e$ randomly from a set of batches in episodic memory $\mathbb{M}_\tau$
12:             Compute Gradients:
13:             $g \leftarrow \{\nabla_{\theta_1}\overline{\mathcal{R}}_B(\theta_1^{\tau-1}), \nabla_{\theta_2}\overline{\mathcal{L}}_B(\theta_2^{\tau-1})\}$ with the noise $\frac{\chi_3}{\lambda}$
14:             $g_{ref} \leftarrow \{\nabla_{\theta_1}\overline{\mathcal{R}}_{B_e}(\theta_1^{\tau-1}), \nabla_{\theta_2}\overline{\mathcal{L}}_{B_e}(\theta_2^{\tau-1})\}$ with the noise $\frac{\chi_3}{\lambda}$
15:             $\tilde{g} \leftarrow g - \frac{g^\top g_{ref}}{g_{ref}^\top g_{ref}} g_{ref}$
16:             Descent: $\{\theta_1^\tau, \theta_2^\tau\} \leftarrow \{\theta_1^{\tau-1}, \theta_2^{\tau-1}\} - \varrho\tilde{g}$
17:     Randomly Select a batch $B \in \mathbf{B}$
18:     $\mathbb{M}_\tau \leftarrow \mathbb{M}_{\tau-1} \cup B$
    Output: $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2)$-Lifelong DP parameters $\{\theta^i\}_{i\in[1,m]} = \{\theta_1^i, \theta_2^i\}_{i\in[1,m]}$
```

Table 3.2 Average Forgetting Measure

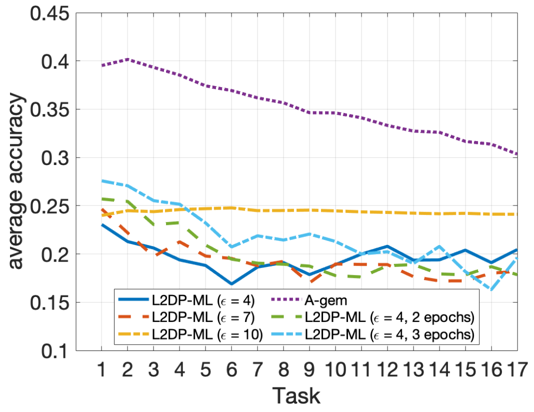
Permuted MNIST	L2DP-ML ($\epsilon = 0.5$)	L2DP-ML ($\epsilon = 1$)	L2DP-ML ($\epsilon = 2$)	A-gem
	0.305 ± 0.00886	0.278 ± 0.00907	0.237 ± 0.00586	0.162 ± 0.01096

Permuted CIFAR-10	L2DP-ML ($\epsilon = 4$)	L2DP-ML ($\epsilon = 7$)	L2DP-ML ($\epsilon = 10$)	A-gem
	0.033 ± 0.00896	0.062 ± 0.01508	0.034 ± 0.00184	0.133 ± 0.00859

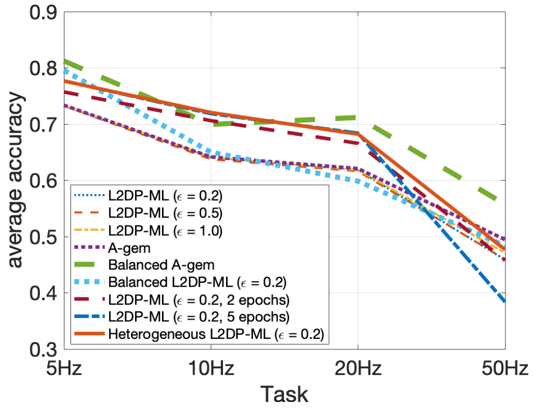
HARW (5Hz - 10Hz - 20Hz - 50Hz)	L2DP-ML ($\epsilon = 0.2$)	L2DP-ML ($\epsilon = 0.5$)	L2DP-ML ($\epsilon = 1$)
	0.1133 ± 0.0003	0.1124 ± 0.00029	0.1106 ± 0.00026
	A-gem	Balanced A-gem	Balanced L2DP-ML ($\epsilon = 0.2$)
	0.1269 ± 0.00045	0.1593 ± 0.00021	0.1309 ± 0.002
	L2DP-ML ($\epsilon = 0.2$, 2 epochs)	L2DP-ML ($\epsilon = 0.2$, 5 epochs)	Heterogeneous L2DP-ML ($\epsilon = 0.2$)
	0.1639 ± 0.00074	0.2031 ± 0.0013	0.1920 ± 0.00034



(a) Permuted MNIST (20 tasks)

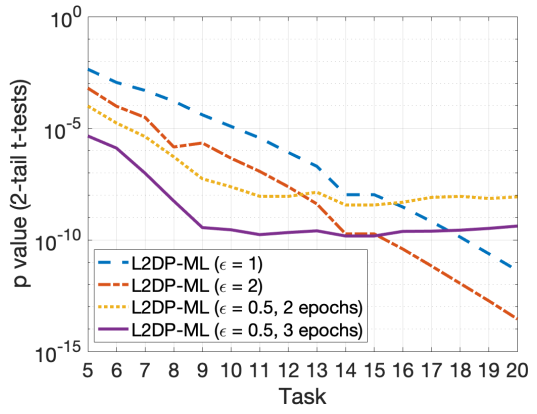


(b) Permuted CIFAR-10 (17 tasks)

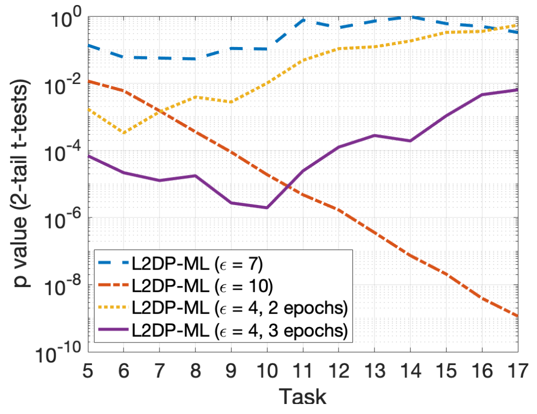


(c) HARW (5Hz - 10Hz - 20Hz - 50Hz)

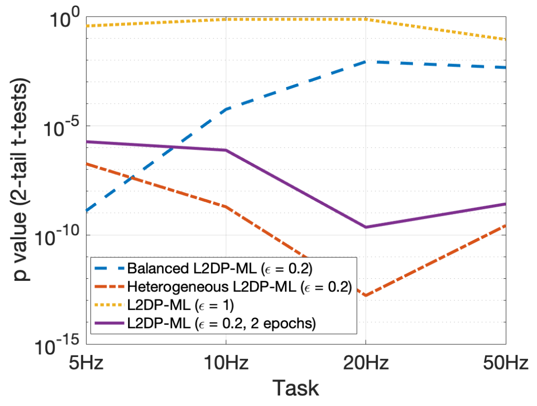
Figure 3.15 Average accuracy on the permuted MNIST and CIFAR-10 datasets, and on the HARW dataset (higher the better).



(a) Permuted MNIST (20 tasks)



(b) Permuted CIFAR-10 (17 tasks)



(c) HARW (5Hz - 10Hz - 20Hz - 50Hz)

Figure 3.16 p value for 2-tail t-tests on the permuted MNIST and CIFAR-10 datasets, and on the HARW dataset (lower the better).

3.7, line 4). **(2)** Using a single draw of Laplace noise across batches (Algorithm 3.7, lines 1-2). That prevents additional privacy leakage, caused by: (i) Generating multiple draws of noise (i.e., equivalent to applying one DP-preserving mechanism multiple times on the same dataset); (ii) Generating new and overlapping batches (which are considered overlapping datasets in the parlance of DP); and (iii) More importantly, for any example x, x is included in *only one* batch. Hence, each *disjoint batch* of data in Algorithm 3.7 can be considered as a *separate dataset* in Algorithm 3.6. **(3)** For each task, we randomly select a batch to place in the episodic memory (Algorithm 3.7, line 17). **(4)** At each training step, a batch from the current task is used to compute the gradient g, and a batch randomly selected from the episodic memory is used to compute the episodic gradient g_{ref} (Algorithm 3.7, lines 11-14). Thus, Algorithm 3.7 still preserves $(\epsilon_1+\epsilon_1/\gamma_{\mathbf{x}}+\epsilon_1/\gamma+\epsilon_2)$-Lifelong DP, by applying Theorem 3.8. The computational complexity and memory consumption are reduced, since only a small batch of data from each task will be stored in the episodic memory.

Heterogeneity Based upon this, our algorithm can be applied to address the heterogeneity in terms of data sizes among tasks, which is different from multi-modal tasks in [201]. We can train one task with multiple epochs, without affecting the Lifelong DP protection in Algorithm 3.7, by 1) keeping all the batches fixed among epochs, and 2) at the end of training each task, we randomly select a batch of that task to place in the episodic memory. The order of the task does not affect the Lifelong DP protection, since the privacy budget is not accumulated across tasks. These distinct properties enable us to customize our training, by having different numbers of training epochs for different tasks and having different training orders of tasks. Tasks with *smaller numbers of data tuples* can have *larger numbers of training epochs*. This helps us to achieve better model utility under the same privacy protection, as shown in our experimental results. Our algorithm can also be used to train L2M models on new data of past tasks. New data is split into disjoint and fixed batches, which will be fed into Algorithm 3.8, without affecting our Lifelong DP protection.

3.3.7 Experiments

Our validation focuses on understanding the impacts of the *privacy budget* ϵ and the *heterogeneity* on model utility. We have experimented on permuted MNIST [105] and permuted CIFAR-10 datasets, and our human activity recognition in the wild (**HARW**) dataset. Permuted MNIST is a variant of MNIST [47] dataset, where each task has a random permutation of the input pixels, which is applied to all the images of that task. We adopt this approach to permute the CIFAR-10 dataset, including the input pixels and three color channels. Our HARW dataset was collected from 116 users, each of whom provided mobile sensor data and labels for their activities on Android phones consecutively in three months (either April 1st - June 31st, or May 10th - August 10th, 2020). HARW is an ultimate task for L2M, since different sensor sampling rates, e.g., 50Hz, 20Hz, 10Hz, and 5Hz, from different mobile devices are considered as L2M tasks. The data collection and processing of our HARW dataset is presented as follows.

Data Collection of HARW Dataset We utilize Android smartphones to collect sensor data "in the wild" from university students as subjects for the following reasons:
(1) University students should have relatively good access to the smartphones and related technologies; **(2)** University students should be more credible and easier to be motivated than other sources (e.g., recruiting test subjects on crowd-sourcing websites); and **(3)** It will be easier for our team to recruit and distribute rewards to students. We launched two data collection runs at two universities for three months each. During the course of three months, we let the participants to collect data and labels by themselves (in the wild), and only intervene through reminding emails if we saw a decline in the amount of daily activities. A total of 116 participants were recorded after the two data collection runs.

Data Processing of HARW Dataset For the demonstration purpose of this work, we use only accelerometer data. Our data processing consists of the following steps: **(1)** Any duplicated data points (e.g., data points that have the same timestamp) are merged by taking the average of their sensor values; **(2)** Using 300 milliseconds as the threshold, continuous data sessions are identified and separated by breaking up the data sequences at any gap that is larger than the threshold; **(3)** Data sessions that have unstable or unsuitable sampling rates are filtered out. We only keep the data sessions that have a stable sampling rate of 5Hz, 10Hz, 20Hz, or 50Hz; **(4)** The label sessions that are associated with each data session (if any) are identified from the raw labels. Note that the label sessions are also filtered with the following two criteria to ensure good quality: (a) The first 10 seconds and the last 10 seconds of each label session are trimmed, due to the fact that users were likely operating the phone during these time periods; (b) Any label session longer than 30 minutes is trimmed down to 30 minutes, in order to mitigate the potential inaccurate labels due to users' negligence (forgot to turn off labeling); and **(5)** We sample data segments at the size of 100 data points with sliding windows. Different overlapping percentages were used for different classes and different sampling rates. The majority classes have 25% overlapping to reduce the number of data segments, while the minority classes have up to 90% overlapping to increase the available data segments. The same principle is applied to sessions with different sampling rates. We sample 15% of data for testing, while the rest are used for training (Table 3.3).

Table 3.3 Statistics of the HARW Dataset

Class	Description	N training	N testing
Walking	Walking	49376	8599
Sitting	Exclude in vehicle	52448	8744
In-Vehicle, Car	Driving, sitting	49536	8586
Cycling		14336	2537
Workout, Running		1984	319
*All classes exclude phone position = "Table"			

Data Normalization of HARW Dataset In our L2DP-ML models, we normalize the accelerometer data with the following steps: **(1)** We compute the mean and variance of each axis (i.e., X, Y, and Z) using only training data to avoid information leakage from the training phase to the testing phase. Then, both training and testing data are normalized with z-score, based on the mean and variance computed from training data; **(2)** Based on this, we clip the values in between $[min, max] = [-2, 2]$ for each axis, which covers at least 90% of possible data values; and **(3)** Finally, all values are linearly scaled to $[-1, 1]$ to finish the normalization process, as $x = 2 \times [\frac{x-min}{max-min} - 1/2]$.

Experiment Setting **A-gem** [54], which is one of the state-of-the-art L2M algorithms, is included in our experiments to show the upper bound in terms of model performance, since A-gem is a noiseless model. We aim to show how much model utility is compromised for the Lifelong DP protection.

To evaluate the heterogeneity, we derive several versions of our algorithm (Algorithm 3.7), including: **(1) Balanced L2DP-ML**, in which all the tasks have the same number of training steps, given a fixed batch size. This is also true for a **Balanced A-gem** algorithm; **(2) L2DP-ML** with the same number of epochs for all the tasks; and **(3) Heterogeneous L2DP-ML**, in which a fixed number of training epochs is assigned to each task. The numbers of epochs among tasks can be different. For instance, 5 epochs are used to train tasks with 5Hz, 10Hz, and 20Hz data, and 1 epoch is used to train the task with a larger volume of 50Hz data. The number of epochs is empirically identified by the data size of each task, since the search space of the number of epochs for each task is exponentially large. We do not consider the model derived from Equation (3.80), since it does not preserve Lifelong DP.

Model Configuration In the permuted MNIST dataset, we used three convolutional layers (32, 64, and 96 features). Each hidden neuron connects with a 5x5 unit patch. A fully-connected layer has 512 units. The batch size s was set to 2,500, and learning rate

$\varrho = 0.1$. Given a predefined total privacy budget ϵ, ϵ_2 is set to be 0.1, and ϵ_1 is computed as: $\epsilon_1 = \frac{\epsilon - \epsilon_2}{(1+1/\gamma+1/\gamma_{\mathbf{x}})}$. This ensures that $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2) = \epsilon$. $\Delta_{\widetilde{\mathcal{R}}} = (14^2 + 2) \times 25$ and $\Delta_{\widetilde{\mathcal{L}}} = 2 \times 512$. In the permuted CIFAR-10 dataset, we used a Resnet-18 network (64, 64, 128, 128, and 160 features) with kernels (4, 3, 3, 3, and 3). One fully-connected layer has 256 neurons. The batch size s was set to 500, and learning rate $\varrho = 0.2$. ϵ_2 is set to $(1 + r/3.0)$ and $\epsilon_1 = (1 + 2r/3.0)/(1 + 1/\gamma + 1/\gamma_{\mathbf{x}})$, where $r \geq 0$ is a ratio to control the total privacy budget ϵ in our experiment. For instance, given $r = 0$, we have that $\epsilon = (\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2) = 2$. $\Delta_{\widetilde{\mathcal{R}}} = 3 \times (14^2 + 2) \times 16$ and $\Delta_{\widetilde{\mathcal{L}}} = 2 \times 256$.

In the HARW dataset, each data tuple includes 100 values $\times$ 3 channels of the accelerometer sensor, i.e., 300 values in total as a model input. The classification output includes five classes of human activities, i.e., walking, sitting, in car, cycling, and running (Table 3.3). Given 20Hz, 5Hz, 10Hz, and 50Hz tasks, we correspondingly have 881, 7553, 621, and 156,033 data points in training and 159, 1,297, 124, and 27,134 data points in testing. We used three convolutional layers (32, 64, and 96 features). Each hidden neuron connects with a 2x2 unit patch. A fully-connected layer has 128 units. The batch size s was set to 120, and learning rate $\varrho = 5e - 5$. Given a total privacy budget ϵ, ϵ_2 is set to 0.1 and $\epsilon_1 = \frac{\epsilon - \epsilon_2}{(1+1/\gamma+1/\gamma_{\mathbf{x}})}$ to ensure that $(\epsilon_1 + \epsilon_1/\gamma_{\mathbf{x}} + \epsilon_1/\gamma + \epsilon_2) = \epsilon$. $\Delta_{\widetilde{\mathcal{R}}} = 4(14^2 + 2)$ and $\Delta_{\widetilde{\mathcal{L}}} = 2 \times 128$.

The experiments were conducted on a server of 4 GPUs, each of which is an NVIDIA TITAN Xp, 12 GB with 3,840 CUDA cores. The number of runs for each experiment on the permuted MNIST and CIFAR-10 datasets is 5 and on the HARW dataset is 10. For reproducibility, our implementation is available here[1].

Evaluation Metrics To evaluate our models, we employ the well-applied average accuracy and forgetting measures after the model has been trained with all the batches up till task τ [202, 54], defined as follows: **(1)** *average accuracy*$_\tau = \frac{1}{\tau}\sum_{t=1}^{\tau} a_{\tau,n,t}$, where $a_{\tau,n,t} \in$

[1]`https://www.dropbox.com/s/xzz318ap700kq4a/L2DP-ML.zip?dl=0`

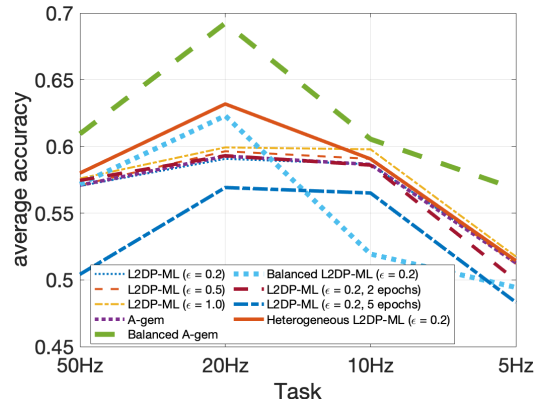
$[0,1]$ is the accuracy evaluated on the test set of task t, after the model has been trained with the n^{th} batch of task τ, and the training dataset of each task, D_τ, consists of a total n batches; **(2)** *average forgetting*$_\tau = \frac{1}{\tau-1}\sum_{t=1}^{\tau-1} f_t^\tau$, where f_t^τ is the forgetting on task t after the model is trained with all the batches up till task τ. f_t^τ is computed as follows: $f_t^\tau = \max_{l\in\{1,\dots,\tau-1\}}(a_{l,n,t} - a_{\tau,n,t})$; and **(3)** We measure the significant difference between two average accuracy curves induced by two models A and B after task τ, using a ***p* value (2-tail t-tests) curve**:

$$p\ value\big(\{\frac{1}{i}\sum_{t=1}^{i} a_{i,n,t}^{(A)}\}_{i\in[1,\tau]}, \{\frac{1}{i}\sum_{t=1}^{i} a_{i,n,t}^{(B)}\}_{i\in[1,\tau]}\big) \tag{3.103}$$

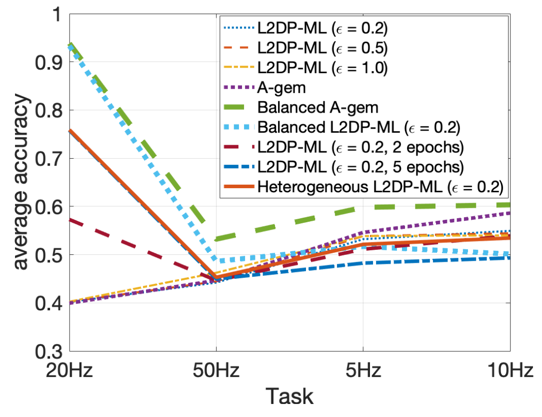
All statistical tests are 2-tail t-tests.

Results in Permuted MNIST Figure 3.15a and Table 3.2 illustrate the average accuracy and forgetting measure of each model as a function of the privacy budget ϵ on the MNIST dataset. There is a small gap in terms of average accuracy between the noiseless A-gem model (showing an upper bound performance) and our L2DP-ML models given a small number of tasks. The gap is increased when the number of tasks increased (23.3% at $\epsilon = 0.5$ with 20 tasks). The larger the privacy budget (i.e., $\epsilon = 2.0$), the higher the average accuracy we can achieve, i.e., an improvement of 9.92% with $p < 2.83e-14$, compared with smaller privacy budgets (i.e., $\epsilon = 0.5$). Even though our model can achieve a high average accuracy given a small number of tasks, the result shows that it is not easy to preserve Lifelong DP while retaining a high model utility. Also, our L2DP-ML models have a relatively good average forgetting with tight privacy protection ($\epsilon = 0.5, 1,$ and 2), compared with the noiseless A-gem model.

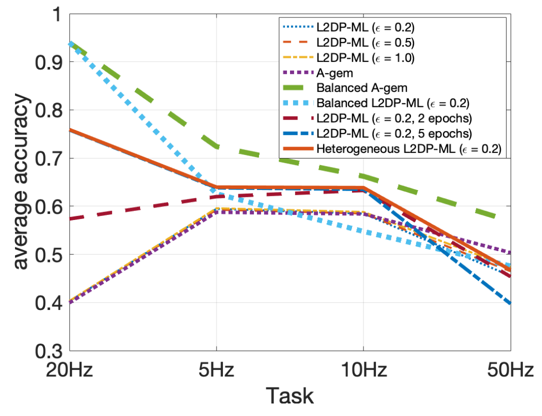
Results in Permuted CIFAR-10 Permuted CIFAR-10 tasks are very difficult to classify, even with the noiseless A-gem model, i.e., 35.24% accuracy on average. Interestingly,



(a) HARW 50Hz - 20Hz - 10Hz - 5Hz

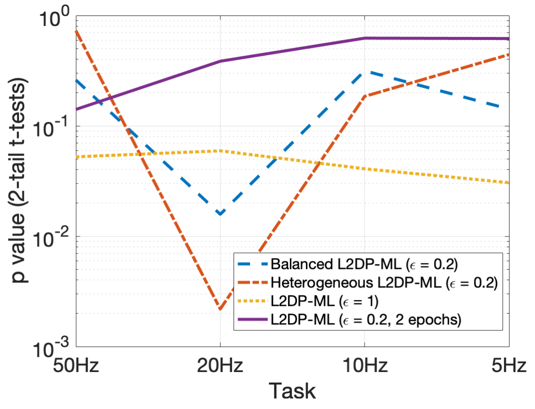


(b) HARW 20Hz - 50Hz - 5Hz - 10Hz

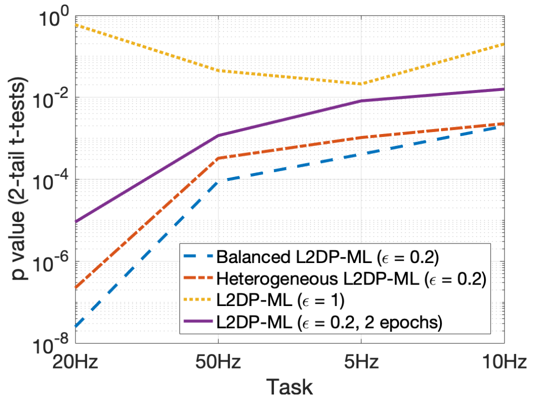


(c) HARW 20Hz - 5Hz - 10Hz - 50Hz

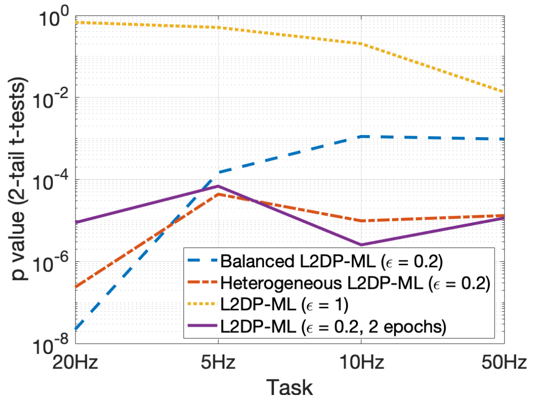
Figure 3.17 Average accuracy and p value for 2-tail t-tests on the HARW dataset with random task orders (higher the better).



(a) HARW 50Hz - 20Hz - 10Hz - 5Hz



(b) HARW 20Hz - 50Hz - 5Hz - 10Hz



(c) HARW 20Hz - 5Hz - 10Hz - 50Hz

Figure 3.18 p value for 2-tail t-tests on the HARW dataset with random task orders (lower the better).

on the CIFAR-10 dataset (Figure 3.15b and Table 3.2), the gap between A-gem and our L2DP-ML model is notably shrunken when the number of tasks increases (from 16.47% with 1 task to 9.89% with 17 tasks, at $\epsilon = 4$). In addition, the average forgetting values in our L2DP-ML are better than the noiseless A-gem model. This is a promising result. We also registered that the larger the privacy budget (i.e., $\epsilon = 10$), the higher the average accuracy that we can achieve, i.e., an improvement of 4.73% with $p < 1.15e-9$, compared with smaller budgets (i.e., $\epsilon = 4$).

Results in HARW On the HARW task, a real-world application (Figure 3.15c and Table 3.2), our L2DP-ML model achieves a very competitive average accuracy, given a very tight DP budget $\epsilon = 0.2$ (i.e., 61.26%) compared with the noiseless A-gem model (i.e., 62.27%), across four tasks. Our model also achieves a better average forgetting, i.e., 11.33, compared with 12.69 of the noiseless A-gem model. That is promising. Increasing the privacy budget modestly increases the model performance. The differences in terms of average accuracy and forgetting are not significant. This is also true, when we randomly flip the order of the tasks (Figure 3.17 and Table 3.4). The results showed that our model can effectively preserve Lifelong DP in HARW tasks.

Heterogeneous Training and DP Budgets Heterogeneous training, with customized numbers of epochs and task orders, further improves our model performance, under the same Lifelong DP protection. Figure 3.16 illustrates the p values between the average accuracy curves of our L2DP-ML, given **1)** heterogeneous training with different numbers of epochs, **2)** task orders, and **3)** privacy budgets, over its basic settings, i.e., $\epsilon = 0.5$ for the permuted MNIST dataset, $\epsilon = 4$ for the permuted CIFAR-10 dataset, and $\epsilon = 0.2$ for the HARW dataset, with the number of training epochs set to 1.

In the permuted MNIST dataset (Figures 3.15a, 3.16a), when our L2DP-ML model is trained with 2 or 3 epochs per task, the average accuracy is improved, i.e., 2.81% given 2 epochs and 4.8% given 3 epochs, with $p < 8.44e-9$. In the permuted CIFAR-10 data,

Table 3.4 Average Forgetting Measure on Random Orders of HARW Tasks

HARW (50Hz - 20Hz - 10Hz - 5Hz)	L2DP-ML ($\epsilon = 0.2$)	L2DP-ML ($\epsilon = 0.5$)	L2DP-ML ($\epsilon = 1$)
	0.1016 $\pm$ 0.0002	0.1012 $\pm$ 0.0001	0.098 $\pm$ 0.0001
	A-gem	Balanced A-gem	Balanced L2DP-ML ($\epsilon = 0.2$)
	0.1029 $\pm$ 0.0002	0.1241 $\pm$ 0.0002	0.1274 $\pm$ 0.0008
	L2DP-ML ($\epsilon = 0.2$, 2 epochs)	L2DP-ML ($\epsilon = 0.2$, 5 epochs)	Heterogeneous L2DP-ML ($\epsilon = 0.2$)
	0.1148 $\pm$ 0.0002	0.1012 $\pm$ 0.0014	0.1442 $\pm$ 0.0003

HARW (20Hz - 50Hz - 5Hz - 10Hz)	L2DP-ML ($\epsilon = 0.2$)	L2DP-ML ($\epsilon = 0.5$)	L2DP-ML ($\epsilon = 1$)
	0.0769 $\pm$ 2.07e-5	0.0761 $\pm$ 3.88e-5	0.0772 $\pm$ 6.7e-5
	A-gem	Balanced A-gem	Balanced L2DP-ML ($\epsilon = 0.2$)
	0.0781 $\pm$ 2.28e-5	0.14 $\pm$ 3.26e-4	0.1248 $\pm$ 0.0013
	L2DP-ML ($\epsilon = 0.2$, 2 epochs)	L2DP-ML ($\epsilon = 0.2$, 5 epochs)	Heterogeneous L2DP-ML ($\epsilon = 0.2$)
	0.0775 $\pm$ 8.45e-5	0.099 $\pm$ 0.0015	0.1268 $\pm$ 0.00028

HARW (20Hz - 5Hz - 10Hz - 50Hz)	L2DP-ML ($\epsilon = 0.2$)	L2DP-ML ($\epsilon = 0.5$)	L2DP-ML ($\epsilon = 1$)
	0.0928 $\pm$ 5.34e-5	0.0921 $\pm$ 8.64e-5	0.089 $\pm$ 8.64e-5
	A-gem	Balanced A-gem	Balanced L2DP-ML ($\epsilon = 0.2$)
	0.0866 $\pm$ 1.1e-4	0.1723 $\pm$ 0.00066	0.144 $\pm$ 0.0031
	L2DP-ML ($\epsilon = 0.2$, 2 epochs)	L2DP-ML ($\epsilon = 0.2$, 5 epochs)	Heterogeneous L2DP-ML ($\epsilon = 0.2$)
	0.1161 $\pm$ 0.0003	0.1792 $\pm$ 0.0017	0.1395 $\pm$ 0.00026

using larger numbers of training epochs shows significant performance improvements over a small number of tasks (Figure 3.16b). When the number of tasks becomes larger, the p values become less significant (even insignificant), compared with the p value curves of larger DP budgets (i.e., $\epsilon = 2$ and $\epsilon = 10$ in the MNIST and CIFAR-10 datasets). Meanwhile, training with a larger number of epochs yields better results with small numbers of tasks (i.e., fewer than 6 tasks), compared with larger DP budgets.

In the HARW tasks, the improvement is more significant (Figures 3.15c, 3.16c). Heterogeneous and Balanced L2DP-ML models outperform the basic settings with uniform numbers of training epochs, i.e., 1, 2, and 5 epochs. On average, we registered an improvement of 1.93% given the Balanced L2DP-ML and an improvement of 5.14% given the Heterogeneous L2DP-ML, over the basic setting (1 training epoch). The results are statistically significant (Figure 3.16c). The average forgetting values of the Balanced L2DP-ML (0.1593) and the Heterogeneous L2DP-ML (0.1920) are higher than the basic setting (0.1133), with $p < 2.19e - 5$ (Table 3.2). This is expected as a primary trade-off in L2M, given a better average accuracy. In fact, the average forgetting values are also notably higher given larger uniform numbers of epochs, i.e., 2 and 5 epochs, and the Balanced A-gem. We do not address this fundamental issue in L2M since it is out-of-scope of this study. We focus on preserving Lifelong DP.

We observe similar results in randomly flipping the order of the tasks (Figures 3.17 and 3.18, Table 3.4). Among all task orders, our Heterogeneous L2DP-ML achieves the best average accuracy (66.4%) with the task order [5Hz, 10Hz, 20Hz, 50Hz] (Figure 3.15c) compared with the worse order [20Hz, 50Hz, 5Hz, 10Hz] (56.69%) (Figure 3.17b), i.e., $p < 9.9e - 5$. More importantly, in both average accuracy and forgetting, our Balanced and Heterogeneous L2DP-ML models achieve a competitive performance compared with the noiseless Balanced A-gem, which is considered to have the upper bound performance, and a better performance compared with having the uniform numbers of epochs across tasks.

This obviously shown that the distinct ability to offer the heterogeneity in training across tasks greatly improves our model performance, under the same Lifelong DP protection.

3.3.8 Conclusion and Future Work

In this dissertation, we showed that L2M introduces unknown privacy risk and challenges in preserving DP. To address this, we established a connection between DP preservation and L2M, through a new definition of Lifelong DP. In Lifelong DP, the participation of any tuple in any training set is protected, under a consistently bounded DP loss, given the released model parameters in both learning and memorizing tasks. A consistently bounded DP means having only one fixed value of the privacy budget, regardless the number of tasks. We proposed the first scalable and heterogeneous mechanism, called L2DP-ML, to preserve Lifelong DP. Our model shows promising results in several tasks with different settings; therefore, opening a long-term avenue to achieve better model utility with lower computational cost, under Lifelong DP.

CHAPTER 4

FEDERATED LEARNING ON MOBILE DEVICES

Chapter Abstract Federated Learning (FL) [106] has the potential to bring deep learning (DL) on mobile devices, while preserving user privacy during model training. Despite progress on theoretical aspects and algorithm/model design for FL [107, 108, 109, 110, 111], the lack of a publicly available FL system has precluded the widespread adoption of FL models on smart phones. In this dissertation, we presents **FLSys**, the first FL system in the literature created using an application-system co-design approach to address the aforementioned research questions. FLSys is a key component toward creating an open ecosystem of FL models and apps that use these model. Such an FL ecosystem will allow third-party model/app developers to easily develop and deploy FL models/apps on smart phones. Consequently, the users will benefit from novel FL apps based on mobile [sensing] data collected on the phones. We co-designed FLSys with a human activity recognition (HAR) in the wild FL model. HAR sensing data was collected in two areas from the phones of 100+ college students during a five-month period. We implemented HAR-Wild, a CNN model tailored to mobile devices, with a data augmentation mechanism to mitigate the problem of non-Independent and Identically Distributed (non-IID) data that affects FL model training in the wild. A sentiment analysis (SA) model is used to demonstrate how FLSys effectively supports concurrent models, and it uses a dataset with 46,000+ tweets from 436 users. We conducted extensive experiments on Android phones and emulators showing that FLSys achieves good model utility and practical system performance.

The rest of the chapter is organized as follows. SSubsection 4.1.4 explains the design of FLSys, while Subsection 4.1.5 describes its prototype implementation. Subsection 4.1.10 presents the HAR model and data. Subsection 4.1.15 shows the experimental results. The chapter concludes in Subsection 4.1.21 with lessons learned and future work.

4.1 FLSys: Toward an Open Ecosystem for Federated Learning Mobile Apps

4.1.1 Contributors to this Dissertation

The prototype of FLSys was designed and implemented collaboratively with my colleague Xiaopeng Jiang. The author's contributions are the design of the training protocol, the implementation of the FL simulation process, and the implementation of cloud-side components in FLSys. Xiaopeng Jiang designed and implemented the mobile-side components, the deep learning model, the communication protocol, and the FL emulation process. The high-level design and the data pre-processing steps were designed and implemented collaboratively by the author and Xiaopeng Jiang. For a better understanding of the framework and my contribution, the whole framework is presented in this dissertation, including Xiaopeng Jiang's part.

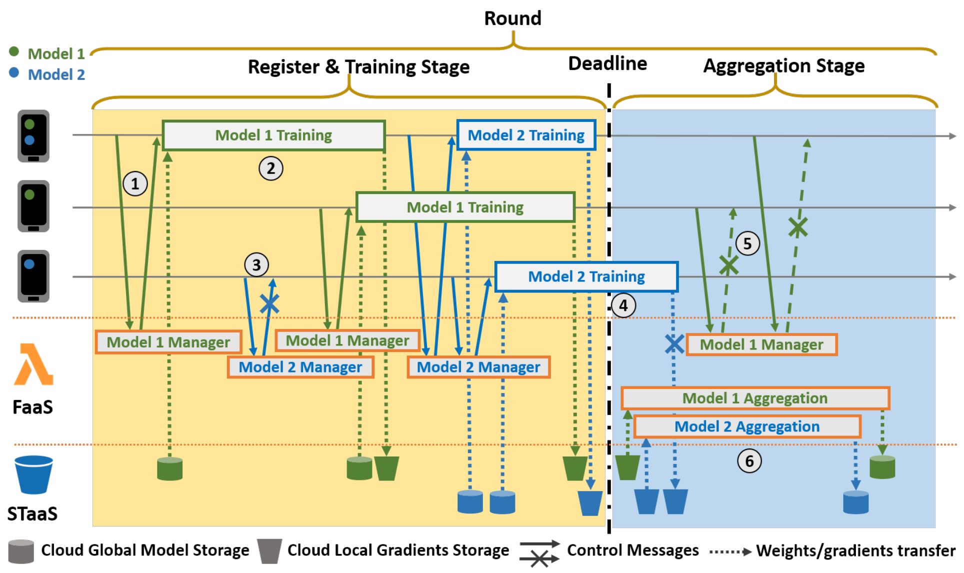
4.1.2 System Requirements

Our aim is to design and build an FL system that addresses the list of important questions mentioned in the Subsection 1.4.1. We use the HAR model, detailed in Subsection 4.1.10, to illustrate an entire category of FL models based on mobile [sensing] data collected in the wild. We extract five key requirements derived from this model and from other real-world FL applications, such as next word prediction, on-device search query suggestion [116], on-device robotic navigation [203], on-device item ranking [106], object recognition [112], sentiment analysis, etc., and utilize them to guide our FLSys design: (*R1*) Effective data collection: The data collection on the phone must balance resource consumption (e.g., battery) with sampling rates required by different models; (*R2*) Tolerate phone unavailability during training: Since the phones may sometimes be disconnected from the network or choose not to communicate to save battery power, the interaction between the phones and the cloud must tolerate such unavailability during federated training; (*R3*) Scalability: The cloud-based FL server of our system must be able to scale to large numbers of users in terms of both computation and storage; (*R4*) Model flexibility: The system must support different DL models for different application scenarios and different aggregation functions in the cloud; and (*R5*) Support for third-party apps: The system must provide programming support for third party apps to concurrently access different models on the phones.

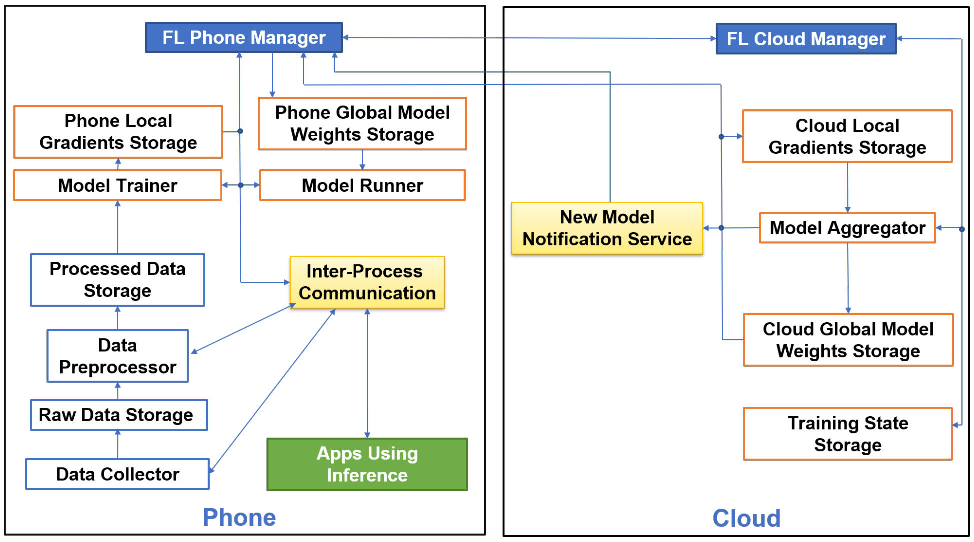
4.1.3 FLSys Overview

FLSys addresses requirements $R1 - R5$ synergistically. Figure 4.1a show its overall process of one training round, and Figure 4.1b shows its system architecture. These figures emphasize four novel contributions made in FLSys, compared with existing FL systems [106, 116, 129, 113, 128]: **(1)** FLSys allows the phones to self-select for training when they have enough data and resources; **(2)** FLSys has an asynchronous design (Figure 4.1a), in which the server in the cloud tolerates client failures/disconnections and allows clients to join training at any time. **(3)** FLSys supports multiple DL models that can be used concurrently by multiple apps; each phone trains and uses only the models for which it has subscribed; and **(4)** FLSys acts as a "central hub" on the phone to manage the training, updating, and access control of FL models used by different apps.

These features balance model utility with mobile device constraints, and can help create an ecosystem of FL models and associated apps. FLSys allows different developers to build FL models/apps and provides a simple way for users to take advantage of these apps, as it offers a unifying system for the development and deployment of FL models and apps that use these models. FLSys acts as common middleware layer for all these apps and models. The users just need to download/install the apps, and FLSys will take care of downloading/installing the FL models used by the apps, will perform FL training as needed, and will run FL inference on behalf of the apps.



(a) Asynchronous Protocol with Phone Self-Selection and Multiple Models



(b) FLSys Architecture

Figure 4.1 FLSys Asynchronous Protocol and Architecture. Typical operations: ① Phone Manager of Client #1 registers with the Cloud Manager of Model 1, which grants registration based on training settings. ② Phone Manager of Client #1 fetches up-to-date global model from a designated storage, trains it with local data, and uploads local gradients to a designated storage. ③ Phone Manager of Client #2 tries to register, but is denied. ④ Phone Manager of Client #2 successfully registers at a later time, but the training misses the deadline, thus its gradients upload is denied. ⑤ Clients #1 and #2 try to register during server aggregation and are denied. ⑥ Each model's Aggregator loads the gradients updates, aggregates them, and saves the aggregated model.

4.1.4 System Architecture

The architecture (Figure 4.1b) has two main components: *(1) FL Phone Manager*, which coordinates the FL activities on the phone; and *(2) FL Cloud Manager*, which coordinates the FL activities in the cloud. These two components work together to support the three phases of the FL operation: data collection and preprocessing, model training and aggregation, and mobile apps using inference. In the following, we describe each phase and explain how the system architecture satisfies the five system requirements.

Data Collection and Preprocessing The FL Phone Manager controls the data collection using one or multiple *Data Collectors*. A basic Data Collector is tasked with collecting data from one sensor at a given sampling rate. Such basic Data Collectors could be embedded in more complex ones to collect different types of data at the same time. It is important to have one app that coordinates data collection because having multiple apps collecting overlapping sets of data multiple times is inefficient. Having the FL Manager to coordinate the data collection also simplifies sensor access control.

To satisfy requirement *R1*, FLSys supports on-demand configuration of sensor types, sampling rates, and the period for data flushing from memory to storage. Each model informs the FL Phone Manager of the type of data and sampling rate it needs. In this way, the FL Phone Manager knows which Data Collectors to invoke and which sampling rates are needed. The FL Phone Manager balances sensing accuracy (i.e., high sampling rate) with resource consumption.

To regulate and keep such balance aligned with the user experience, the FLSys has three features: (1) include several built-in sampling rate settings, with empirical values from our experiences; and (2) collect key statistics of the data collection (e.g., CPU time consumed, battery life impact, etc.) and show them to the user, upon request; and (3) provide global level controls for the user to adjust the data collection behaviors, should the user feel that their experience is impacted by data collection.

The Data Collectors store the sensed data in the *Raw Data Storage* and inform the FL Phone Manager each time new data is added to the Raw Data Storage. For efficiency, the Data Collectors can buffer a certain amount of sensed data in memory before committing it to the storage. The FL Phone Manager can dynamically reconfigure the data flushing period that defines when the data is written to storage. Data Collectors set this data flushing period. Some models may use the raw data directly, while others may require additional processing. The FL Phone Manager decides when to invoke the model-specific *Data Processors*, which will store the data in the *Processed Data Storage*. This is a matter of policy and can be done any time new data is available in the Raw Storage Data or at a regular interval. The only constraint is to have all the data preprocessed before a new local model training operation.

Federated Training To satisfy requirement *R2*, we make two design decisions. First, FLSys allows the phones to self-select for training when they have enough data and resources. This is different from traditional FL architectures [106], where the server selects the phones to participate in training, which may not be available or may not have enough data or resources for training. Second, in FLSys, the communication between the phones and the cloud is asynchronous to cope with phone disconnections. The software at the cloud side is designed to tolerate missing messages from the phones. Overall, FLSys reduces communication overhead and increases client utility, at the expense of less control in the client sampling process, compared to [106].

In order to use a given model on the phone, the FL Phone Manager first registers the phone with the *FL Cloud Manager*. If the phone model and mobile OS are known to work with the model, the FL Cloud Manager registers the phone with the *New Model Notification Service*, which works as a Publish-Subscribe cloud service, and returns the subscription to the phone. This subscription allows the phone to receive asynchronous notifications when a new global model is available for download. The FL Phone Manager downloads the model at a time determined based on the model usage frequency and power settings.

The training for each model is done in rounds. The FL Cloud Manager decides the duration of a round, based on preferences associated with each model. For example, the server may start a new aggregation (i.e., by invoking the *Model Aggregator* for a certain model) when a given time interval has passed or when a certain number of local training updates have been received from the phones. The FL Phone Manager decides when to participate in training. This decision is done based on local policies that attempt to balance inference accuracy, the amount of input data for training, and the resources consumed during training. The intention to participate in training for a given model is conveyed by a message sent to the FL Cloud Manager. Based on the model preferences (e.g., amount of data, and the number of users in a training round), the server may decide to ask the phone to train for the model and to provide the FL Phone Manager with a URL to upload the results in the *Cloud Local Gradients Storage*. If there is a deadline for participation in the round, the FL Cloud Manager lets the FL Phone Manager know about it.

The FL Phone Manager invokes the *Model Trainer* for the given model and passes as parameter the location of the data in the Processed Data Storage. After the training is done, the Model Trainer stores the newly computed gradients in the *Phone Local Gradients Storage*. The FL Phone Manager decides when to upload these gradients to the Cloud Local Gradients Storage. The FL Cloud Manager will invoke the Model Aggregator for the model when the duration for the round expires or when enough updates have been uploaded. The Model Aggregator reads the updates from the Cloud Local Gradients Storage, computes the aggregated weights, and stores them in the *Cloud Global Model Weights Storage*. The intermediate training state is stored in the *Training State Storage* to provide lower I/O latency compared with the other types of cloud storage in our design. This is because FLSys needs frequent access to these data during training. Then, the Model Aggregator sends a notification via the New Model Notification Service to let the phones know that a new model version is available.

The cloud-side system satisfies requirement *R3*, as it can scale to large numbers of users due to its modular design that decouples computation, communication, storage, and notification services. The cloud elasticity features of each service allow different services to scale up or down according to the workload.

As we observe from the architecture, each model is managed individually by FLSys, and multiple models can co-exist both at the phones and the cloud. In the cloud, different models use independent cloud resources, which can be scaled independently. On the phone, independent model trainers and inference runners are responsible for different applications. The cloud contains all the models in the system, while each phone contains only the models for which it has subscribed. This modular design allows our system to satisfy requirement *R4*.

Mobile Apps Using Inference We decouple mobile apps that need inference on the phones from the models that provide the inference. This allows an app to use multiple models, while the same model can be used by multiple apps. FLSys provides an API and a library that can be used by third-party app developers to perform inference using DL models on the phone. In this way, the system architecture satisfies requirement *R5*. When an app needs an inference from a model, it sends a request to the FL Phone Manager using one of the OS IPC mechanisms. The FL Phone Manager then generates the input for the inference from the data stored in the Processed Data Storage of Raw Data Storage, and then invokes the *Model Runner* with this input. The Model Runner sends the result to the App using the IPC.

Model Concurrency Given the design of FLSys, both the FL Phone Manager and the FL Cloud Manager are able to handle multiple models concurrently. However, the meanings of concurrency are slightly different for each side. FL Cloud Manager needs to handle the aggregation of all models that are registered with it. Also there is the need to communicate to a potentially large number of clients for each model at the same time. FLSys handles this concurrency through services provided by the underlying cloud platform, which support concurrency by design. FLSys just needs to orchestrate the invocation of these services. The FL Phone Manager needs to handle concurrent training and inference. Our preliminary experiments on smart phones show parallel training of multiple models is very slow due to resource contention. It also affects the user experience on the phones. Therefore, we decided to train models sequentially. The FL Phone Manager can request to participate in training rounds for multiple models concurrently, but it locally decides a sequential order in which to train these models, based on parameters such as frequency of model usage by apps, the training round deadlines, and historical training latency for each model. Finally, the inference requests from the apps are executed as soon as they are received to maintain good user experience.

4.1.5 Prototype Implementation

We implemented an end-to-end FLSys prototype in Android and AWS cloud, which have been chosen because they are the market leaders for mobile OSs and cloud platforms, respectively. The FLSys design is general and it can be implemented in other mobile OSs and cloud platforms. The prototype implements all of the components described in the system architecture (Figure 4.1b). This subsection reviews the implementation technologies, the reasons for selecting them, and then focuses on the Android implementation and the AWS implementation of FLSys.

Deep Learning Framework We choose Deep Learning for Java (DL4J) as the underlying framework for the on-device DL-related operations (i.e., training and model execution) because it is the only mature framework that supports model training on Android devices. While the Model Aggregator in the cloud could be implemented using other DL technologies, for consistency, we implement it in DL4J as well. The models are stored as zipped JSON and bin files stored in folders on the phone and in AWS S3 buckets in the cloud.

On-device Communication For IPC among Android apps/services, we use Android Bound Service and Android Intent. A bound service can efficiently serve another application component because it does not run in the background indefinitely. Through IPC, the FL Phone Manager can provide third-party apps with an interface to request inference results without revealing the model or the data. Furthermore, it can communicate with the Data Collector.

Cloud Platform and Services We opt to utilize the Function-as-a-service (FaaS) architecture for our cloud computation. The core cloud components of FLSys are implemented and deployed as AWS Lambda functions [204]. We decided to choose FaaS for our implementation for five reasons. First, it matches our asynchronous, event-based design, as Lambda functions are triggered by events. Second, it provides fine-grained scalability at the function level; therefore leading to less resource consumption in the cloud. Furthermore, computation and storage are scaled automatically and independently by the cloud platform. Third, unlike other cloud platforms, it does not require running virtual machines when no computation is necessary; this saves additional resources and reduces cost. Fourth, FaaS simplifies the development and deployment of our prototype because it does not require software installation, system configuration, etc. Fifth, different functions can be implemented in different programming languages making the implementation even more flexible.

Lambda functions are triggered in different ways in our prototype. We use the AWS API Gateway to define and deploy HTTP and REST APIs. For instance, we create a REST API to relay clients' requests to participate in the FL training to the Lambda function that handles these requests. We also use the AWS EventBridge to define rules to trigger and filter events for Lambda functions.

FLSys uses a number of cloud services for storage, authentication, and publish-subscribe communication. For model storage, validation datasets, and FL Cloud Manager configuration files, we use AWS S3, which offers a reliable and cost-effective solution for data accessed infrequently. More importantly, AWS S3 buckets can be accessed directly by phones, which simplifies the asynchronous communication in FLSys. To authenticate clients and allow them to upload and download models from the AWS S3, FLSys uses Identity Pool in AWS Cognito. To store data that is accessed frequently, such as training round states and model states, we use AWS DynamoDB, a reliable NoSQL database. AWS SNS is utilized in conjunction with the Google FCM to notify clients when newly trained models are ready. The use of a Google Cloud service in our AWS implementation was necessary in order to push notifications directly to apps on the phones when a new global model is ready in the cloud.

4.1.6 Phone Implementation

The phone implementation (left-side of Figure 4.1b) consists of three apps: a FL Phone Manager, a HAR Data Collector, and a Testing App used to test model inference.

Data Collector We implemented a HAR Data Collector app designed for long-term and battery efficient data collection. To that end, sensor values are not collected at an enforced fixed high frequency, but are instead collected independently through Android listeners whose actual frequency is variable, determined by the underlying OS. This is appropriate for data collection in the wild. In our experience, this tends to be much friendlier to the performance and battery life of the user devices, lowering the risk that a user abandons FLSys prematurely due to concerns about how it is affecting their device resources. Furthermore, users are given the option to pause or stop data collection of all or a subset of sensors in case they have resource consumption or privacy concerns. For simplicity, the raw data and the processed data are stored as files.

FL Phone Manager The FL Phone Manager app decides to initiate an on-device training round based on evaluating a Ready To Config policy (RTCp). We implemented a simple policy to check if the phone is charging and is connected to the network before declaring its availability for training. If yes, it sends a Ready To Config message (RTCm) to the FL Cloud Manager. RTCm is implemented as an HTTP request with JSON payload and is sent to a REST API URL in AWS. The FL Cloud Manger either accepts or denies the phone's participation in this training round, based on a simple Accept/Deny for Training policy (A/DFTp) that checks the phone model and client identity.

The phone is accepted for a round of training when it receives an Accept For Training message (AFTm). AFTm contains the information of the AWS S3 locations from where to download the latest global model weights and where to upload the local gradients. The message also contains the deadline for this training round's completion. The FL Phone Manager evaluates a Start To Train policy (STTp) based on the available device resources and the round's deadline to determine whether to actually perform the on-device training for this round or not.

The FL Phone Manager will create the corresponding Model Trainer if it decides to train. The Model Trainer is implemented with Android native *AsyncTask* class to ensure the trainer is not terminated by Android, even when the app is idle. *AsyncTask* also enables multiple trainers to train in the background. Once the training is complete, the Model Trainer uploads the local gradients to the corresponding AWS S3 location.

Model inference is implemented as a background service with Android Interface Definition Language (AIDL), and it gets inference requests from third-party apps. When such a request is received, the FL Phone Manager uses the current sensor data from the Data Collector as input for the model, runs the inference, and responds to the third-party apps with the inference results.

Testing App We implemented a simple testing App to test model inference. The App uses *AidlConnection* to interface with the FL Phone Manager. Let us note that the App itself does not access any data or model.

4.1.7 Cloud Implementation

The cloud implementation (right-side of Figure 4.1b) consists of two main components: FL Cloud Manager and Model Aggregator.

FL Cloud Manager The FL Cloud Manager is implemented as a series of Lambda Functions (FaaS service in AWS). When starting a training round, it reads a configuration file and determines the deadline for the round (i.e., the time when the round must finish). During the period between the start time and the deadline, the FL Cloud Manager accepts or denies clients' requests for training (RTCm). When the deadline is reached, the FL Cloud Manager executes the Model Aggregator according to the Start for Aggregation policy (SFAp). The current policy checks if enough clients have submitted their local gradients in the AWS S3 (a configurable parameter). Then, the Lambda function implementing the FL CLoud Manager schedules an event for itself to perform the next training round and terminate. The training process stops when the pre-defined number of rounds is achieved, or the desired performance (model accuracy) is achieved, if the model developers provided a validation dataset.

Model Aggregator For implementation simplicity, the Model Aggregator uses the federated average technique [205], with the assumption that each client contributes equally to the global model in each training round. When it is invoked, it loads the uploaded local gradients, and aggregates their gradients to the global model of this round. Once the global model is updated, the Model Aggregator invokes AWS SNS to notify clients that they can download the newly aggregated model. Note that the Model Aggregator is called dynamically through reflection, such that different aggregation functions can be dynamically swapped.

4.1.8 Asynchronous Federate Averaging Implementation

Algorithm 4.1 shows the pseudo-code of our asynchronous federated averaging process. The algorithm consist of three procedures, which execute asynchronously. “ClientLoop” (lines 1-12) runs at clients and executes a round of training (lines 7-12), if the phone self-selects for training and the cloud accepts it (lines 1-6). “ServerRTCmHandler” (lines 13-17) is a part of the FL Cloud Manager and decides whether a phone is accepted for training. “ServerLoop” (lines 18-40) also runs at the FL Cloud Manager. It performs the aggregation of local gradients and controls the progression of training. The clients participating in a training round must submit their local gradients before the deadline for the round expires. When the deadline comes, the procedure first evaluates the Start for Aggregation policy, which checks whether there are enough local gradient updates in order to preform aggregation. If yes, the aggregation is preformed (line 24-26); if not, this round is aborted, but the uploaded gradient updates will be carried to the next round. After aggregation, the procedure may check against pre-defined conditions to decide whether this aggregation outcome should be accepted or not (lines 27-30). Finally, the procedure checks if a new round should be started by evaluating the Start New Round policy. If a new round is to be started, a new deadline will be set (lines 33-36). Otherwise, the procedure terminates.

4.1.9 FLSys Setup Workflow

By design, FLSys acts as a service provider that handles multiple FL models with minimum input from the users. The setup procedures for FLSys are divided into two stages. The first stage involves the FL Cloud Manager and the app developers without user involvement. The second stage involves the FL Phone Manager and the mobile apps that use FL models, and it requires minimum user involvement. The FL Cloud Manager is deployed before the first stage, and the FL Phone Manager should be installed on the user’s device before the

Algorithm 4.1 AsynFedAveraging

```
1: procedure ClientLoop( )
2:     while true do
3:         $readyToConfig \leftarrow$ evaluateReadyToConfigPolicy($powerState$, $wifiState$,...)
4:         if $readyToConfig$ then
5:             $response \leftarrow$ sendRTCm( )
6:             if $response$ == “AFT” then
7:                 $\mathcal{B} \leftarrow$ sampling($\mathcal{D}_{\mathcal{L}}$)
8:                 $\theta_l \leftarrow \theta^t$
9:                 for batch $b \in \mathcal{B}$ do
10:                    $\theta_l \leftarrow \theta_l - \eta \nabla \mathcal{L}(\theta_l; b)$
11:                $\Delta_l \leftarrow \theta_l - \theta^t$
12:                uploadClientGradients($\Delta_l$)

13: procedure ServerRTCmHandler($RTCm$)
14:     if evaluateAcceptForTrainingPolicy($RTCm$) then
15:         returnResponse(“AFT”)
16:     else
17:         returnResponse(“DFT”)

18: procedure ServerLoop( )
19:     $deadlineTriggered \leftarrow false$
20:     setupDeadline( ) ($deadlineTriggered \leftarrow true$ when triggered)
21:     while true do
22:         if $deadlineTriggered$ then
23:             if evaluateStartForAggregationPolicy( ) then
24:                 $\{\Delta_1, ...\Delta_k\} \leftarrow$ loadClientGradients( )
25:                 $\Delta^t = (\sum_k \Delta_k)/k$
26:                 $\theta^{t+1} \leftarrow \theta^t + \gamma \Delta^t$
27:                 if isRoundAcceptable( ) then
28:                     acceptRound($\theta^{t+1}$)
29:                 else
30:                     abortRound( )
31:             else
32:                 abortRound( )
33:             if evaluateStartNewRoundPolicy( ) then
34:                 startNewRound( )
35:                 $deadlineTriggered \leftarrow false$
36:                 setupDeadline( )
37:             else
38:                 stopTraining( )
39:         else
40:             wait( )
```

second stage. To illustrate these stages, let us briefly explain the setup workflow using the HAR app as an example.

In the first stage, the developers of the HAR app need to register the model with the FL Cloud Manager. If the model is developed by the app developers, then the developers need to provide the FL model to be trained and the training plan (e.g., training frequency, number of rounds, number of participants in a round, etc.) to register the app. If the developers plan to use an existing FL model, then they need to specify which model to use to register the app. After registration, an unique key for the authentication between the app and the FL Phone Manager in the second stage will be provided.

The second stage is typically triggered during the installation process of the HAR app on the user's device. The app will communicate with the FL Phone Manager and authenticate itself using the aforementioned unique key. Once the app is successfully authenticated, the FL Phone Manager will perform a series of operations and eventually become ready to serve the FL model for the app. These operations including: (1) Register the phone with the FL Cloud Manager; (2) Set up communication channels with the app; (3) If the model does not exist on the phone, it downloads the model specified by the app and the training plan from the FL Cloud Manager; If the model already exists on the phone, it only establishes the connection between the app and that model; and (4) Set up the local training schedule and notify the user. After the second stage, the FL model that the HAR app needs is installed on the phone, ready for inference and training. The training plan can be adjusted by the developers through the FL Cloud Manager, if the developers own the model. User-experience related parameters can be adjusted by the user through the FL Phone Manager.

4.1.10 HAR-Wild: Data, Model, and Training

We co-designed FLSys with a HAR model, which was used to extract the main requirements for FLSys, and then to demonstrate the efficiency and effectiveness of FLSys. To show that FLSys works with different concurrent models, we also implemented and evaluated a sentiment analysis (SA) model, as described in Subsection 4.1.15. In this Subsection, we describe the HAR dataset, our HAR-Wild model, and its training algorithm using data augmentation to deal with non-IID data in the wild.

Table 4.1 Total Number of Minutes for Each Labeled Activity.

Label	Total minutes	Label	Total minutes
sitting	862544.50	table	864904.00
walking	158087.40	mounted	49440.29
driving	38013.98	strap	2485.22
lying	488596.70	vehicle_car	72121.55
cycling	2589.50	vehicle_train/subway	355.24
workout_gym	3649.47	vehicle_motorcycle	88.84
workout_running	2212.69	home	1171968.00
workout_others	16500.13	outside	41886.80
palm	511092.20	travel	14094.12
bag	6446.19	elevator	924.09
pocket	99557.67	office	17562.08

4.1.11 Data Collection

Although there are good HAR datasets publicly available, e.g., WISDM [123], UCI HAR [124], they are not representative for real-life situations. These datasets were collected in rigorously controlled environments on standardized devices and controlled activities, in which the participants only focused on collecting sensor data with a usually high and fixed sampling rate frequency, i.e., 50Hz or higher.

Thus, given our goal to test FLSys with data collected in the wild, we have used our Data Collector, described in Subsection 4.1.6, to collect data from 116 users at two universities. We opt to collect data from university students as subjects for the following reasons: **(1)** University students should have relatively good access to the smartphones and related technologies; **(2)** University students should be more credible and easier to be motivated than other sources (e.g., recruiting test subjects on crowd-sourcing websites); and **(3)** It will be easier for our team to recruit and distribute rewards to students. The data collection was approved by the IRBs at both universities.

The students collected data from April 1st to August 10th, 2020. Each user provided mobile accelerometer data and labels of their activities on their personal Android phones. We provided labels in five categories for participants to choose form: “Walking,” “Sitting,” “In Car,” “Cycling,” and “Running”. The phones were naturally heterogeneous, and the daily-life activities were not constrained by our experiments.

Therefore, we collected a novel HAR dataset in the wild that is different from the existing datasets in the following three aspects: **(1)** The sensors’ sampling rates vary from time to time and from user to user, due to battery constrains, device variability, and usability targets; **(2)** The same basic activity will generate different signals since different users will have different habits of carrying smart phones; **(3)** Label distributions are not just biased, but vary significantly among users.

The data and labels for each user are uploaded to the backend server that runs on a secure AWS instance fully managed by Unknot.id and NJIT. Upon registration, each account will get a unique random identifier for the user. The raw data and labels are associated only with this identifier to ensure user anonymity. The total size of the accumulated raw data is about 1 TB.

4.1.12 Data Preprocessing

Our data processing consists of the following steps: **(1)** Any duplicated data points (e.g., data points that have the same timestamp) are merged by taking the average of their sensor values; **(2)** Using 300 milliseconds as the threshold, continuous data sessions are identified and separated by breaking up the data sequences at any gap that is larger than the threshold; **(3)** Data sessions that have unstable or unsuitable sampling rates are filtered out. We only

keep the data sessions that have a stable sampling rate of 5Hz, 10Hz, 20Hz, or 50Hz; **(4)** Data sessions are also filtered with the following two criteria to ensure good quality: (a) The first 10 seconds and the last 10 seconds of each data session are trimmed, due to the fact that users were likely operating the phone during these time periods; (b) Any data session longer than 30 minutes is trimmed down to 30 minutes, in order to mitigate the potential inaccurate labels due to users' negligence (forgot to turn off labeling); and
(5) We sample data segments at the size of 100 data points with sliding windows. Different overlapping percentages were used for different classes and different sampling rates. The majority classes have 25% overlapping to reduce the number of data segments, while the minority classes have up to 90% overlapping to increase the available data segments. The same principle is applied to sessions with different sampling rates. We sample 15% of data for testing, while the rest are used for training. Details are shown in Table 4.2.

Data Normalization In our models, the accelerometer data is normalized as $x \in [-1, 1]^d$ to achieve better model utility. We compute the mean and variance of each axis (i.e., X, Y, and Z) using only training data to avoid information leakage from the training phase to the testing phase. Then, both training and testing data are normalized with z-score, based on the mean and variance computed from training data. Based on this results, we choose to clip the values in between $[min, max] = [-2, 2]$ for each axis, which covers at least 90% of possible data values. Finally, all values are linearly scaled to $[-1, 1]$ to finish the normalization process:

$$x = 2 \times [\frac{x - min}{max - min} - 1/2] \tag{4.1}$$

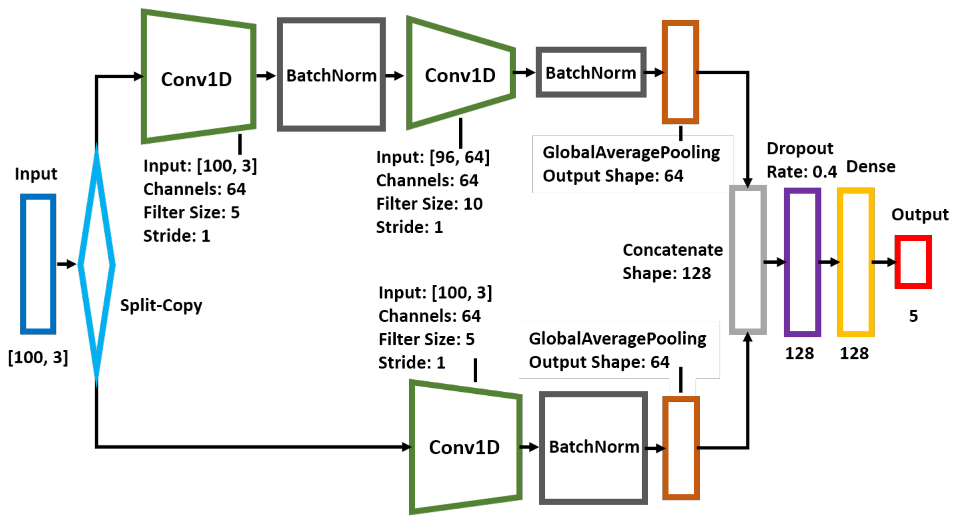


Figure 4.2 HAR-Wild model architecture.

4.1.13 Model Design

The design of our HAR-Wild model has two requirements: low computational complexity and small memory footprint. Satisfying these requirements ensures the model can work efficiently on phones. Figure 4.2 shows our model architecture. For a low computation complexity, HAR-Wild is based on CNN (instead of RNN, e.g., LSTM) and tailored to work well on mobile devices. In addition, instead of using data from multiple sensors, HAR-Wild can achieve comparable results with several baseline approaches by using only accelerometer data, which makes the training faster.

The accelerometer data are processed into data segments of shape $[3, 100]$, indicating 100 data points of 3 axis: X, Y, and Z. We leverage the recipe of ResNet model [206] into a small-size model, by using the processed accelerometer data as input of (1) a sequence of a 1D-CNN - a Batch Norm - a 1D-CNN - a Batch Norm - a Flatten layer, and (2) a sequence of a 1D-CNN - a Batch Norm - a Flatten layer. The two flatten layers are concatenated before feeding them into a sequence of a Drop Out layer - a Dense layer - and an Output layer. By doing so, HAR-Wild can memorize and transfer the low level latent features learned from the very first 1D-CNN, directly derived from the input data, to the output layer for better classification. We use Global Average Pooling [207] given its small memory footprint,

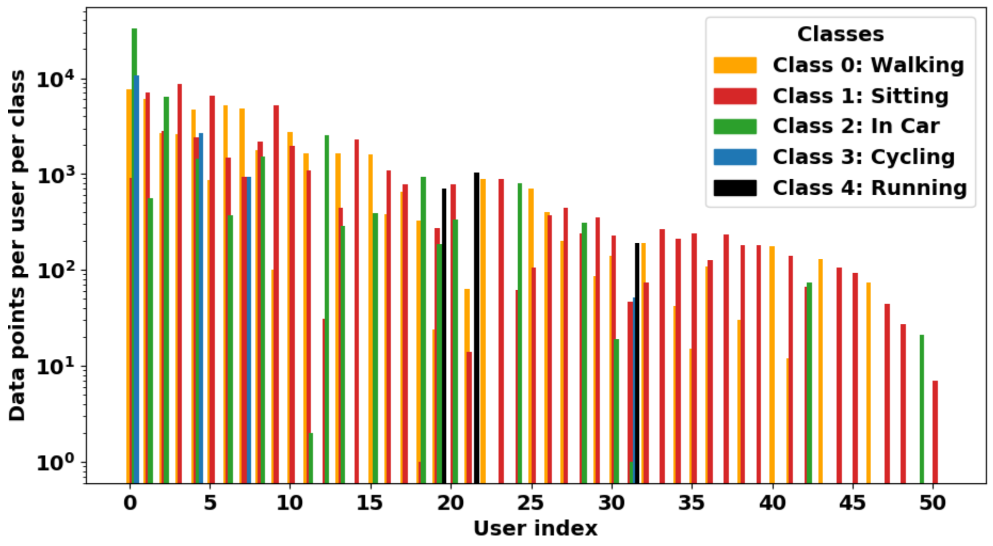


Figure 4.3 Number of data points of each class for each user.

instead of the popular Local Max/Average Pooling [208]. A small-size model is expected to perform better on data collected in the wild, since the data will likely have more distribution drift, increasing the chance of model overfitting on large-size models.

4.1.14 HAR-Wild Async Augmented Training

The performance of FL models is negatively affected by non-IID data distribution [133, 109, 127], and we observed this to be true for HAR-Wild as well. Figure 4.3 shows the data distribution of HAR-Wild. To address this problem, we leverage data augmentation training [209] and tailor it to mitigate the distortion in computing gradients at client-side by balancing the client data with a small number of augmentation data samples without an undue computational cost.

The pseudo-code for HAR-Wild Asynchronous Augmented Learning is described in Algorithm 4.2. This algorithm is integrated in Algorithm 4.1 by replacing lines 7-12 from Algorithm 4.1 with the AugmentedGradients procedure in Algorithm 4.2. Before the whole training process starts, the FL Cloud Manager executes the procedure Init (lines 1-3, Algorithm 4.2), which first collects a small pool of random samples for each class that will be used for data augmentation (line 2). These data can be collected from a small number of volunteers or controlled users who share IID data with the FL Cloud Manager in FLSys. The augmentation data pool could also come from publicly available datasets. Then, the augmentation data pool $\mathcal{A}$ is delivered to each client (line 3). In each training round, each client (i.e., phone) randomly samples the augmentation data (line 8). Then, the sampled augmentation data $\mathcal{D}_{\mathcal{A}}$ will be combined with the local data $\mathcal{D}_{\mathcal{L}}$ (line 10, Concatenate($\mathcal{D}_{\mathcal{A}}$, $\mathcal{D}_{\mathcal{L}}$)) to compute the local gradients (lines 11-13, LocalTraining). The local gradients are then sent to the cloud for the asynchronous average aggregation and model update (line 14).

In order to deliver the augmentation data to the clients (line 3), we consider two objectives: **(i)** privacy-preserving, and **(ii)** communication efficiency. One naive approach is to send data to augment the missing classes at the clients in each training round, since the local missing data can change over time. In this approach, the FL Cloud Manager needs to know which classes are missing for each client in each training round. This could increase the communication cost and significantly increase data privacy risk; since the cloud learns certain aspects of the user behavior based on the classes that miss data over time. To achieve both privacy-preserving and communication efficiency, the approach implemented in our FLSys (Algorithm 4.2) first delivers the entire augmentation data to every client only once at the beginning of the training process. Then, the clients use only the data necessary to augment their missing data in each training round. Given the small size of the augmentation data, the overhead of also sending unnecessary augmentation data is minimal, without causing extra data privacy risk.

4.1.15 Evaluation

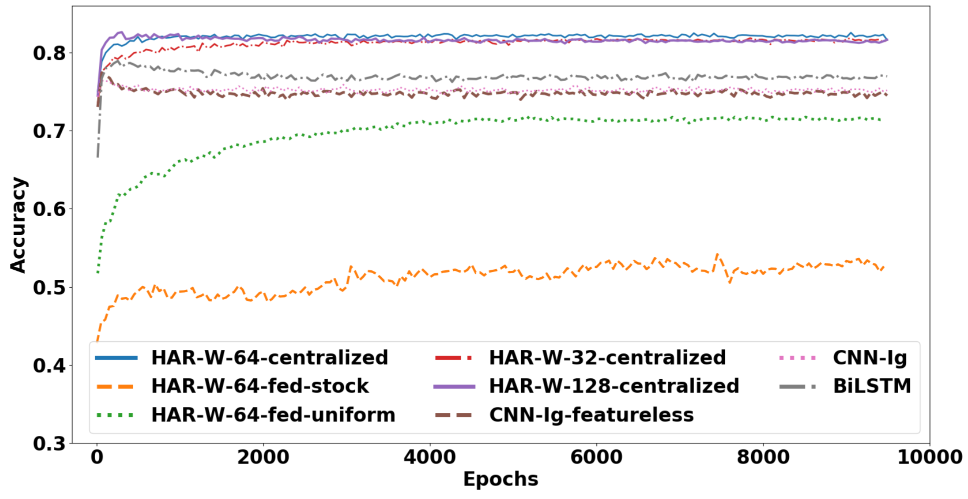
The evaluation has two main goals: **(i)** Analyze the performance of the two FL models, HAR-Wild and sentiment analysis (SA), to understand if they perform similarly with their centralized counter-parts; **(ii)** Quantify the system performance of FLSys with HAR-Wild and SA on Android and AWS. In terms of system performance, we investigate energy

Algorithm 4.2 HAR-Wild Asynchronous Augmented Learning

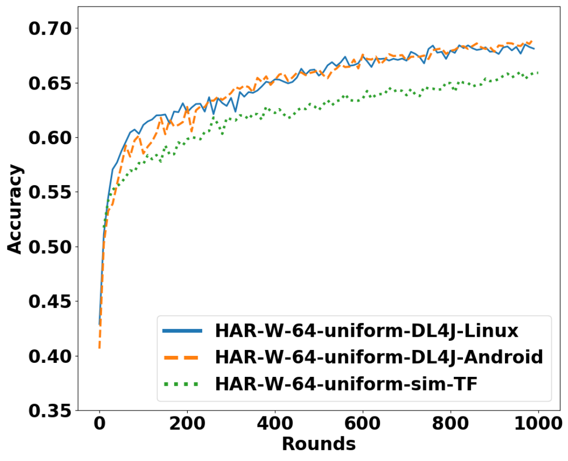
```
 1: procedure Init($clients$)
 2:     augmentation pool $\mathcal{A} \leftarrow$ sampleAugmentData($clients$)
 3:     deliverAugmentPool($\mathcal{A}$, $clients$)
 4: procedure AugmentedGradients(Round $t$, Client $i$)
 5:     Augmentation data pool $\mathcal{A}$
 6:     Local data pool $\mathcal{L}_i$
 7:     $\theta_l \leftarrow \theta^t$
 8:     augmentation data $\mathcal{D}_{\mathcal{A}}$ = sampleAugmentData($\mathcal{A}$)
 9:     local data $\mathcal{D}_{\mathcal{L}}$ = sampleData($\mathcal{L}_i$)
10:     training data $\mathcal{D}_{\mathcal{T}}$ = concatenate($\mathcal{D}_{\mathcal{A}}$, $\mathcal{D}_{\mathcal{L}}$)
11:     for batch $b \in \mathcal{D}_{\mathcal{T}}$ do
12:         $\theta_l \leftarrow \theta_l - \eta\nabla\mathcal{L}(\theta_l; b)$
13:     $\Delta_l \leftarrow \theta_l - \theta^t$
14:     uploadClientGradients($\Delta_i$)
```

efficiency and memory consumption on the phone, system tolerance to phones that do not upload local gradients, and FL aggregation scalability in the cloud. We also study the overall response time for third party apps that use FLSys on the phone. For model evaluation, we use Accuracy, Precision, Recall, and F1-score metrics. For system performance, we report execution time and memory consumption for both the phones and the cloud, and battery consumption on the phones.

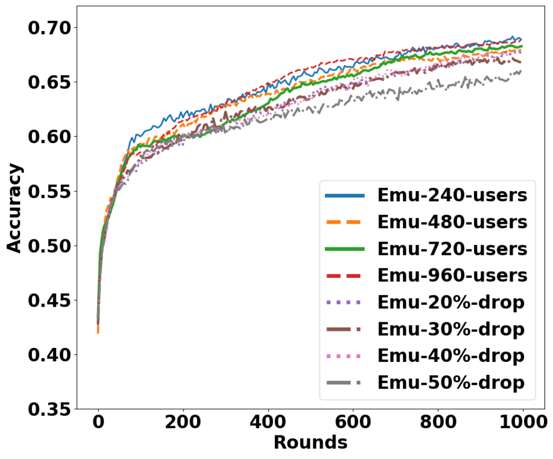
Most of the evaluation is done in the context of HAR-Wild, which illustrates a typical FL model based on mobile sensing data. However, to demonstrate that FL works for different models, we also show results for the SA model. The rest of the Subsection is organized as follows: Subsection 4.1.16 evaluates HAR-Wild under different scenarios. Subsection 4.1.17 describes the SA model and shows its performance. Subsection 4.1.18 shows the HAR-Wild performance over the FLSys prototype. Since we did not have enough phones, we show the results using Android emulators to replay each user's data. Subsection 4.1.19 presents scalability and fault-tolerance results for HAR-Wild over FLSys in AWS. Finally, Subsection 4.1.20 presents results for HAR-Wild and SA over FLSys on two types of Android phone models.



(a) Centralized and Simulated FL



(b) Android Emulated FL vs. Simulated FL



(c) Linux FL Emulation

Figure 4.4 HAR-Wild accuracy under different settings.

4.1.16 HAR-Wild Model Evaluation

Table 4.2 shows the basic information of our collected dataset used for all HAR-Wild experiments. Some of the users have very limited numbers of labeled activities; thus, we select data from 51 users who labeled a reasonable amount of samples.

We perform centralized and simulated evaluation to assess HAR-Wild's utility compared with several baselines. Centralized training works as an upper bound performance for FL models. In addition, it allows us to fine-tune the model's hyper parameters. The

Table 4.2 Number of Samples in the Dataset for 51 Users

Type	Class 0	Class 1	Class 2	Class 3	Class 4
	Walking	Sitting	In Car	Cycling	Running
Training	48855	51499	49185	14281	1920
Testing	8514	8828	8595	2514	319

evaluation includes three variants of HAR-Wild: *HAR-W-32*, *HAR-W-64*, and *HAR-W-128*, which have the numbers of convolution-channels set to 32, 64, and 128. We derive two versions of our *HAR-W-64* model in simulated FL with (*HAR-W-64-uniform*) and without (*HAR-W-64-stock*) the data augmentation. The augmentation data, consisting of 640 samples of each class, is fixed and shared with all clients.

Baseline approaches To demonstrate that HAR-Wild is competitive with respect to state-of-the-art HAR models, we consider two baseline models: (1) *Bidirectional LSTM* with 3-axial accelerometer data as input. This is a typical model for time-series data, and we fine-tune it based on grid-search of hyperparameters; and (2) The CNN-based models proposed by Ignatov [120], with additional features (*CNN-Ig*) and without additional features (*CNN-Ig_featureless*) using recommended settings in [120]. For a fair comparison, we used TensorFlow implementations for all models. Table 4.3 shows all the hyper-parameters and model configurations.

Results Figure 4.4a shows that HAR-Wild models outperform the baseline approaches. On average, HAR-W-64 performs best and achieves 82.49% accuracy compared with 78.68%, 76.39%, and 77.08% of the BiLSTM, CNN-Ig and CNN-Ig-featureless. Our HAR-Wild models also achieve the best performance in all the other metrics (Table 4.4). Overall, HAR-W-64 (60,613 trainable weights) has the best trade-off among model accuracy, convergence speed, and model size, and we use it in all the following experiments for HAR-Wild.

Table 4.3 Model Settings of HAR-W and Baselines

Model	Optimizer	Other key parameters
HAR-Wild (centralized)	Adam	LR=0.0005, dropout_rate=0.4, batch_size=1024, Sampling: Same as class distribution
HAR-Wild (sim-FL)	Adam	client_LR=0.005, server_LR=1.0, dropout_rate=0.4, batch_size=128, Sampling: $[50, 100]$ samples per class, $[15, 30]$ augment samples per class
HAR-Wild (FLSys)	Adam	client_LR=0.005, server_LR=1.0, dropout_rate=0.4, batch_size=64, Sampling: $[50, 100]$ samples per class, $[15, 30]$ augment samples per class
CNN-Ig (centralized)	Adam	LR=0.0005, dropout_rate=0.05, batch_size=1024 Sampling: Same as class distribution
BiLSTM (centralized)	Adam	LR=0.0005, dropout_rate=0.2, batch_size=1024 Sampling: Same as class distribution

In the simulated FL, we replay the data collected in the wild for each user. HAR-Wild trained with FL and data augmentation achieves 71.8% accuracy, which is about 10% less than the accuracy of the centralized-trained HAR-Wild. This is the cost of privacy-protection provided by FL. We notice the substantial benefits of data augmentation, since HAR-W-64-uniform outperforms HAR-W-64-stock (without data augmentation), in all metrics.

Table 4.4 HAR-Wild vs. Baselines: Macro-Model Performance

Model	Accuracy	Precision	Recall	F1-score
HAR-W-32-centralized	0.8186	0.8486	0.8360	0.8409
HAR-W-64-centralized	0.8249	0.8512	0.8354	0.8428
HAR-W-128-centralized	0.8262	0.8529	0.8449	0.8484
BiLSTM	0.7868	0.8074	0.7831	0.7941
CNN-Ig	0.7639	0.7970	0.7715	0.7834
CNN-Ig_featureless	0.7708	0.8004	0.7779	0.7878
HAR-W-64-fed-stock	0.5368	0.3828	0.3569	0.3190
HAR-W-64-fed-uniform	0.7181	0.7464	0.7419	0.7378

4.1.17 SA Model Evaluation

FLSys is designed and implemented to be flexible, in the sense that the training and inference of multiple models can run concurrently. On the server, different applications use independent AWS resources. On the phone, independent model trainers and inference runners are responsible for different applications. This subsection showcases the training performance of the SA model, a text analysis application that interprets and classifies the emotions (positive or negative) from text data. With the inferred emotions of mobile users' private text data, a smart keyboard may automatically generate emoji to enrich the text before sending.

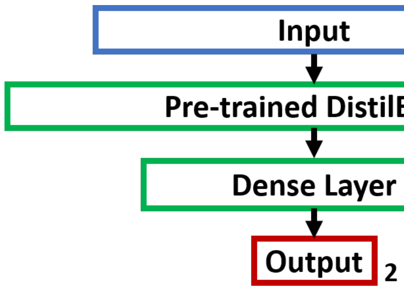


Figure 4.5 SA model architecture.

We build the SA model specifically for tweet data. We use the FL benchmark dataset Sentiment140 [1], which consists of 1,600,498 tweets from 660,120 users. We select the users with at least 70 tweets, and this sub-dataset contains 46,000+ samples from 436 users. Figure 4.5 shows our SA model architecture. We first extract a feature vector of size 768 from each tweet with DistilBERT [210]. Then, we apply two fully connected layers with relu and softmax activation, respectively, to classify the feature vector into positive or negative. The number of hidden states of the first fully connected layer is set to 128 to balance the convergence speed and model size. In the FL version of the model, 5% of the users are used for data augmentation, and the rest of the users follow 4:1 train-test split.

Results While the reference implementation associated with this benchmark dataset reached 70% accuracy [211] using 100 users with stacked LSTM in FL simulation, our SA model achieves superior performance, as shown in Table 4.5. Centralized learning achieves 81% accuracy, while FL achieves 79% accuracy (an acceptable drop).

4.1.18 HAR-Wild over FLSys Emulation Performance

This set of experiments tests FLSys with HAR-Wild by running the actual phone code in Android emulators. However, since Android emulation is slow and costly, we run several larger-scale experiments with the same DL4J algorithms and functions in Linux,

[1]http://help.sentiment140.com/home Retrieved on June 15, 2021

Table 4.5 SA Model Performance per Class in FLSys Training

	Class	Accuracy	Precision	Recall	F1-score	Support
CL	negative	0.81	0.75	0.69	0.72	3159
	positive		0.84	0.88	0.86	5746
FL	negative	0.79	0.73	0.64	0.68	3159
	positive		0.81	0.87	0.84	5746

which is much faster; this enables us to conduct larger-scale experiments. This subsection demonstrate that FLSys and HAR-Wild can work well in real-life, where they are deployed on Android phones.

All the phone components of the prototype, except for Data Collector and Data Preprocessor, run in the emulators. The cloud part of the prototype runs in AWS. The Android emulators run on top of virtual machines (VMs) in Google Cloud, as AWS does not support nested virtualization. We run 10 VMs in Google Cloud, and each VM has 16 vCPUs and 60GB memory. On each instance, we run 4 Android v10 emulators from AVD manager in Android Studio. Each emulator is loaded with 3 users' data files, and each file is sampled twice as different clients. In each round, each Android emulator participates in training on behalf of a few clients. We set the deadline for the round in the FL Cloud Manager to 6 minutes.

Results Figure 4.4b shows that HAR-Wild with 64 clients emulation in both Android and Linux on FLSys achieve comparable accuracy with the simulated FL with TensorFlow, i.e., 69.07%, 68.50%, and 66.00%. Figure 4.4c shows the results of HAR-Wild with higher number of clients (up to 960) using Linux emulations. The client data was over-sampled from the original 51 users. HAR-Wild model achieves up to 69.17% accuracy, and more clients help the model converge quicker with better performance.

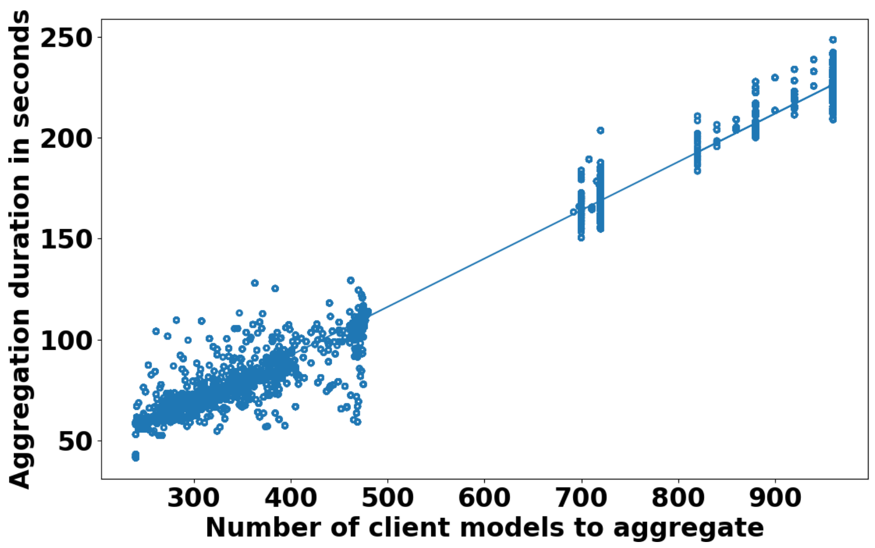


Figure 4.6 Aggregation time and participating clients

4.1.19 Fault Tolerance and Scalability

Fault Tolerance In daily life, some clients may fail to upload a trained model to the FL Cloud Manager due to network or computation issues. This set of experiments verifies the fault tolerance of FLSys in terms of model performance as a percentage of clients dropped out randomly in each round. Figure 4.4c shows the accuracy of HAR-Wild with up to 50% clients dropping out randomly from 480 clients in each round. With 1,000 rounds of training, the accuracy is reduced by at most 3.11%. This is a promising result showing that FLSys can tolerate reasonably large dropout rates during training.

Scalability As discussed in Subsection 4.1.5, computation and storage scale independently in the cloud for FLSys. This set of experiments verifies the scalability of FLSys across training rounds. The only FL function that may be computationally intensive in the cloud is the Model Aggregator. Figure 4.6 shows the Model Aggregator in AWS scales linearly with the number of participating clients. We also observe that the aggregation of 960 clients generally finishes in less than 4 minutes. By interpolating these results and given the current 15 minutes execution time limit of an AWS Lambda process [204], the FLSys prototype (with single-threaded aggregator) can handle up to 3,600 clients, which is a sufficient number of clients, per training round. This number can be multiplied substantially by implementing both thread-level and process-level parallelization to handle real-world traffic volume.

4.1.20 FLSys Performance on Smart Phones

We benchmarked FLSys with HAR-Wild and SA on Android phones using a testing app to evaluate training and inference performance. We also assessed the resource consumption on the phones. We used two phones with different specs (Google Pixel 3 and Pixel 3a).

Table 4.6 Model Performance per Class in FLSys Training

Model	Class	Accuracy	Precision	Recall	F1-score
HAR-W-64	0	0.6907	0.7003	0.6628	0.6810
	1		0.5922	0.8655	0.7032
	2		0.8606	0.5443	0.6668
	3		0.8324	0.6450	0.7268
	4		0.6682	0.9028	0.7680

Training Performance Table 4.7 shows the training time and the resource consumption on the phones. The training time is recorded by training 650 samples for 5 epochs for HAR-Wild, and 100 samples for 5 epochs for SA, which are the optimum scenarios determined in Subsection 4.1.18. Foreground training is done while leaving the screen on, and it uses the full single core capacity. It provides a lower bound for the training time. However, in reality, we expect training to be done in the background, either on battery or on charger. As in practice, other apps or system processes working in background may interfere with training. We take 10 measurements for each benchmark, and report the mean and standard deviation.

Training for one round is fast on the phones. The foreground training time on the more powerful phone, Pixel 3, is just 0.7 min for HAR-Wild, and 0.22 min for SA. The

Table 4.7 Training Resource Consumption and Latency

Model	HAR		SA	
Phone	Google Pixel 3a	Google Pixel 3	Google Pixel 3a	Google Pixel 3
Maximum RAM Usage (MB)	156	165	128	136
Foreground Training Time Mean/SD (min)	1.23/0.01	0.70/0.06	0.33/0.005	0.22/0.002
Background Training Time on Charger Mean/SD (min)	3.94/0.04	3.58/0.10	0.84/0.006	0.76/0.02
Background Training Time on Battery Mean/SD (min)	85.82/33.07	79.96/36.82	25.42/5.72	24.19/8.12
Battery Consumption per Round (mAh)	9.72	3.79	2.02	0.76
Number of Training Rounds for Full Battery	308	769	1481	3846

Table 4.8 Inference Resource Consumption and Latency

Model	HAR		SA	
Phone	Google Pixel 3a	Google Pixel 3	Google Pixel 3a	Google Pixel 3
Maximum RAM Usage (MB)	158	177	108	129
Foreground Inference Time Mean/SD (millisecond)	38.48/10.07	36.59/6.43	11.90/3.71	10.11/2.88
Background Inference Time on Charger Mean/SD (millisecond)	99.73/19.76	99.60/33.69	20.65/4.45	15.59/5.89
Background Inference Time on Battery Mean/SD (millisecond)	100.11/19.69	100.11/21.45	19.58/3.93	17.42/5.69
Battery Consumption per prediction (μAh)	4.12	1.94	2.3	0.17
Millions of inferences for Full Battery	0.73	1.50	1.30	17.63

background training time on charger, which is the expected situation for FL training, is good for any practical situation. The phones experience a higher training time compared with the foreground case (completed one training round in less than 4 minutes). The background training time on battery is notably longer, since Android attempts to balance computation with battery saving.

The results show training is also feasible in terms of resource consumption. The maximum RAM usage of the app is less than 165MB, and modern phones are equipped with sufficient RAM to handle it. While we did not perform experiments for battery consumption in the foreground (as this test was used just for a lower bound on computation time), we measured battery consumption for background training on battery. The phones could easily perform hundreds of rounds of training on a fully charged battery. It is worth noting that, typically, one round of training per day is enough, as the users need enough time to collect new data.

Inference Performance The results in Table 4.8 demonstrate that FLSys can be used efficiently by third-party apps. The inference time is measured within the testing app, and thus includes the latency due to both FLSys and the FL models. We continuously perform predictions for 30 minutes and report the average values. The inference time for the three scenarios on the third-party app, foreground, background on charger, and background on battery, follows a similar trend as training. FLSys and HAR-Wild/SA have reasonable resource consumption, which make them effective in practice.

In the real world, inference may be expected in both foreground and background. One inference can be completed within tens of milliseconds for foreground and background on charger, which is good for all practical applications. The latency for background on battery is hundreds of milliseconds, which is still acceptable for many applications. However, the two phones experience high standard deviation in this scenario. This is because inference is typically a short task, and whenever it is interfered by other processes, its execution may be significantly prolonged. Nevertheless, background inference is typically used to assist an application, such as advertisement, and even latency in the orders of seconds may be acceptable for practical purposes.

FLSys and HAR-Wild also have reasonable resource consumption, which make them effective in practice. The maximum RAM usage for inference is less than 177MB. The battery consumption per prediction is low enough to execute millions of predictions with a full battery. With further optimization, such as model pruning and compression, the resource consumption can be even lower.

4.1.21 Conclusions, Lessons Learned, and Future Work

This chapter presented our experience with designing, building, and evaluating FLSys, an end-to-end federated learning system. FLSys was designed based on requirements derived from real-life applications that learn from mobile user data collected in the wild, such as human activity recognition (HAR). Compared with existing FL systems, FLSys balances model utility with resource consumption on the phones, tolerates client failures/disconnections and allows clients to join training at any time, supports multiple DL models that can be used concurrently by multiple apps, and acts as a "central hub" on the phone to manage the training, updating, and access control of FL models used by different apps. We built a complete prototype of FLSys in Android and AWS, and used this prototype to demonstrate that FLSys is effective and efficient in practice in terms of model performance, resource usage, and latency. We believe FLSys can open the path toward creating an FL ecosystem of models and apps for privacy-preserving deep learning on mobile sensing data. In terms of actual deployment of FLSys in practice, we believe it can be offered as FL as a Service (FLaaS) by cloud providers. Next, we report lessons learned and future work.

Build mechanisms to cope with non-IID data Since our data collection happened during the Covid-19 pandemic, we expected to see somewhat similar data from users who mostly stayed indoors. However, the data was non-IID strengthening the idea that data collected in the wild will almost always be non-IID. A future work in FLSys is to provide support for model and data-specific augmentation and other approaches to cope with non-IID data.

Beware the simulation pitfalls One common practice in FL simulations is to use the same instances/placeholders in memory for the different clients. Such simulations must carefully reset the instances for different clients to avoid any information leakage among clients, which can never happen in a real system. Our initial experiments showed unexpectedly different results between simulations and Android emulators with DL4J for the same settings. The first problem we discovered was that Batch Normalization (BN) is not supported in DL4J for specific data shapes. We implemented our own BN in DL4J, but the simulation results still did not match the experimental results. Finally, we realized that BN does not work well for FL (consistent with [212]), but it does work in the simulations due to shared instances among the simulated clients. Thus, the FL models used in the reported experiments do not use BN. The second problem we noticed was that the Adam optimizer worked well for simulation, but not for the Android emulator experiments. This was also caused by shared instances accessed by all clients in the simulation. This should not happen in practice given privacy leakage through the shared instances. The lesson learned was that simulation may show better results than experiments with real systems for FL. Since most of FL papers in the literature are based on simulations, which may suffer from similar problems with the ones described here. We believe FLSys offers an opportunity to test such FL models in real-life conditions.

Balance mobile resources and model accuracy In the current FL literature, there are no results to show the FL models work well on mobile devices, while consuming a limited amount of resources on these devices (e.g., battery power, memory). A lesson that we understood early on is that FLSys will need to balance resource usage on mobiles with model accuracy. Therefore, FLSys used an asynchronous design in which policies on the mobile devices are evaluated to decide when it makes sense for the device to participate in training and consume resources. Our results show that good model accuracy can be achieved even when a significant number of mobile devices do not participate in training in order to save resources. Let us also note that real systems cannot expect to run the same number of rounds that we observe in simulations. For example, it is common to see 10,000 rounds in simulations. However, in real life, mobile devices may not train more than once a day due to both resource consumption and lack of enough new data. In such a situation, running 10,000 rounds will take over 27 years. Thus, models must be optimized for a realistic number of rounds.

Design for flexibility FLSys was designed for model flexibility on the phones from the beginning (i.e., allow apps to use multiple interchangeable models). However, we did not originally design for flexiblity in the cloud. At first, we used virtual machines in the cloud and durable cloud storage for all FL operations. When we analyzed scalability and performance issues, we realized that an FaaS solution and different types of storage are necessary. Therefore, we changed the design of the FLSys in the cloud to allow for different types of cloud platforms and storage options. Thus, FLSys can easily be ported to other cloud platforms beyond AWS.

Future Work In the near future, we will add features to allow FLSys to support continuous data collection, which is what we expect to see in real-life scenarios. We will also focus on designing and implementing privacy and security components for FLSys. Finally, we plan to improve FLSys from a DevOps point of view. We will evaluate the system performance under concurrent training of multiple models, plug-n-play modules, and support a dashboard.

CHAPTER 5

OTHER WORK

5.1 Recursive Structure Similarity: A Novel Algorithm for Graph Clustering

5.1.1 Abstract

A various number of graph clustering algorithms have been proposed and applied in real-world applications such as network analysis, bio-informatics, social computing, and etc. However, existing algorithms usually focus on optimizing specified quality measures at the global network level, without carefully considering the destruction of local structures which could be informative and significant in practice. In this work, we propose a novel clustering algorithm for undirected graphs based on a new structure similarity measure which is computed in a recursive procedure. Our method can provide robust and high-quality clustering results, while preserving informative local structures in the original graph. Rigorous experiments conducted on a variety of benchmark and protein datasets show that our algorithm consistently outperforms existing algorithms.

5.1.2 Introduction

In general, graph clustering, or community detection, is to detect cluster or community structures in a given system or network by analyzing its corresponding graph using information from the graph's topology. Graph clustering is one of the most important ways of network analysis [213, 214].

A wide variety of methods have been developed to perform graph clustering, and many of them can obtain impressive results in competitive running time. However, there are still weaknesses and limitations in existing methods. For instance, in Spectral clustering algorithm [213], the relation between the original minimum cut problem over actual clustering and the relaxed version of it is still not clear, and the necessity of providing the number of clusters k limits the scope of the Spectral clustering. For Modularity optimization methods [215, 216, 217], they suffer from inconsistency and may easily get stuck in local maximum due to their randomness nature. Even very effective, Markov Cluster Algorithm (MCL) [214] has an issue of eliminating the graph's original structure. This results in producing star-like clusters, which may destroy dense and informative local structures.

In this work, we propose a novel community detection algorithm for undirected graph based on a new structure similarity measure. Structure similarity in general is a type of similarity measure that shows how two nodes are similar to each other, at the graph structure point of view. By incorporating structure similarity with Shannon's information theory, we define a novel *recursive structure similarity* as a measure of the weight, or the carrying information, for each edge in the graph. In our algorithm, the structure similarity is calculated recursively until it converged to an equilibrium state in which each edge has a consistent weight, which enables clustering through removing weak edges. Rigorous experiments conducted on well-known benchmark datasets and well-constructed protein-protein interaction datasets show that our algorithm consistently outperforms existing algorithms given a variety of quality and accuracy measures.

5.1.3 Related Work

Community detection or graph clustering has been extensively researched through many different tracks. Here we briefly revisit the early and traditional methods of addressing the problem, as well as the state-of-the-art methods.

Hierarchical clustering algorithms [218] approach this problem by revealing a hierarchical structure of the target graph, given that many graphs do have hierarchical structures. Partitional clustering approaches [219] are often used to find a given number of clusters by mapping data points into metric space and then minimizing some chosen cost functions which are based on distance measures. For instance, DBSCAN [220] utilizes density by scanning each data point's perimeter and merging points and clusters if they are within a distance threshold.

Spectral clustering has been researched extensively, but there are still problems to be solved. For example, the relationship between the original minimum cut problem and the relaxed version of the graph partition problem is still unclear. One of the most recent works [221] proposed a quite effective normalized spectral clustering algorithm.

The divisive method proposed by Girvan and Newman [222] is a significant contribution for clustering. In this method, edge betweenness, as a centrality measure, is used to pick edges that are to be removed. Later, Modularity optimization-based methods [223, 217] that use greedy techniques to give approximated results became widely adopted.

Another branch contains Random-Walk based methods such as MCL [214] and Infomap [224]. MCL iteratively performs a sequence of expansion, inflation, and normalization operations on the transfer matrix of the graph, which represents the Random Walk probability of every edge, until it converges. It was commonly used in bio-informatics because of its simplicity. Infomap [224] algorithm uses a probability flow of Random Walks to represent information flow in the network, and then reveals community structure by compressing a description of the probability flow.

5.1.4 Recursive Structure Similarity

In this subsection, we first give the definition of the graph and show how the weight, or the carrying information, of each edge is quantified by defining variance and covariance using Shanon's information theory. Then, we demonstrate how community detection can be done using the quantified information.

We consider an undirected and unweighted network $G = (V, E)$ with n nodes $\in V$ and l edges $\in E$. Given node i and node j, their direct connection status is denoted as I_{ij}, where $I_{ij} = 1$ if the two nodes are directly connected, and $I_{ij} = 0$ otherwise. We always have $I_{ij} = I_{ji}$, since edges are undirected. We adopt the convention that $I_{ii} = 1$, which means that any node i is considered connected with itself. We have the edge weight between node i and j denoted as θ_{ij}, and $\theta_{ij} = \theta_{ji}$. A weight matrix $\Theta = (\theta_{ij})_{n\times n}$ is where each edge's weight θ_{ij} is kept. Originally, all entries in Θ equals to 1.0. Our goal is to quantify the information carried in each edge, which is then reflected in weight matrix Θ. By setting a threshold τ, weakly connected edges will be removed to extract communities.

Quantifying the Information Carried in the Network G To quantify the information carried in the network G, we first quantify how much information is carried in a single node and a single edge. Let us consider only one node i and one of its neighbors k. Given the fact that, for every pair of nodes (e.g., node i and node k), we have both θ_{ik} and θ_{ki} in the weight matrix Θ. For making notation confusion free, we denote one of the nodes i as i' and replicate all its connections to k, as shown in Figure 5.1. The self-connection I_{ii} is now denoted as a direct connection $I_{ii'} = 1$ between i and i'. Also, an indirect connection between i and i' through neighbor k appears. Considering all the original neighbors of node i, in Figure 5.1, the weighted sum of all the indirect connections between i and i' is:

$$\sum_{k \neq i, k \neq i'} I_{ik}\theta_{ik}\theta_{ki'} \tag{5.1}$$

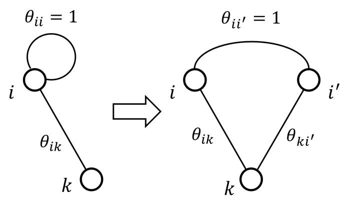


Figure 5.1 The variance of a node i, where i' is a replication of i and k denotes the neighbors of i.

For a node i, we define its variance, σ_i^2, as the sum of weighted connections that i has, including direct and indirect connections, given weight matrix Θ. The intuition of this definition is that the variance of a node i given its connections is a measure of how much structural information is reserved in its local network. The greater variance a node i has, the larger amount of the local structural information it carries. Given that $\theta_{ik} = \theta_{ki'}$, the variance of a node i can be denoted as follows:

$$\sigma_i^2 = \sigma_i^2(\Theta) = 1 + \sum_{k \neq i, k \neq i'} I_{ik}\theta_{ik}\theta_{ki'} = 1 + \sum_{k \neq i} I_{ik}(\theta_{ik})^2 \tag{5.2}$$

Similarly, the covariance of an edge ij between a pair of nodes i and j is a measure of how much information is shared among i and j. A pair of nodes with higher covariance is more likely to be in the same cluster, since they share more information and cutting them will cost more information loss. The covariance between nodes i and j can be simply defined as a product of how much information each node carries and the weighted connection between them:

$$Cov_{ij}(\Theta) = \sigma_i \sigma_j \theta_{ij} \tag{5.3}$$

By convention, the total information the network $G = (V, E)$ carries can be measured as the sum of the covariances of all the edges in E:

$$Info(G) = \sum_{ij \in E} Cov_{ij}(\Theta) = \sum_{ij \in E} \sigma_i \sigma_j \theta_{ij} \tag{5.4}$$

In (5.4), the total information in the network G is measured based on how nodes directly share information via only weighted direct connections θ_{ij}. However, nodes can also indirectly share information to each other through their common neighbors. Therefore, the total information or the covariance $Cov_{ij}(\Theta)$ between node i and node j needs to be decomposed into direct and indirect information sharing, correspondingly denoted as $d_{ij}(\Theta)$

and $d^c_{ij}(\Theta)$. Given two different nodes i and j, the direct information sharing is difficult to define at this moment, but the indirect information sharing $d^c_{ij}(\Theta)$ can be defined as follows:

$$d^c_{ij}(\Theta) = \sum_{k=1}^{n} I_{ik} I_{kj} d_{ik}(\Theta) d_{kj}(\Theta) \tag{5.5}$$

The direct information sharing $d_{ij}(\Theta)$ can be then defined:

$$d_{ij}(\Theta) = Cov_{ij}(\Theta) - d^c_{ij}(\Theta) \tag{5.6}$$

From (5.6), we have an alternative expression of the covariance $Cov_{ij}(\Theta)$ as:

$$\sigma_i \sigma_j \theta_{ij} = d_{ij}(\Theta) + \sum_{k=1}^{n} I_{ik} I_{kj} d_{ik}(\Theta) d_{kj}(\Theta) \tag{5.7}$$

5.1.5 Recursive Structure Similarity Algorithm

In (5.7), $d_{ij}(\Theta)$ can be further defined as an arbitrary kernel function of the weight matrix Θ. For clustering problems, we formulate $d_{ij}(\Theta)$ as: $d_{ij}(\Theta) = \theta_{ij}$. Equation (5.7) now becomes:

$$\sigma_i \sigma_j \theta_{ij} = \theta_{ij} + \sum_{k=1}^{n} I_{ik} I_{kj} \theta_{ik} \theta_{kj} \tag{5.8}$$

To generalize (5.8), we use a hyper-parameter λ to control the degree of the direct information contributes to the total information of the graph as follows:

$$\sigma_i \sigma_j \theta_{ij} = \lambda \theta_{ij} + \sum_{k=1}^{n} I_{ik} I_{kj} \theta_{ik} \theta_{kj}. \tag{5.9}$$

Now we are ready to define our Recursive Structure Similarity (RSS) algorithm (Alg. 5.1) to solve (5.9) and learn the weight matrix Θ. Since (5.9) depends on the weight matrix Θ, while the weight matrix Θ is also the goal that we want to learn, we begin with some initial estimation of the weights and compute them iteratively (re-weighting) until reaching convergence. Start from an initial estimation of the weight matrix, by applying the coordinate descent algorithm, we have the explicit formula for $\theta_{ij}^{(m)}$ at the m-th iteration based on the previous state of the weight matrix $\Theta^{(m-1)}$ as follows:

$$\theta_{ij}^{(m)} = \frac{\lambda \theta_{ij}^{(m-1)} + \sum_{k=1}^{n} I_{ik} I_{kj} \theta_{ik}^{(m-1)} \theta_{kj}^{(m-1)}}{\sigma_i^{(m-1)} \sigma_j^{(m-1)}} \tag{5.10}$$

Equation (5.10) is also the definition of the new *recursive structure similarity* measure.

Convergence Analysis In our algorithm, we consider two extreme cases: **(1)** When λ is extremely large, in the numerator of right-hand-side of (5.10), the first term $\lambda \theta_{ij}^{(m-1)}$ dominates the second term $\sum_{k=1}^{n} I_{ik} I_{kj} \theta_{ik}^{(m-1)} \theta_{kj}^{(m-1)}$. Therefore, we have $\Theta^{(m)} = \Theta^{(m-1)} = \ldots = \Theta^{(0)}$. This means the weight matrix $\Theta^{(0)}$ will stay unchanged and we could not learn the weights; and **(2)** When $\lambda = 0$, we can simplify (5.9) as $\Sigma(\Theta)\Theta\Sigma(\Theta) = \Theta^2$, where $\Sigma(\Theta) = \text{diag}\{\sigma_1, \cdots, \sigma_n\}$. By the property of idempotent matrix, we can show that $\Theta^{(m)}$ converges to an identity matrix. Through these two facts we can conclude that, when enough iterations of (5.10) are preformed, any λ value that is between the two extreme stages will bring Θ to a stable state where communities can be distinguished.

Algorithm 5.1 Recursive Structure Similarity

```
1: procedure Parameter Initialization
2:     $G \leftarrow Graph(n, l)$ // import the graph
3:     $\Theta \leftarrow \{\}$
4:     for all $i, j \in (I_{ij} = 1)$ do
5:         $\theta_{ij} \leftarrow 1.0$ // default edge weight=1.0
6:         $\Theta \leftarrow \Theta \cup \theta_{ij}$
7:     $\Theta^{'} \leftarrow \Theta$ // create a copy of $\Theta$
8:     for all $\theta_{ij} \in \Theta$ do
9:         $\theta_{ij} \leftarrow init(\theta_{ij}, G)$
10: procedure Parameter Estimation
11:     $p \leftarrow 0$ // loop index
12:     $\delta \leftarrow \epsilon$ // $\epsilon$ being the threshold of $\delta$ value
13:     while $p < max\_iteration$ and $\delta >= \epsilon$ do
14:         for all $\theta_{ij} \in \Theta$ do
15:             $\theta^{'}_{ij} \leftarrow RSS(\theta_{ij})$ // $RSS()$ being the Equation (5.10)
16:             $\theta_{ij} \leftarrow \theta^{'}_{ij}$
17:         $\delta \leftarrow d(\Theta, \Theta^{'})$ // $d(\Theta^m, \Theta^{m+1})$ being the Equation (5.12)
18:         $p \leftarrow p + 1$
19: procedure Community Detection
20:     $C \leftarrow \{\}$ // Empty set of clusters
21:     for all $\theta_{ij} \in \Theta$ do
22:         if $\theta_{ij} > \tau$ then
23:             if $G_i$ & $G_j \notin$ (any c $\in C$) then
24:                 $c \leftarrow \{G_i, G_j\}$ // create a new cluster
25:                 $C \leftarrow C \cup c$
26:             else if $G_i \in (c \in C)$ & $G_j \notin (c \in C)$ then
27:                 $c \leftarrow c \cup G_j$
28:             else if $G_i \notin (c \in C)$ & $G_j \in (c \in C)$ then
29:                 $c \leftarrow c \cup G_i$
    return $C$
```

Community Detection with RSS Here we present how we use RSS to perform community detection. The pseudo code is shown in Alg. 5.1. Our Recursive Structure Similarity Clustering algorithm consists of three stages. The initialization stage reads the graph from the file and initializes the weight matrix Θ. All the entries in the Θ would be initiated by using the modified cosine similarity as:

$$\frac{2 + |\Gamma_i \cap \Gamma_j|}{\sqrt{(1 + |\Gamma_i|)(1 + |\Gamma_j|)}} \tag{5.11}$$

where $|\Gamma_i|$ is the number of neighbors of node i.

In the second stage, RSS is applied iteratively to update the weight matrix Θ. In each iteration, (5.10) is applied to each entry in the current $\Theta^{(m)}$ to create $\Theta^{(m+1)}$. The hyper-parameter λ was set to 2 for our tests. This stage is terminated when a desired number of iterations ($max\_iteration$) is reached, or when the average difference between $\Theta^{(m)}$ and $\Theta^{(m+1)}$ is smaller than a pre-defined threshold ϵ:

$$\frac{1}{l} \sum_{I_{ij}=1} \left|\theta_{ij}^{(m)} - \theta_{ij}^{(m-1)}\right| < \epsilon \tag{5.12}$$

The setting of $max\_iteration$ and ϵ depends on the graph, the required level of convergence, and the desire of running time. The $max\_iteration$ was set to 5000, and we used $\epsilon = 10^{-8}$ in our tests to ensure good convergence.

In the final stage, the input is the weight matrix Θ and the list of nodes, and the output is a set of clusters. The clusters are separated from each other by removing edges that have connection strengths below certain threshold τ. Threshold τ is a hyper-parameter that controls the sparsity of the result. A larger τ will let more edges to be removed thus creates more clusters. The choice of τ is domain dependent. An example of it is $\tau = 10^{-5}$. Given a Θ, this stage can be repeated to find the optimum τ for desired clustering results.

Here we analyze our algorithm's time complexity. For a reasonably sparse graph with n nodes and l edges, $\mathbb{E}_{deg}(G)$ being the expected average degree of each node, and $\mathbb{E}_{neighbors}(i, j)$ being the expected number of common neighbors that any pair of nodes i and j has, then the time complexity will be $O\big(2lm\mathbb{E}_{deg}(G)\mathbb{E}_{neighbors}(i, j)\big)$. Also, in the second stage, which is most time-consuming stage, each m-th step of this algorithm is based on the static $(m-1)$-th state, and the order of entries to be updated does not affect the final result. This property enables a distributed parallel implementation of our algorithm for handling large dataset.

5.1.6 Experimental Settings

Datasets Six datasets were used for our experiments. The small benchmark datasets we used were: "Zachary's karate club" (Karate), a social network; Polbooks, a co-purchase network; PPI5, a protein-protein interaction network; and DBLP, a citation network. For datasets with ground-truth, we used LCDIP and BioGRID protein-protein interaction (PPI) network datasets, both of which were used by [225] for evaluation purposes. The LCDIP and BioGRID datasets were accompanied by the gold standards, MIPS and SGD [225]. Table 5.1 shows details of each dataset and ground-truth.

Baseline Approaches We chose four effective clustering algorithms as baseline approaches with which to compare with. These measures are: MCL [214], FastUnfolding [216], Infomap [224], and a newly proposed version of Spectral clustering [221].

5.1.7 Accuracy and Quality Measures

We first focused on how accurately the clustering results could represent or align with real, meaningful communities. We used the following five measures: Clustering-wise Sensitivity (CWS) [226], Clustering-wise Positive Predictive Value (PPV) [226],

Table 5.1 Datasets

Type	Name	# of nodes	# of edges	# of clusters
Benchmark Network	Karate	34	78	-
	Polbooks	126	440	-
	PPI5	184	261	-
	DBLP	1,230	3,410	-
PPI Network	LCDIP	4,980	22,076	-
	BioGRID	5,640	59,748	-
PPI Gold Standard	SGD	-	-	323
	MIPS	-	-	203

Geometric Accuracy (ACC) [226], Fraction ($FRAC$) [226, 227], and Maximum Matching Ratio (MMR) [225]. For all these measures, the higher the scores are, the better.

For measuring the quality of community detection results, we chose the following seven measures suggested by [228]: Internal Density (*In-Dens*) [229], Edges Inside (*Edges-In*) [229], Average Degree (*Avg-Deg*) [229], Cut Ratio (*CutRatio*) [230], Normalized Cut (*N-Cut*) [231], Average-ODF (*Avg-ODF*) [232], and Modularity (*Mod.*) [233]. These measures in general can show us, for a given set of nodes, how community-like its connectivity structure is [228], disregarding if the clusters reflect real-world communities or not. Note that all these measures are computed in a per-cluster manner, then an average score of all clusters was used. For *In-Dens*, *Edges-In*, *Avg-Deg*, and *Mod.*, a higher score is better; for *CutRatio*, *N-Cut*, and *Avg-ODF*, a lower score is better.

Consistency and Robustness Measures There are many possible definitions of structure similarity [234], but our algorithm should work with any structure similarity definition as the initial estimation of the *recursive structure similarity*. Thus, we tested whether using different initial structure similarity estimations will affect the consistency and robustness of our algorithm. The meaning of consistency for our algorithm is unusual since there is no randomness in the results. Instead, we want to know whether, using different initial structure similarity measures will change the outcomes of our algorithm, as the iteration process goes. Robustness refers to whether our algorithm will converge to the same equilibrium states given different initial structure similarity estimations.

For testing the consistency and robustness, we chose six structure similarity measures suggested by [234] as our algorithm's initial weight for each edge. These initial measures were modified to fit the definition of *recursive structure similarity*. We tested the two following scenarios: **(1)** Robustness: whether our algorithm would converge to the same or similar results, when different initial measures are used. **(2)** Consistency: when we change the clustering threshold (τ value), whether different initial structure similarity measures would cause the structure of clusters in the results to change. The aforementioned seven clustering quality measures were used. In the following formulas (Equation (5.13)), We show the definitions of the six different structure similarity measures of a node pair. In these formulas, i and j are any two nodes, and Γ_i means the set of neighbors of i.

$$\begin{aligned}
Default(Cosine) &= \frac{2 + |\Gamma_i \cap \Gamma_j|}{\sqrt{(1 + |\Gamma_i|)(1 + |\Gamma_j|)}} \\
Jaccard &= \frac{2 + |\Gamma_i \cap \Gamma_j|}{2 + |\Gamma_i \cup \Gamma_j|} \\
min &= \frac{2 + |\Gamma_i \cap \Gamma_j|}{min((1 + |\Gamma_i|), (1 + |\Gamma_j|))} \\
sqrt &= \sqrt{2 + |\Gamma_i \cap \Gamma_j|} \\
square &= 2 + |\Gamma_i \cap \Gamma_j|^2 \\
vertex &= \frac{2 + |\Gamma_i \cap \Gamma_j|}{(1 + |\Gamma_i|)(1 + |\Gamma_j|)}
\end{aligned} \tag{5.13}$$

5.1.8 Accuracy results

In this test, to ensure fair comparison, we used each algorithm's default parameter. For our algorithm (denoted as RSS), we used parameter settings mentioned in algorithm subsection. For MCL, we used $I = 3.0$ as the inflation value. For Infomap, we ran the program with default $Markov - time$ option value 1.0. For FastUnfolding (denoted as FUF), we ran it without q (Modularity increase threshold) parameter, which means highest accuracy [216]. Because Spectral was not suitable for graphs in this scale, we did not include Spectral in accuracy test. Since there were two gold standards for both BioGRID and LCDIP datasets, we evaluated each result with both gold standards.

From the shown results (Table 5.2) we can see that, in all dataset and ground-truth combinations, our algorithm beats all others in the measures of ACC, FRAC, MMR, and PPV. These results clearly show that in protein-protein interaction network analysis, our algorithm has a significant advantage over other methods in terms of the quality of the clusters and the number of meaningful clusters. We can also see that our method does not have the lead in CWS, but does not fall short by much. The reason for this could be that our method tend to predict more clusters with sizes smaller than those in the gold standards.

5.1.9 Structural quality results

Because these seven quality measure scores are strongly correlated to the number of clusters in a clustering result, we had to select appropriate parameters for each algorithm. Once each method gave the exact same number of clusters for each dataset, a fair comparison could be made. This constrain means that the clustering results might not be optimized (e.g., 4 clusters instead of 2 for Karate dataset, due to the fact that FastUnfolding can not produce result with 2 clusters). During the test we found that if the dataset is large, it was very unlikely to be able to align the number of clusters for all methods, and this was the reason for us to use only small benchmark datasets for quality testing. To adjust the clustering results, for our algorithm, we adjust τ value. For MCL it means to adjust the $-I$ value.

Table 5.2 Ground-Truth Based Accuracy Results

Algorithm	ACC	CWS	FRAC	MMR	PPV
Dataset: BioGRID, gold standard: SGD					
RSS	**0.619**	0.811	**0.377**	**0.202**	**0.472**
MCL	0.310	0.388	0.055	0.044	0.247
FUF	0.537	0.797	0.226	0.112	0.362
Infomap	0.555	**0.831**	0.187	0.081	0.371
Dataset: LCDIP, gold standard: MIPS					
RSS	**0.400**	0.414	**0.350**	**0.152**	**0.386**
MCL	0.269	0.199	0.238	0.110	0.363
FUF	0.383	0.460	0.270	0.117	0.319
Infomap	0.387	**0.470**	0.217	0.085	0.319
Dataset: LCDIP, gold standard: SGD					
RSS	**0.587**	0.644	**0.358**	**0.184**	**0.534**
MCL	0.409	0.335	0.271	0.153	0.500
FUF	0.518	0.621	0.294	0.148	0.432
Infomap	0.526	**0.650**	0.197	0.089	0.426
Dataset: BioGRID, gold standard: MIPS					
RSS	**0.460**	0.572	**0.318**	**0.139**	**0.370**
MCL	0.274	0.352	0.032	0.016	0.214
FUF	0.434	0.630	0.191	0.089	0.300
Infomap	0.456	**0.650**	0.169	0.068	0.320

For Infomap we adjust the $Markov\text{-}time$ option. For FastUnfolding algorithm, we adjust parameter q. For Spectral clustering we just give the k (number of clusters) value. Due to the page limitation, here we show some of the results for each dataset in Table 5.3.

In Table 5.3, for measures of *Edges-In*, *CutRatio*, *Avg-ODF*, and *Mod.*, our results are mostly dominant in all datasets. On the other hand, for *In-Dens*, *Avg-Deg*, and *N-Cut*, although our method does not outperform others, there is no other method is dominant in all datasets. These results show that our algorithm can consistently provide clustering results that are good in each aspect of structural quality. But it is also interesting to see that, for the measures from the same aspect, e.g., *In-Dens*, *Edges-In*, and *Avg-Deg*, our method only excels in *Edges-In*. One possible reason to explain this is that our method can find most densely connected clusters, but tend to assign nodes on the edges of the clusters to larger clusters thus lowering some of the normalized measures.

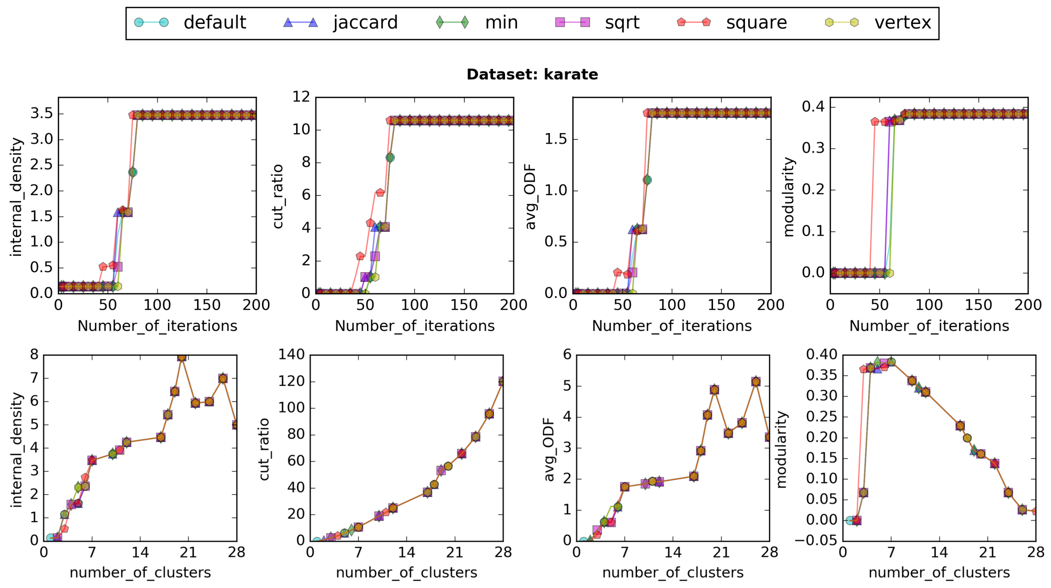


Figure 5.2 Upper: Consistency test result on Karate. Lower: Robustness test result on Karate

Table 5.3 Quality Test Results

Algorithm	In-Dens	Edges-In	Avg-Deg	CutRatio	N-Cut	Avg-ODF	Mod.
Dataset: PPI5, Number of Clusters: 40							
RSS	13.54	**187**	45.53	**39.49**	23.68	**5.15**	**0.63**
MCL	26.41	152	58.09	58.61	18.44	12.76	0.55
Spectral	25.58	152	55.72	49.96	18.53	11.52	0.52
FUF	**26.88**	160	**60.17**	57.05	**17.92**	12.09	0.58
Infomap	20.72	161	55.16	80.01	19.30	9.22	0.58
Dataset: Karate, Number of Clusters: 4							
RSS	1.58	**63**	10	**4.08**	1.75	**0.63**	0.37
MCL	1.45	49	8.66	8.31	2.00	1.84	0.19
Spectral	**1.80**	57	**12.42**	4.91	1.15	0.92	**0.42**
FUF	**1.80**	57	**12.42**	4.91	1.15	0.92	**0.42**
Infomap	1.95	59	12.25	4.72	**1.15**	0.99	0.42
Dataset: Polbooks, Number of Clusters: 18							
RSS	9.16	**314**	42.46	**56.42**	12.12	**5.67**	**0.46**
MCL	11.91	148	41.11	111.42	12.59	11.90	0.25
Spectral	12.81	187	**55.90**	79.34	**10.55**	9.74	0.34
FUF	9.15	261	44.88	86.28	11.92	6.29	0.42
Infomap	**13.38**	250	52.73	90.82	11.01	9.01	0.42
Dataset: DBLP, Number of Clusters: 252							
RSS	185.34	**2,567**	715.18	**347.53**	101.05	60.47	**0.74**
MCL	188.99	1,970	650.97	618.97	117.12	101.55	0.57
spectral	**212.61**	2,347	740.41	362.55	**86.95**	76.08	0.67
FUF	210.53	2,491	**762.60**	373.53	88.38	75.15	0.72
Infomap	158.88	2,339	676.07	905.84	107.46	**52.25**	0.68

5.1.10 Consistency and robustness results

We evaluated the consistency of our algorithm by plotting the number of iterations that the algorithm runs against clustering quality measures. In each sub-figure of upper part of Figure 5.2, X axis is the number of iterations our algorithm runs, and Y axis is the value of quality measures. Different line colors and shapes stands for different initial structure similarity measures.

As we can see in the upper part of Figure 5.2, although there are some variations at the beginning and early stage, all results align perfectly with other results at the end, which indicates that our algorithm is consistent against different initial structure similarity measures.

We evaluated the robustness by plotting the number of clusters against each structural quality measure. In each sub-figure of lower part of Figure 5.2, the X axis is the number of clusters detected by our algorithm, and the Y axis is the value of quality measures. Different line color stands for different initial structure similarity definitions. As we can see in lower part of Figure 5.2, except when the number of clusters is small, different initial structure similarity measures have very little impact on the structure quality of detected clusters, which means that as long as the initial structure similarity measure follows general definition of *recursive structure similarity*, our algorithm will give robust clustering results.

5.1.11 Conclusions

In this work, we proposed a novel clustering algorithm for detecting community structures in undirected graphs. We newly defined a *recursive structure similarity* based on Shannon's information theory, which can quantify the carried information of edges using variance and covariance. And the clustering process is done by conducting a recursive procedure that effectively shows the graph's networking process. We put our algorithm into a competition for clustering quality and accuracy with widely used clustering algorithms. The results show that our algorithm can consistently outperform other methods in the majority of quality and accuracy measures. Other test results confirm that our algorithm is also robust against different variants of initial structure similarity measures. Another advantage that our algorithm has is that there is no random factor to affect the outcome.

CHAPTER 6

CONCLUSIONS AND FUTURE DIRECTIONS

In this chapter, we conclude this dissertation in each direction it covers and show the future research directions.

6.1 Drug Abuse Detection and Analysis in Online Social Media

In this line of work, we explore the usefulness of online social media in the detecting and tracking of the drug abuse epidemic. In the first part, we explore the utilization of crowd-sourcing platform for acquiring labeled dataset, where drug abuse related tweets are identified from regular tweets filtered with drug abuse terms. Then, we design an ensembled deep learning model and employ self-learning technique to train the model on the naturally biased dataset and achieve superior performance over traditional machine learning models. Based on this model, in the second part, we run our model on a dataset of three million tweets. We perform various statistical, temporal, and spacial analysis on the identified drug abuse related tweets. The interesting patterns identified from the results provide insights of the usefulness of online social media in reflecting the trend of drug abuse epidemic. Having the detection results, in the third part, we develop a community-focused drug abuse monitoring and supporting system that provides a function rich visualization interface that can help local communities an organizations being informed about drug trends, locating drug abuse hot-spots, and reaching online users who may in need for help.

During the course of the aforementioned projects, we also identified many directions for future work. Here we list the most important ones: (1) To establish a long term, large scale drug abuse trend monitoring system that uses Lifelong Learning (L2M) models to track the ever changing and emerging trend of drug abuse; (2) To expend the source of data from Twitter to other popular social media platforms, e.g. Reddit, Instagram, or even short video-based Tiktok, and to extend the model from using only text data to images and sounds;

(3) To build a real-time drug abuse monitoring system that is capable of continuously processing and integrating data from multiple sources, training and evaluating detection models, and performing comprehensive visualization; (4) To explore the popularity and impact of the songs with drug abuse mentions in their lyrics over the social media users.

6.2 Differential Privacy in Deep Learning with Certified Robustness Bounds

In this line of work, we propose novel privacy protection mechanisms with theoretical foundations that enhance the privacy protection while keeping the models' utilities. In the first part, we proposed Adaptive Laplace Mechanism (AdLM), a DP preserving mechanism for deep learning that makes the consumption of privacy budget independent to the number of training epochs. It also improves the model utility over previous privacy mechanism by injecting noise into different parts of the features. In the second part, we established a connection among DP preservation to protect the training data, adversarial learning, and certified robustness. We developed a stochastic batch training mechanism to bypass the vanilla iterative batch-by-batch training, enabling large scale distributed DP training. We also proposed a new Monte Carlo Estimation scheme to stabilized the estimation of the robustness bounds of prediction results. In the third part, that L2M introduces unknown privacy risk and challenges in preserving DP. We proposed a new definition of Lifelong DP that protects any tuple in any training set with a consistently bounded DP loss. We also proposed the first scalable and heterogeneous mechanism, L2DP-ML to preserve Lifelong DP.

The future work of the differential privacy track falls in the following categories: (1) Test and refine the proposed mechanisms with real-world platforms, datasets, and newer attacks; (2) Explore newer and better mechanisms to achieve better model utility with lower computational cost, under regular DP and Lifelong DP.

6.3 Federated Learning on Mobile Devices

In this line of work, we present our experience with designing, building, and evaluating FLSys, an end-to-end federated learning system. FLSys was designed based on requirements derived from real-life applications that locally collects data from mobile users in the wild and trains FL models, such as human activity recognition (HAR). The main design goal of the FLSys is to preserve the resources on the mobile phones, by providing a "central hub" that manages the training and evaluation of FL models for different applications. This also provides a unified user experience by having all the FL related settings at one place, reducing the chance of having conflicted settings for different applications. FLSys is also robust to failures and disconnections, and allows clients to join training at any time. We built a complete prototype of FLSys in Android and AWS, and used this prototype to demonstrate that FLSys is effective and efficient in practice in terms of model performance, resource usage, and latency.

In the future, we plan to further develop the FLSys in the following directions: (1) To add features to allow continuous data collection and on device processing, which aligns with real-world needs; (2) To implement privacy and security features and intergral components of the FLSys; (3) To improve FLSys from a DevOps point of view, including continues model evaluation, OTA re-configuration and plug-n-play modules.

REFERENCES

[1] Office of Health Policy, “HHS acting secretary declares public health emergency to address national opioid crisis,” Accessed: June 17, 2019. [Online]. Available: https://www.hhs.gov/about/news/2017/10/26/hhs-acting-secretary-declares-public-health-emergency-address-national-opioid-crisis.html

[2] Substance Abuse and Mental Health Services Administration, U.S. Department of Health and Human Services, “Key substance use and mental health indicators in the united states,” Accessed: June 17, 2019. [Online]. Available: https://datafiles.samhsa.gov/

[3] National Institute on Drug Abuse, “Overdose death rate,” Accessed: June 17, 2019. [Online]. Available: https://www.drugabuse.gov/related-topics/trends-statistics/overdose-death-rates

[4] Gun Violence Archive, “Gun violence archive,” Accessed: June 17, 2019. [Online]. Available: http://www.gunviolencearchive.org/past-tolls

[5] J. K. O’ Donnell, J. Halpin, C. L. Mattson, B. A. Goldberger, and R. M. Gladden, “Deaths involving fentanyl, fentanyl analogs, and u-47700—10 states, july–december 2016,” *Morbidity and Mortality Weekly Report*, vol. 66, no. 43, pp. 1197–1202, 2017. doi: 10.15585/mmwr.mm6643e1

[6] B. Hansen, K. Miller, and C. Weber, “Early evidence on recreational marijuana legalization and traffic fatalities,” *Economic Inquiry*, vol. 58, no. 2, pp. 547–568, 2020. doi: 10.1111/ecin.12751

[7] E. J. D’ Amico, A. Rodriguez, J. S. Tucker, E. R. Pedersen, and R. A. Shih, “Planting the seed for marijuana use: changes in exposure to medical marijuana advertising and subsequent adolescent marijuana use, cognitions, and consequences over seven years,” *Drug and Alcohol Dependence*, vol. 188, pp. 385–391, 2018. doi: 10.1016/j.drugalcdep.2018.03.031

[8] S. Aslam, “Twitter by the numbers,” Accessed: June 17, 2019. [Online]. Available: https://www.omnicoreagency.com/twitter-statistics

[9] A. Signorini, A. M. Segre, and P. M. Polgreen, “The use of Twitter to track levels of disease activity and public concern in the us during the influenza a h1n1 pandemic,” *PLOS ONE*, vol. 6, no. 5, pp. 1–10, 2011. doi: 10.1371/journal.pone.0019467

[10] Y. Aphinyanaphongs, A. Lulejian, D. P. Brown, R. Bonneau, and P. Krebs, “Text classification for automatic detection of e-cigarette use and use for smoking cessation from Twitter: a feasibility pilot,” in *Proceedings of the Pacific Symposium on Biocomputing*, no. 21, 2016, pp. 480–491.

[11] J. C. Bosley, N. W. Zhao, S. Hill, F. S. Shofer, D. A. Asch, L. B. Becker, and R. M. Merchant, “Decoding Twitter: surveillance and trends for cardiac arrest and resuscitation communication,” *Resuscitation*, vol. 84, no. 2, pp. 206–212, 2013. doi: 10.1016/j.resuscitation.2012.10.017

[12] M. Chary, N. Genes, A. McKenzie, and A. F. Manini, “Leveraging social networks for toxicovigilance,” *Journal of Medical Toxicology*, vol. 9, no. 2, pp. 184–191, 2013. doi: 10.1007/s13181-013-0299-6

[13] N. Hossain, T. Hu, R. Feizi, A. M. White, J. Luo, and H. Kautz, “Precise localization of homes and activities: detecting drinking-while-tweeting patterns in communities,” in *Proceedings of the 10th International AAAI Conference on Web and Social Media*, 2016, pp. 587–590.

[14] M. Myslín, S.-H. Zhu, W. Chapman, and M. Conway, “Using Twitter to examine smoking behavior and perceptions of emerging tobacco products,” *Journal of Medical Internet Research*, vol. 15, no. 8, p. e174, 2013. doi: 10.2196/jmir.2534

[15] A. Sarker, K. O’ connor, R. Ginn, M. Scotch, K. Smith, D. Malone, and G. Gonzalez, “Social media mining for toxicovigilance: automatic monitoring of prescription medication abuse from Twitter,” *Drug Safety*, vol. 39, no. 3, pp. 231–240, 2016. doi: 10.1007/s40264-015-0379-4

[16] C. L. Hanson, B. Cannon, S. Burton, and C. Giraud-Carrier, “An exploration of social circles and prescription drug abuse through Twitter,” *Journal of Medical Internet Research*, vol. 15, no. 9, p. e189, 2013. doi: 10.2196/jmir.2741

[17] C. L. Hanson, S. H. Burton, C. Giraud-Carrier, J. H. West, M. D. Barnes, and B. Hansen, “Tweaking and tweeting: exploring Twitter for nonmedical use of a psychostimulant drug (Adderall) among college students,” *Journal of Medical Internet Research*, vol. 15, no. 4, p. e62, 2013. doi: 10.2196/jmir.2503

[18] E. C. McNaughton, R. A. Black, M. G. Zulueta, S. H. Budman, and S. F. Butler, “Measuring online endorsement of prescription opioids abuse: an integrative methodology,” *Pharmacoepidemiology and Drug Safety*, vol. 21, no. 10, pp. 1081–1092, 2012. doi: 10.1002/pds.3307

[19] L. Shutler, L. S. Nelson, I. Portelli, C. Blachford, and J. Perrone, “Drug use in the Twittersphere: a qualitative contextual analysis of tweets about prescription drugs,” *Journal of Addictive Aiseases*, vol. 34, no. 4, pp. 303–310, 2015. doi: 10.1080/10550887.2015.1074505

[20] National Institute on Drug Abuse, University of Michigan, “Monitoring the future,” Accessed: June 17, 2019. [Online]. Available: http://www.monitoringthefuture.org

[21] D. A. Kessler, S. Natanblut, D. Kennedy, E. Lazar, P. Rheinstein, C. Anello, D. Barash, I. Bernstein, R. Bolger, K. Cook, M. P. Couig, J. Donlon, J. Johnson, C. Lorraine, T. McGinnis, J. Nazario, S. Nightingale, C. Peck, M. Pendergast, S. Rastogi, C. Reynolds, R. Schapiro, L. Tollefson, and A. Wion, "Introducing MEDWatch: a new approach to reporting medication and device adverse effects and product problems," *The Journal of the American Medical Association*, vol. 269, no. 21, pp. 2765–2768, 06 1993. doi: 10.1001/jama.1993.03500210065033

[22] American Association of Poison Control Centers, "National poisoning data system," Accessed: June 17, 2019. [Online]. Available: www.aapcc.org/data-system

[23] D. Brookoff, E. A. Campbell, and L. M. Shaw, "The underreporting of cocaine-related trauma: drug abuse warning network reports vs hospital toxicology tests," *American Journal of Public Health*, vol. 83, no. 3, pp. 369–371, 1993. doi: 10.2105/ajph.83.3.369

[24] D. S. Hasin, A. L. Sarvet, M. Cerdá, K. M. Keyes, M. Stohl, S. Galea, and M. M. Wall, "US adult illicit cannabis use, cannabis use disorder, and medical marijuana laws: 1991-1992 to 2012-2013," *JAMA Psychiatry*, vol. 74, no. 6, pp. 579–588, 2017. doi: 10.1001/jamapsychiatry.2017.0724

[25] P. Seth, R. A. Rudd, R. K. Noonan, and T. M. Haegerich, "Quantifying the epidemic of prescription opioid overdose deaths," *American Journal of Public Health*, vol. 108, no. 4, pp. 500–502, 2018. doi: 10.2105/AJPH.2017.304265

[26] M. Chary, N. Genes, C. Giraud-Carrier, C. Hanson, L. S. Nelson, and A. F. Manini, "Epidemiology from tweets: estimating misuse of prescription opioids in the USA from social media," *Journal of Medical Toxicology*, vol. 13, no. 4, pp. 278–286, 2017. doi: 10.1007/s13181-017-0625-5

[27] H.-W. Meng, S. Kath, D. Li, and Q. C. Nguyen, "National substance use patterns on Twitter," *PLOS ONE*, vol. 12, no. 11, pp. 1–15, 11 2017. doi: 10.1371/journal.pone.0187691

[28] T. Ding, W. K. Bickel, and S. Pan, "Social media-based substance use prediction," *Computing Research Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1705.05633

[29] S. S. Simpson, N. Adams, C. M. Brugman, and T. J. Conners, "Detecting novel and emerging drug terms using natural language processing: a social media corpus study," *JMIR Public Health and Surveillance*, vol. 4, no. 1, p. e2, 2018. doi: 10.2196/publichealth.7726

[30] T. Katsuki, T. K. Mackey, and R. Cuomo, "Establishing a link between prescription drug abuse and illicit online pharmacies: analysis of Twitter data," *Journal of Medical Internet Research*, vol. 17, no. 12, p. e280, 2015. doi: 10.2196/jmir.5144

[31] P. M. Coloma, B. Becker, M. C. J. M. Sturkenboom, E. M. van Mulligen, and J. A. Kors, “Evaluating social media networks in medicines safety surveillance: two case studies,” *Drug Safety*, vol. 38, no. 10, pp. 921–930, 2015. doi: 10.1007/s40264-015-0333-5

[32] H. Hu, P. Moturu, K. Dharan, J. Geller, S. Di Lorio, N. Phan, H. Vo, and S. Chun, “Deep learning model for classifying drug abuse risk behavior in tweets,” in *Proceedings of the IEEE International Conference on Healthcare Informatics*, 2018. doi: 10.1109/ICHI.2018.00066 pp. 386–387.

[33] C. Kong, J. Liu, H. Li, Y. Liu, H. Zhu, and T. Liu, “Drug abuse detection via broad learning,” in *Proceedings of the International Conference on Web Information Systems and Applications*, 2019, pp. 499–505.

[34] D. Weissenbacher, A. Sarker, A. Klein, K. O’ Connor, A. Magge, and G. Gonzalez-Hernandez, “Deep neural networks ensemble for detecting medication mentions in tweets,” *Journal of the American Medical Informatics Association*, vol. 26, no. 12, pp. 1618–1626, 09 2019. doi: 10.1093/jamia/ocz156

[35] D. Mahata, J. Friedrichs, R. R. Shah, and J. Jiang, “Detecting personal intake of medicine from Twitter,” *IEEE Intelligent Systems*, vol. 33, no. 4, pp. 87–95, 2018. doi: 10.1109/MIS.2018.043741326

[36] Y. Zhang, Y. Fan, Y. Ye, X. Li, and E. L. Winstanley, “Utilizing social media to combat opioid addiction epidemic: automatic detection of opioid users from Twitter,” in *Workshops at the 32nd AAAI Conference on Artificial Intelligence*, 2018.

[37] J. Li, Q. Xu, N. Shah, and T. K. Mackey, “A machine learning approach for the detection and characterization of illicit drug dealers on instagram: model evaluation study,” *Journal of Medical Internet Research*, vol. 21, no. 6, p. e13803, 2019. doi: 10.2196/13803

[38] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, “Self-taught learning: transfer learning from unlabeled data,” in *Proceedings of the 24th International Conference on Machine Learning*, 2007. doi: 10.1145/1273496.1273592 pp. 759–766.

[39] Y. Bengio, “Learning deep architectures for ai,” *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009. doi: 10.1561/2200000006

[40] J. Weston, F. Ratle, and R. Collobert, “Deep learning via semi-supervised embedding,” in *Proceedings of the 25th International Conference on Machine Learning*, 2008. doi: 10.1145/1390156.1390303 pp. 1168–1175.

[41] A. Bettge, R. Roscher, and S. Wenzel, “Deep self-taught learning for remote sensing image classification,” *Computing Research Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1710.07096

[42] X. Dong, D. Meng, F. Ma, and Y. Yang, “A dual-network progressive approach to weakly supervised object detection,” in *Proceedings of the 25th ACM International Conference on Multimedia*, 2017. doi: 10.1145/3123266.3123455 pp. 279–287.

[43] J. Gan, L. Li, Y. Zhai, and Y. Liu, “Deep self-taught learning for facial beauty prediction,” *Neurocomputing*, vol. 144, pp. 295–303, 11 2014. doi: 10.1016/j.neucom.2014.05.028

[44] Y. Yuan, X. Liang, X. Wang, D.-Y. Yeung, and A. Gupta, “Temporal dynamic graph lstm for action-driven video object detection,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1801–1810.

[45] L. Orsolini, D. Papanti, J. Corkery, and F. Schifano, “An insight into the deep web; why it matters for addiction psychiatry?” *Human Psychopharmacology: clinical and Experimental*, vol. 32, no. 3, p. e2573, 2017. doi: 10.1002/hup.2573

[46] S. Schmidt, “‘it is taking people out’: more than 70 people overdose on k2 in a single day in new haven,” 2018, accessed: June 17, 2020. [Online]. Available: https://www.washingtonpost.com/news/morning-mix/wp/2018/08/16/it-is-taking-people-out-more-than-70-people-overdose-on-k2-in-a-single-day-in-new-haven

[47] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. doi: 10.1109/5.726791

[48] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. doi: 10.1162/neco.1997.9.8.1735

[49] H. Hu, N. Phan, J. Geller, H. Vo, B. Manasi, X. Huang, S. Di Lorio, T. Dinh, and S. A. Chun, “Deep self-taught learning for detecting drug abuse risk behavior in tweets,” in *Proceedings of the 7th International Conference on Computational Social Networks*, 2018. doi: 10.1007%2F978-3-030-04648-4_28 pp. 330–342.

[50] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, “Towards fully autonomous driving: systems and algorithms,” in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2011. doi: 10.1109/IVS.2011.5940562 pp. 163–168.

[51] D. Lowd and C. Meek, “Good word attacks on statistical spam filters,” in *Proceedings of the Second Conference on Email and Anti-Spam*, 2005. [Online]. Available: http://www.ceas.cc/papers-2005/125.pdf

[52] Y. Cheng, F. Wang, P. Zhang, and J. Hu, “Risk prediction with electronic health records: a deep learning approach,” in *Proceedings of the SIAM International Conference on Data Mining*, 2016. doi: 10.1137/1.9781611974348.49 pp. 432–440.

[53] E. Choi, A. Schuetz, W. F. Stewart, and J. Sun, “Using recurrent neural network models for early detection of heart failure onset,” *Journal of the American Medical Informatics Association*, vol. 24, no. 2, pp. 361–370, 2017. doi: 10.1093/jamia/ocw112

[54] A. Chaudhry, M. Ranzato, M. Rohrbach, and M. Elhoseiny, "Efficient lifelong learning with A-GEM," in *Proceedings of the 7th International Conference on Learning Representations*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=Hkf2_sC5FX

[55] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Proceedings of the Theory of Cryptography Conference*, 2006, pp. 265–284.

[56] K. Chaudhuri and C. Monteleoni, "Privacy-preserving logistic regression," in *Proceedings of the Advances in Neural Information Processing Systems*, vol. 21. Curran Associates, Inc., 2009, pp. 289–296. [Online]. Available: https://proceedings.neurips.cc/paper/2008/file/8065d07da4a77621450aa84fee5656d9-Paper.pdf

[57] F. McSherry and I. Mironov, "Differentially private recommender systems: building privacy into the netflix prize contenders," in *Proceedings of the 15th ACM Special Interest Group on Knowledge Discovery and Data Mining International Conference on Knowledge Discovery and Data Mining*. ACM, 2009. doi: 10.1145/1557019.1557090 pp. 627–636.

[58] F. McSherry and K. Talwar, "Mechanism design via differential privacy," in *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, 2007. doi: 10.1109/FOCS.2007.66 pp. 94–103.

[59] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 53rd Annual Allerton Conference on Communication, Control, and Computing*, 2015. doi: 10.1109/ALLERTON.2015.7447103 pp. 909–910.

[60] N. Phan, Y. Wang, X. Wu, and D. Dou, "Differential privacy preservation for deep auto-encoders: an application of human behavior prediction," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. AAAI Press, 2016, pp. 1309–1316. [Online]. Available: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12174

[61] C. Dwork and J. Lei, "Differential privacy and robust statistics," in *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*. ACM, 2009. doi: 10.1145/1536414.1536466 pp. 371–380.

[62] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the ACM Special Interest Group on Security, Audit and Control Conference on Computer and Communications Security*. ACM, 2016. doi: 10.1145/2976749.2978318 pp. 308–318.

[63] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48, 2016, pp. 201–210. [Online]. Available: https://proceedings.mlr.press/v48/gilad-bachrach16.html

[64] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM Special Interest Group on Security, Audit and Control Conference on Computer and Communications Security*. ACM, 2015. doi: 10.1145/2810103.2813677 pp. 1322–1333.

[65] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proceedings of the IEEE Symposium on Security and Privacy*, vol. 1. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.49 pp. 39–57.

[66] N. Kardan and K. O. Stanley, "Mitigating fooling with competitive overcomplete output layer neural networks," in *Proceedings of the International Joint Conference on Neural Networks*, 2017. doi: 10.1109/IJCNN.2017.7965897 pp. 518–525.

[67] A. Matyasko and L.-P. Chau, "Margin maximization for robust classification using deep learning," in *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2017. doi: 10.1109/IJCNN.2017.7965869 pp. 300–307.

[68] M. Cisse, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, "Parseval networks: improving robustness to adversarial examples," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, 2017, pp. 854–863.

[69] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, 2018, pp. 5283–5292. [Online]. Available: http://proceedings.mlr.press/v80/wong18a.html

[70] H. Salman, J. Li, I. P. Razenshteyn, P. Zhang, H. Zhang, S. Bubeck, and G. Yang, "Provably robust deep learning via adversarially trained smoothed classifiers," in *Proceedings of the 33rd Conference on Neural Information Processing Systems*, 2019, pp. 11 289–11 300. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/3a24b25a7b092a252166a1641ae953e7-Abstract.html

[71] N. Phan, X. Wu, and D. Dou, "Preserving differential privacy in convolutional deep belief networks," *Machine Learning*, vol. 106, no. 9-10, pp. 1681–1704, 2017. doi: 10.1007/s10994-017-5656-2

[72] N. Phan, X. Wu, H. Hu, and D. Dou, "Adaptive laplace mechanism: differential privacy preservation in deep learning," in *Proceedings of the IEEE International Conference on Data Mining*. IEEE Computer Society, 2017. doi: 10.1109/ICDM.2017.48 pp. 385–394.

[73] L. Yu, L. Liu, C. Pu, M. E. Gursoy, and S. Truex, "Differentially private model publishing for deep learning," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 2019. doi: 10.1109/SP.2019.00019 pp. 332–349.

[74] J. Lee and D. Kifer, "Concentrated differentially private gradient descent with adaptive per-iteration privacy budget," in *Proceedings of the 24th ACM Special Interest Group on Knowledge Discovery and Data Mining International Conference on Knowledge Discovery and Data Mining*. ACM, 2018. doi: 10.1145/3219819.3220076 pp. 1656–1665.

[75] L. Song, R. Shokri, and P. Mittal, "Privacy risks of securing machine learning models against adversarial examples," in *Proceedings of the ACM Special Interest Group on Security, Audit and Control Conference on Computer and Communications Security*. ACM, 2019. doi: 10.1145/3319535.3354211 pp. 241–257.

[76] N. Phan, M. N. Vu, Y. Liu, R. Jin, D. Dou, X. Wu, and M. T. Thai, "Heterogeneous gaussian mechanism: preserving differential privacy in deep learning with provable robustness," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. ijcai.org, 2019. doi: 10.24963/ijcai.2019/660 pp. 4753–4759.

[77] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," in *Proceedings of the 6th International Conference on Learning Representations*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=Bys4ob-Rb

[78] B. Wu, S. Zhao, G. Sun, X. Zhang, Z. Su, C. Zeng, and Z. Liu, "P3SGD: patient privacy preserving SGD for regularizing deep cnns in pathological image classification," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00220 pp. 2099–2108.

[79] Z. Xu, S. Shi, A. X. Liu, J. Zhao, and L. Chen, "An adaptive and fast convergent approach to differentially private deep learning," in *Proceedings of the 39th IEEE Conference on Computer Communications*. IEEE, 2020. doi: 10.1109/INFOCOM41043.2020.9155359 pp. 1867–1876.

[80] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: training imagenet in 1 hour," *Computing Research Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1706.02677

[81] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.41 pp. 3–18.

[82] Y. Wang, C. Si, and X. Wu, "Regression model fitting under differential privacy and model inversion attack," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*. AAAI Press, 2015, pp. 1003–1009. [Online]. Available: http://ijcai.org/Abstract/15/146

[83] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *Computing Research Repository*, 2016. [Online]. Available: http://arxiv.org/abs/1611.03814

[84] R. Miotto, L. Li, B. A. Kidd, and J. T. Dudley, "Deep patient: an unsupervised representation to predict the future of patients from the electronic health records," *Scientific Reports*, vol. 6, no. 1, pp. 1–10, 2016. doi: 10.1038/srep26094

[85] M. Roumia and S. Steinhubl, "Improving cardiovascular outcomes using electronic health records," *Current Cardiology Reports*, vol. 16, no. 2, p. 451, 2014. doi: 10.1007/s11886-013-0451-6

[86] J. Wu, J. Roy, and W. F. Stewart, "Prediction modeling using ehr data: challenges, strategies, and a comparison of machine learning approaches," *Medical care*, pp. S106–S113, 2010. doi: 10.1097/MLR.0b013e3181de9e17

[87] S. M. Plis, D. R. Hjelm, R. Salakhutdinov, E. A. Allen, H. J. Bockholt, J. D. Long, H. J. Johnson, J. S. Paulsen, J. A. Turner, and V. D. Calhoun, "Deep learning for neuroimaging: a validation study," in *Proceedings of the 2nd International Conference on Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1312.5847

[88] M. Helmstaedter, K. L. Briggman, S. C. Turaga, V. Jain, H. S. Seung, and W. Denk, "Connectomic reconstruction of the inner plexiform layer in the mouse retina," *Nature*, vol. 500, no. 7461, pp. 168–174, 2013. doi: 10.1038/nature12346

[89] M. Riemer, I. Cases, R. Ajemian, M. Liu, I. Rish, Y. Tu, and G. Tesauro, "Learning to learn without forgetting by maximizing transfer and minimizing interference," in *Proceedings of the 7th International Conference on Learning Representations*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=B1gTShAct7

[90] D. Abati, J. Tomczak, T. Blankevoort, S. Calderara, R. Cucchiara, and B. E. Bejnordi, "Conditional channel gated networks for task-aware continual learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2020. doi: 10.1109/CVPR42600.2020.00399 pp. 3930–3939.

[91] X. Tao, X. Hong, X. Chang, S. Dong, X. Wei, and Y. Gong, "Few-shot class-incremental learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.01220 pp. 12 180–12 189.

[92] J. Rajasegaran, S. H. Khan, M. Hayat, F. S. Khan, and M. Shah, "itaml: an incremental task-agnostic meta-learning approach," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, 2020. doi: 10.1109/CVPR42600.2020.01360 pp. 13 585–13 594.

[93] H. Shin, J. K. Lee, J. Kim, and J. Kim, "Continual learning with deep generative replay," in *Proceedings of the 29th Conference on Neural Information Processing Systems*, 2017, pp. 2990–2999. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/0efbe98067c6c73dba1250d2beaa81f9-Abstract.html

[94] C. Wu, L. Herranz, X. Liu, J. van de Weijer, and B. Raducanu, "Memory replay gans: learning to generate new categories without forgetting," in *Proceedings of the 30th Conference on Neural Information Processing Systems*, 2018, pp. 5966–5976. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/a57e8915461b83adefb011530b711704-Abstract.html

[95] O. Ostapenko, M. Puscas, T. Klein, P. Jahnichen, and M. Nabi, "Learning to remember: A synaptic plasticity driven framework for continual learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.01158 pp. 11 321–11 329.

[96] N. Phan, M. T. Thai, H. Hu, R. Jin, T. Sun, and D. Dou, "Scalable differential privacy with certified robustness in adversarial learning," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, 2020, pp. 7683–7694. [Online]. Available: http://proceedings.mlr.press/v119/phan20a.html

[97] B. Wang and N. Hegde, "Privacy-preserving q-learning with functional noise in continuous spaces," in *Proceedings of the 31st Conference on Neural Information Processing Systems*, 2019, pp. 11 323–11 333. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/6646b06b90bd13dabc11ddba01270d23-Abstract.html

[98] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PLOS ONE*, vol. 10, no. 7, pp. 1–46, 07 2015. doi: 10.1371/journal.pone.0130140

[99] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[100] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 08 2014. doi: 10.1561/0400000042

[101] M. Lécuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2019. doi: 10.1109/SP.2019.00044 pp. 656–672.

[102] A. Tavanaei, "Embedded encoder-decoder in convolutional networks towards explainable AI," *Computing Research Repository*, 2020. [Online]. Available: https://arxiv.org/abs/2007.06712

[103] S. Farquhar and Y. Gal, “Differentially private continual learning,” *Computing Research Repository*, 2019. [Online]. Available: http://arxiv.org/abs/1902.06497

[104] N. Phan, M. T. Thai, D. M. Shila, and R. Jin, “Differentially private lifelong learning,” in *Proceedings of the Privacy in Machine Learning Workshop of the 31st Conference on Neural Information Processing Systems*, 2019. [Online]. Available: https://priml-workshop.github.io/priml2019/papers/PriML2019_paper_34.pdf

[105] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, “Overcoming catastrophic forgetting in neural networks,” *Computing Research Repository*, 2016. [Online]. Available: http://arxiv.org/abs/1612.00796

[106] K. A. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, B. McMahan, T. V. Overveldt, D. Petrou, D. Ramage, and J. Roselander, “Towards federated learning at scale: system design,” in *Proceedings of Machine Learning and Systems*, 2019.

[107] A. K. Sahu, T. Li, M. Sanjabi, M. Zaheer, A. Talwalkar, and V. Smith, “On the convergence of federated optimization in heterogeneous networks,” *Computing Research Repository*, 2018. [Online]. Available: http://arxiv.org/abs/1812.06127

[108] D. Sarkar, A. Narang, and S. Rai, “Fed-focal loss for imbalanced data classification in federated learning,” *Computing Research Repository*, 2020. [Online]. Available: https://arxiv.org/abs/2011.06283

[109] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-iid data,” *Computing Research Repository*, 2018. [Online]. Available: http://arxiv.org/abs/1806.00582

[110] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, “Self-balancing federated learning with global imbalanced data in mobile systems,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 59–71, 2021. doi: 10.1109/TPDS.2020.3009406

[111] D. C. Verma, G. White, S. Julier, S. Pasteris, S. Chakraborty, and G. Cirincione, “Approaches to address the data skew problem in federated learning,” in *Proceedings of the Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006. SPIE, 2019. doi: 10.1117/12.2519621 pp. 542–557.

[112] Y. Liu, A. Huang, Y. Luo, H. Huang, Y. Liu, Y. Chen, L. Feng, T. Chen, H. Yu, and Q. Yang, “Fedvision: an online visual object detection platform powered by federated learning,” in *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, vol. 34, no. 08, 2020. doi: 10.1609/aaai.v34i08.7021 pp. 13 172–13 179.

[113] C. He, S. Li, J. So, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, and S. Avestimehr, "Fedml: A research library and benchmark for federated machine learning," *Computing Research Repository*, 2020. [Online]. Available: https://arxiv.org/abs/2007.13518

[114] FedAI, "FATE: an industrial grade federated learning framework," 2021, accessed: June 18, 2021. [Online]. Available: https://fate.fedai.org/

[115] V. Mugunthan, A. Peraire-Bueno, and L. Kagal, "Privacyfl: A simulator for privacy-preserving and secure federated learning," in *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*. ACM, 2020. doi: 10.1145/3340531.3412771 pp. 3085–3092.

[116] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: improving google keyboard query suggestions," *Computing Research Repository*, 2018.

[117] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020. doi: 10.1109/MSP.2020.2975749

[118] G. D. Jacobsen and K. H. Jacobsen, "Statewide covid-19 stay-at-home orders and population mobility in the united states," *World Medical & Health Policy*, vol. 12, no. 4, pp. 347–356, 2020. doi: 10.1002/wmh3.350

[119] K. R. Choi, M. V. Heilemann, A. Fauer, and M. Mead, "A second pandemic: mental health spillover from the novel coronavirus (COVID-19)," *Journal of the American Psychiatric Nurses Association*, vol. 26, no. 4, pp. 340–343, 2020. doi: 10.1177/1078390320919803

[120] A. Ignatov, "Real-time human activity recognition from accelerometer data using convolutional neural networks," *Applied Soft Computing*, vol. 62, pp. 915–922, 2018. doi: 10.1016/j.asoc.2017.09.027

[121] A. Murad and J.-Y. Pyun, "Deep recurrent neural networks for human activity recognition," *Sensors*, vol. 17, no. 11, p. 2556, 2017. doi: 10.3390/s17112556

[122] F. Hernández, L. F. Suárez, J. Villamizar, and M. Altuve, "Human activity recognition on smartphones using a bidirectional lstm network," in *Proceedings of the XXII Symposium on Image, Signal Processing and Artificial Vision*, 2019. doi: 10.1109/STSIVA.2019.8730249 pp. 1–5.

[123] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *ACM Special Interest Group on Knowledge Discovery and Data Mining Explorations Newsletter*, vol. 12, no. 2, pp. 74–82, 2011. doi: 10.1145/1964897.1964918

[124] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, “A public domain dataset for human activity recognition using smartphones,” in *Proceedings of the 21st European Symposium on Artificial Neural Networks*, 2013.

[125] R. Chavarriaga, H. Sagha, A. Calatroni, S. T. Digumarti, G. Tröster, J. d. R. Millán, and D. Roggen, “The opportunity challenge: a benchmark database for on-body sensor-based activity recognition,” *Pattern Recognition Letters*, vol. 34, no. 15, pp. 2033–2042, 2013. doi: 10.1016/j.patrec.2012.12.014

[126] Y. Chen, K. Zhong, J. Zhang, Q. Sun, and X. Zhao, “Lstm networks for mobile human activity recognition,” in *Proceedings of the International Conference on Artificial Intelligence: Technologies and Applications*, 2016. doi: 10.2991/icaita-16.2016.13 pp. 50–53.

[127] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. A. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D’Oliveira, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, “Advances and open problems in federated learning,” *Computing Research Repository*, 2019. [Online]. Available: http://arxiv.org/abs/1912.04977

[128] Q. Yang, Y. Liu, T. Chen, and Y. Tong, “Federated machine learning: concept and applications,” *ACM Transactions on Intelligent Systems and Technology*, vol. 10, no. 2, pp. 1–19, 01 2019. doi: 10.1145/3298981

[129] D. Verma, G. White, and G. de Mel, “Federated ai for the enterprise: a web services based implementation,” in *Proceedings of the IEEE International Conference on Web Services*, 2019. doi: 10.1109/ICWS.2019.00016 pp. 20–27.

[130] Z. Feng, H. Xiong, C. Song, S. Yang, B. Zhao, L. Wang, Z. Chen, S. Yang, L. Liu, and J. Huan, “Securegbm: secure multi-party gradient boosting,” in *Proceedings of the IEEE International Conference on Big Data*, 2019. doi: 10.1109/BigData47090.2019.9006000 pp. 1312–1321.

[131] L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai, “Privacy-preserving deep learning via additively homomorphic encryption,” *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 5, pp. 1333–1345, 2018. doi: 10.1109/TIFS.2017.2787987

[132] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, “Practical secure aggregation for privacy-preserving machine learning,” in *Proceedings of the ACM Special Interest Group on Security, Audit and Control Conference on Computer and Communications Security*. ACM, 2017. doi: 10.1145/3133956.3133982 pp. 1175–1191.

[133] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, “Federated optimization: distributed machine learning for on-device intelligence,” *Computing Research Repository*, 2016. [Online]. Available: http://arxiv.org/abs/1610.02527

[134] Office of the Law Recision Counsel, “Drug Abuse Prevention and Control. Definitions, 21 U.S.C Sect. 802,” Accessed: June 17, 2019. [Online]. Available: https://www.deadiversion.usdoj.gov/21cfr/21usc/802.htm

[135] M. Buhrmester, T. Kwang, and S. D. Gosling, “Amazon’s mechanical turk: a new source of inexpensive, yet high-quality, data?” *Perspectives on Psychological Science*, vol. 6, no. 1, pp. 3–5, 2011. doi: 10.1177/1745691610393980

[136] D. Mahata, J. Friedrichs, Hitkul, and R. R. Shah, “#phramacovigilance - exploring deep learning techniques for identifying mentions of medication intake from Twitter,” *Computing Research Repository*, 2018. [Online]. Available: http://arxiv.org/abs/1805.06375

[137] X. Zhang, J. Zhao, and Y. LeCun, “Character-level convolutional networks for text classification,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, 2015, pp. 649–657.

[138] NoSlang.com, “Drug slang translator,” Accessed: June 17, 2019. [Online]. Available: https://www.noslang.com/drugs/dictionary.php

[139] G. A. Miller, “WordNet: a lexical database for English,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995. doi: 10.1145/219717.219748

[140] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *Computing Research Repository*, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[141] J. Pennington, R. Socher, and C. D. Manning, “Glove: global vectors for word representation,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[142] F. Godin, “Twitter Word2vec model,” 2015, accessed: June 17, 2019. [Online]. Available: https://www.fredericgodin.com/software/

[143] A. Sarker and G. Gonzalez, “A corpus for mining drug-related knowledge from Twitter chatter: language models and their utilities,” *Data in Brief*, vol. 10, pp. 122–131, 2017. doi: 10.1016/j.dib.2016.11.056

[144] L. A. Jeni, J. F. Cohn, and F. De La Torre, “Facing imbalanced data–recommendations for the use of performance metrics,” in *Proceedings of the International Humaine Association Conference on Affective Computing and Intelligent Interaction*, 2013. doi: 10.1109/ACII.2013.47 pp. 245–251.

[145] K. A. Hallgren, "Computing inter-rater reliability for observational data: an overview and tutorial," *Tutorials in Quantitative Methods for Psychology*, vol. 8, no. 1, pp. 23–34, 2012. doi: 10.20982/tqmp.08.1.p023

[146] U.S. National Institute on Drug Abuse, "Commonly abused drugs," Accessed: June 17, 2019. [Online]. Available: https://www.drugabuse.gov/drug-topics/commonly-used-drugs-charts

[147] twitter.com, "Twitter privacy policy," Accessed: June 17, 2019. [Online]. Available: twitter.com/en/privacy

[148] H. Hu, N. Phan, J. Geller, S. Iezzi, H. T. Vo, D. Dou, and S. A. Chun, "An ensemble deep learning model for drug abuse detection in sparse Twitter-Sphere," in *Proceedings of the 17th World Congress on Medical and Health Informatics*, vol. 264. IOS Press, 2019. doi: 10.3233/SHTI190204 pp. 163–167.

[149] Y. Zhao, A. Curtis, X. Ye, J. Yang, C. Ma, S. AL-Dohuki, F. Kamw, and S. Jamonnak, "Neighborvis," 2018, accessed: June 17, 2019. [Online]. Available: http://vis.cs.kent.edu/NeighborVis/index.html

[150] D. Kifer and A. Machanavajjhala, "No free lunch in data privacy," in *Proceedings of the ACM Special Interest Group on Management of Data International Conference on Management of Data*. ACM, 2011. doi: 10.1145/1989323.1989345 pp. 193–204.

[151] U. Erlingsson, V. Pihur, and A. Korolova, "Rappor: randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the ACM Special Interest Group on Security, Audit and Control Conference on Computer and Communications Security*. ACM, 2014. doi: 10.1145/2660267.2660348 pp. 1054–1067.

[152] C. Liu, S. Chakraborty, and P. Mittal, "Dependence makes you vulnberable: differential privacy under dependent tuples," in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium*. The Internet Society, 2016. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/dependence-makes-you-vulnerable-differential-privacy-under-dependent-tuples.pdf

[153] Y. Cao, M. Yoshikawa, Y. Xiao, and L. Xiong, "Quantifying differential privacy under temporal correlations," in *Proceedings of the IEEE 33rd International Conference on Data Engineering*, 2017. doi: 10.1109/ICDE.2017.132 pp. 821–832.

[154] T.-H. H. Chan, M. Li, E. Shi, and W. Xu, "Differentially private continual monitoring of heavy hitters from distributed streams," in *Proceedings of the International Symposium on Privacy Enhancing Technologies Symposium*, 2012, pp. 140–159.

[155] S. Song, K. Chaudhuri, and A. D. Sarwate, "Stochastic gradient descent with differentially private updates," in *Proceedings of the IEEE Global Conference on Signal and Information Processing*, 2013. doi: 10.1109/GlobalSIP.2013.6736861 pp. 245–248.

[156] P. Jain, P. Kothari, and A. Thakurta, "Differentially private online learning," in *Proceedings of the 25th Annual Conference on Learning Theory*, vol. 23, 2012, pp. 24.1–24.34. [Online]. Available: https://proceedings.mlr.press/v23/jain12.html

[157] X. Xiao, G. Wang, and J. Gehrke, "Differential privacy via wavelet transforms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 08, pp. 1200–1214, 08 2011. doi: 10.1109/TKDE.2010.247

[158] Y. Wang, X. Wu, and L. Wu, "Differential privacy preserving spectral graph analysis," in *Proceedings of the Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, 2013, pp. 329–340.

[159] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 26th Annual Advances in Neural Information Processing Systems*, 2012, pp. 1106–1114. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[160] G. B. Arfken and H.-J. Weber, *Mathematical Methods for Physicists*. San Diego, CA, USA: Academic Press Harcourt Brace Jovanovich, 1967.

[161] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett, "Functional mechanism: regression analysis under differential privacy," *Proceedings of the Very Large Data Base Endowment*, vol. 5, no. 11, pp. 1364–1375, 2012. doi: 10.14778/2350229.2350253

[162] P. Smolensky, *Information Processing in Dynamical Systems: foundations of Harmony Theory*. Cambridge, MA, USA: MIT Press, 1986, pp. 194–281.

[163] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009. doi: 10.1145/1553374.1553453 pp. 609–616.

[164] T. M. Apostol, *Calculus, Volume 1*. Hoboken, NJ, USA: John Wiley & Sons, 1991.

[165] K. Chatzikokolakis, M. E. Andrés, N. E. Bordenabe, and C. Palamidessi, "Broadening the scope of differential privacy using metrics," in *Proceedings of the 13th International Symposium on Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, vol. 7981. Springer, 2013. doi: 10.1007/978-3-642-39077-7_5 pp. 82–102.

[166] M. Abadi, Ú. Erlingsson, I. Goodfellow, H. B. McMahan, I. Mironov, N. Papernot, K. Talwar, and L. Zhang, "On the protection of private information in machine learning systems: two recent approches," in *Proceedings of the IEEE 30th Computer Security Foundations Symposium*, 2017, pp. 1–6. [Online]. Available: https://conferences.computer.org/sp/pdfs/csf/2017/3217a001.pdf

[167] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson, "Scalable private learning with PATE," in *Proceedings of the 6th International Conference on Learning Representations*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=rkZB1XbRZ

[168] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of the 3rd International Conference on Learning Representations*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6572

[169] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia II, X. Xing, X. Liu, and C. L. Giles, "Learning adversary-resistant deep neural networks," *Computing Research Repository*, 2016. [Online]. Available: http://arxiv.org/abs/1612.01401

[170] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2016. doi: 10.1109/SP.2016.41 pp. 582–597.

[171] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proceedings of the IEEE European Symposium on Security and Privacy*. IEEE, 2016. doi: 10.1109/EuroSP.2016.36 pp. 372–387.

[172] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," in *Proceedings of the 3rd International Conference on Learning Representations*, 2015. [Online]. Available: http://arxiv.org/abs/1412.5068

[173] N. Papernot and P. McDaniel, "Extending defensive distillation," *Computing Research Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1705.05264

[174] H. Hosseini, Y. Chen, S. Kannan, B. Zhang, and R. Poovendran, "Blocking transferability of adversarial examples in black-box learning systems," *Computing Research Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1703.04318

[175] J. H. Metzen, T. Genewein, V. Fischer, and B. Bischoff, "On detecting adversarial perturbations," in *Proceedings of the 5th International Conference on Learning Representations*. OpenReview.net, 2017. [Online]. Available: https://openreview .net/forum?id=SJzCSf9xg

[176] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. D. McDaniel, "On the (statistical) detection of adversarial examples," *Computing Research Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1702.06280

[177] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: detecting adversarial examples in deep neural networks," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium,*. The Internet Society, 2018. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018 _03A-4_Xu_paper.pdf

[178] M. Abbasi and C. Gagné, “Robustness to adversarial examples through an ensemble of specialists,” in *Proceedings of the 5th International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=S1cYxlSFx

[179] J. Gao, B. Wang, and Y. Qi, “Deepmask: masking DNN models for robustness against adversarial samples,” *Computing Research Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1702.06763

[180] F. Tramèr, A. Kurakin, N. Papernot, I. J. Goodfellow, D. Boneh, and P. D. McDaniel, “Ensemble adversarial training: attacks and defenses,” in *Proceedings of the 6th International Conference on Learning Representations*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=rkZvSe-RZ

[181] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” in *Proceedings of the 5th International Conference on Learning Representations*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=BJm4T4Kgx

[182] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” in *Proceedings of the 5th International Conference on Learning Representations*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=HJGU3Rodl

[183] J. Cohen, E. Rosenfeld, and Z. Kolter, “Certified adversarial robustness via randomized smoothing,” in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, 2019, pp. 1310–1320.

[184] B. Li, C. Chen, W. Wang, and L. Carin, “Second-order adversarial attack and certifiable robustness,” *Computing Research Repository*, 2018. [Online]. Available: http://arxiv.org/abs/1809.03113

[185] W. Rudin, *Principles of Mathematical Analysis*, 3rd ed. New York, NY, USA: McGraw-Hill New York, 1976. ISBN 007054235. [Online]. Available: http://www.loc.gov/catdir/toc/mh031/75017903.html

[186] Wikipedia, “Operator norm,” 2018, accessed: June 17, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Operator_norm

[187] C. Xie, Y. Wu, L. v. d. Maaten, A. L. Yuille, and K. He, “Feature denoising for improving adversarial robustness,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00059 pp. 501–509.

[188] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, “Trojaning attack on neural networks,” in *Proceedings of the 25th Annual Network and Distributed System Security Symposium*. The Internet Society, 2018. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-5_Liu_paper.pdf

[189] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Proceedings of the 32nd Conference on Neural Information Processing Systems*, 2018, pp. 6106–6116. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/22722a343513ed45f14905eb07621686-Abstract.html

[190] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *Computing Research Repository*, 2016. [Online]. Available: http://arxiv.org/abs/1602.05629

[191] TensorFlow, "TensorFlow SoftMax implementation," 2018, accessed: June 17, 2020. [Online]. Available: https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/python/ops/nn_impl.py

[192] D. Hendrycks and T. G. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *Proceedings of the 7th International Conference on Learning Representations*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=HJz6tiCqYm

[193] Baidu.com, "Fedcube," 2020, accessed: June 17, 2020. [Online]. Available: http://fedcube.baidu.com/

[194] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: identifying vulnerabilities in the machine learning model supply chain," *Computing Research Repository*, 2017. [Online]. Available: http://arxiv.org/abs/1708.06733

[195] R. Pang, H. Shen, X. Zhang, S. Ji, Y. Vorobeychik, X. Luo, A. Liu, and T. Wang, "A tale of evil twins: adversarial inputs versus poisoned models," in *Proceedings of the ACM Special Interest Group on Security, Audit and Control Conference on Computer and Communications Security*. ACM, 2020. doi: 10.1145/3372297.3417253 pp. 85–99.

[196] S. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: incremental classifier and representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.587 pp. 5533–5542.

[197] D. Lopez-Paz and M. Ranzato, "Gradient episodic memory for continual learning," in *Proceedings of the 29th Conference on Neural Information Processing Systems*, 2017, pp. 6467–6476. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/f87522788a2be2d171666752f97ddebb-Abstract.html

[198] S. Ebrahimi, F. Meier, R. Calandra, T. Darrell, and M. Rohrbach, "Adversarial continual learning," in *16th European Conference on Computer Vision*, ser. Lecture Notes in Computer Science, vol. 12356. Springer, 2020. doi: 10.1007/978-3-030-58621-8_23 pp. 386–402.

[199] M. Lécuyer, R. Spahn, K. Vodrahalli, R. Geambasu, and D. Hsu, “Privacy accounting and quality control in the sage differentially private ML platform,” *ACM Special Interest Group in Operating Systems Operating Systems Review*, vol. 53, no. 1, pp. 75–84, 2019. doi: 10.1145/3352020.3352032

[200] M. Joseph, A. Roth, J. Ullman, and B. Waggoner, “Local differential privacy for evolving data,” *Journal of Privacy and Confidentiality*, vol. 10, no. 1, 2020. doi: 10.29012/jpc.718

[201] H. Liu, F. Sun, and B. Fang, “Lifelong learning for heterogeneous multi-modal tasks,” in *Proceedings of the International Conference on Robotics and Automation*. IEEE, 2019. doi: 10.1109/ICRA.2019.8793517 pp. 6158–6164.

[202] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. Torr, “Riemannian walk for incremental learning: understanding forgetting and intransigence,” in *Proceedings of the 15th European Conference on Computer Vision*, ser. Lecture Notes in Computer Science, vol. 11215. Springer, 2018. doi: 10.1007/978-3-030-01252-6_33 pp. 556–572.

[203] B. Liu, L. Wang, and M. Liu, “Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4555–4562, 2019. doi: 10.1109/LRA.2019.2931179

[204] Amazon Web Services, “Lambda quotas,” 2021, accessed: June 18, 2021. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html

[205] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54, 2017, pp. 1273–1282.

[206] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016. doi: 10.1109/CVPR.2016.90 pp. 770–778.

[207] M. Lin, Q. Chen, and S. Yan, “Network in network,” in *Conference Track Proceedings of the 2nd International Conference on Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1312.4400

[208] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[209] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S. Kim, “Communication-efficient on-device machine learning: federated distillation and augmentation under non-iid private data,” *Computing Research Repository*, 2018. [Online]. Available: http://arxiv.org/abs/1811.11479

[210] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter," *Computing Research Repository*, 2019. [Online]. Available: http://arxiv.org/abs/1910.01108

[211] S. Caldas, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "LEAF: A benchmark for federated settings," *Computing Research Repository*, 2018. [Online]. Available: http://arxiv.org/abs/1812.01097

[212] X. Li, M. Jiang, X. Zhang, M. Kamp, and Q. Dou, "Fedbn: federated learning on non-iid features via local batch normalization," *Computing Research Repository*, 2021. [Online]. Available: https://arxiv.org/abs/2102.07623

[213] W. E. Donath and A. J. Hoffman, "Lower bounds for the partitioning of graphs," *IBM Journal of Research & Development*, vol. 17, no. 5, pp. 420–425, 09 1973. doi: 10.1147/rd.175.0420

[214] A. J. Enright, S. Van Dongen, and C. A. Ouzounis, "An efficient algorithm for large-scale detection of protein families," *Nucleic Acids Research*, vol. 30, no. 7, pp. 1575–1584, 2002. doi: 10.1093/nar/30.7.1575

[215] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Physical Review E*, vol. 70, no. 6, p. 066111, 2004. doi: 10.1103/PhysRevE.70.066111

[216] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008. doi: 10.1088/1742-5468/2008/10/p10008

[217] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On modularity clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 2, pp. 172–188, 2007. doi: 10.1109/TKDE.2007.190689

[218] J. Friedman, T. Hastie, R. Tibshirani *et al.*, *The Elements of Statistical Learning*. New York, NY, USA: Springer Series in Statistics, 2001, vol. 1.

[219] M. E. Celebi, *Partitional Clustering Algorithms*. New York, NY, USA: Springer International Publishing, 2014.

[220] T. N. Tran, K. Drab, and M. Daszykowski, "Revised dbscan algorithm to cluster data with dense adjacent clusters," *Chemometrics and Intelligent Laboratory Systems*, vol. 120, pp. 92–96, 2013. doi: https://doi.org/10.1016/j.chemolab.2012.11.006

[221] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: analysis and an algorithm," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. MIT Press, 2001, pp. 849–856.

[222] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. 12, pp. 7821–7826, 2002. doi: 10.1073/pnas.122653799

[223] M. E. J. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Physical review E*, vol. 69, no. 2, p. 026113, 2004. doi: 10.1103/PhysRevE.69.026113

[224] M. Rosvall and C. T. Bergstrom, “Maps of random walks on complex networks reveal community structure,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 105, no. 4, pp. 1118–1123, 2008. doi: 10.1073/pnas.0706851105. [Online]. Available: https://www.pnas.org/content/105/4/1118

[225] T. Nepusz, H. Yu, and A. Paccanaro, “Detecting overlapping protein complexes in protein-protein interaction networks,” *Nature Methods*, vol. 9, no. 5, pp. 471–472, 2012. doi: 10.1038/nmeth.1938

[226] S. Brohee and J. Van Helden, “Evaluation of clustering algorithms for protein-protein interaction networks,” *BMC Bioinformatics*, vol. 7, no. 1, pp. 1–19, 2006. doi: 10.1186/1471-2105-7-488

[227] G. D. Bader and C. W. V. Hogue, “An automated method for finding molecular complexes in large protein interaction networks,” *BMC Bioinformatics*, vol. 4, no. 1, pp. 1–27, 2003. doi: https://doi.org/10.1186/1471-2105-4-2

[228] J. Yang and J. Leskovec, “Defining and evaluating network communities based on ground-truth,” *Knowledge and Information Systems*, vol. 42, no. 1, pp. 181–213, 2015. doi: 10.1007/s10115-013-0693-z

[229] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, “Defining and identifying communities in networks,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. 9, pp. 2658–2663, 2004. doi: 10.1073/pnas.0400054101

[230] S. Fortunato, “Community detection in graphs,” *Physics Reports*, vol. 486, no. 3-5, pp. 75–174, 2010. doi: https://doi.org/10.1016/j.physrep.2009.11.002

[231] J. Shi and J. Malik, “Normalized cuts and image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000. doi: 10.1109/34.868688

[232] G. W. Flake, S. Lawrence, and C. L. Giles, “Efficient identification of web communities,” in *Proceedings of the 6th ACM Special Interest Group on Knowledge Discovery and Data Mining International Conference on Knowledge Discovery and Data Mining*. ACM, 2000. doi: 10.1145/347090.347121 pp. 150–160.

[233] M. E. J. Newman, “Modularity and community structure in networks,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 103, no. 23, pp. 8577–8582, 2006. doi: https://doi.org/10.1073/pnas.0601602103

[234] E. A. Leicht, P. Holme, and M. E. J. Newman, “Vertex similarity in networks,” *Physical Review E*, vol. 73, no. 2, p. 026120, 2006. doi: 10.1103/PhysRevE.73.026120