Differential Privacy in Image Classification using ResNet-20 and DP-SGD Optimization

Praveen Rangavajhula
Department of Computer Science
University of Georgia
Athens, GA, 30602
praveen.rangavajhula@uga.edu

Alexander Darwiche
Department of Computer Science
University of Georgia
Athens, GA, 30605
alexander.darwiche@uga.edu

Deven Allen
Department of Computer Science
University of Georgia
Athens, GA, 30605
dca09692@uga.edu

Abstract

This project proposes a differentially private image classification system using ResNet-20 with various optimizers, starting with Differentially Private Stochastic Gradient Descent (DP-SGD) as a baseline. We aim to make incremental improvements with additional optimization techniques, exploring both non-private and DP versions of optimizers, and justifying our choices based on prior work and their potential to outperform others. The project will focus on achieving competitive accuracy while satisfying privacy guarantees. Additionally, we are investigating ways to enhance accuracy by modifying optimizer components, such as gradient clipping (potentially using techniques like automatic clipping).

1 Introduction

The increase in prevalence of machine learning models, especially in image classification, has coincided with concerns over privacy [10]. Differential privacy (DP) specifically addresses these concerns by ensuring that models do not inadvertently leak sensitive information about individual data points. In this proposal, we will use the ResNet-20 model, which is well-suited for datasets like CIFAR-10 [6], and will implement and improve DP-SGD to achieve better privacy guarantees without significantly compromising accuracy.

2 Motivation and Problem Statement

In many real-world applications, the concerns over privacy leakage can hinder the deployment of machine learning models. Current state-of-the-art models like ResNet-20 achieve high accuracy but are vulnerable to attacks that could leak sensitive information. An example of a differentially private optimizer is Differentially Private Stochastic Gradient Descent (DP-SGD), which has been shown to effectively reduce privacy leaks, but has challenges in balancing model accuracy and privacy [1]. Even with the strong foundation that DP-SGD provides, we believe there is room for improvements that can achieve both higher accuracy and stronger privacy guarantees.

Preprint. Under review.

3 Methodology

3.1 Model Architecture: ResNet-20

We propose utilizing the ResNet-20 model [6] for CIFAR-10, a standard dataset for image classification tasks. We selected a 20-layer ResNet for its deep architecture and strength in image classification problems [5]. 20 layers should be enough depth to adequately model many features, while not encountering the higher training error encountered with excessively "deep" architectures. [5]

If necessary, we may modify the architecture slightly to optimize for DP compatibility.

3.2 Non-private Optimizers to Try

We propose trying 3 non-private optimizers to establish baseline performance on CIFAR-10.

- **SGD:** Standard Stochastic Gradient Descent (SGD) for baseline comparison. This optimizer works by calculating the gradient at each data point and updating the model parameters with the following update rule:

  $$\mathbf{x}_1 = \mathbf{x}_1 - \alpha \cdot \mathbf{g}_t$$

  where $\mathbf{x}_1$ is model parameters, α is the learning rate, and $\mathbf{g}_t$ is the gradient at that data point.
- **RMSprop:** Root Mean Square Propagation (RMSprop) builds on SGD by including the moving average factor. This factor functions by scaling the gradient each step, based on the gradient of the previous data points. This is done by scaling the gradient, at each model parameter update, by the moving average squared gradient:

  $$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)(\mathbf{g}_t^2 + \xi \mathbf{1}_d)$$

  where $\mathbf{v}_{t-1}$ is the squared gradient average from the previous step, $\mathbf{g}_t^2$ is the squared gradient of the current step, β_2 is the squared gradient moving average factor and $\xi \mathbf{1}_d$ is a constant vector [2, 7].
- **ADAM:** Adaptive Moment Estimation (ADAM) further build on RMSprop by including another moving average factor, this time for the gradient. In the general gradient update rule formula, instead of gradient, ADAM substitutes in the gradient moving average:

  $$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t$$

  where $\mathbf{m}_{t-1}$ is the gradient moving average from the previous step, $\mathbf{g}_t$ is the gradient of the current step, and β_1 is the gradient moving average factor [2].

3.3 Components of DP-SGD

We propose implementing DP-SGD as our baseline privacy-preserving algorithm.

- **Gradient Clipping:** Limits the influence of individual examples during training.
- **Noise Addition:** Adds noise to gradients to ensure privacy (via Opacus) [13].
- **Privacy Accounting:** We will use Rényi Differential Privacy (RDP) for privacy budget tracking [9].

3.4 Incremental Improvements

After we have privatized SGD, we propose making the following enhancements:

- **Upgrading to RMSprop:** Adding an adaptive learning rate by incorporating the moving average of gradients squared [2].
- **Upgrading to ADAM:** Adding a moving average for gradient (ADAM) [2].
- **Modification of clipping:** Automatic gradient clipping methods that adjusts the clipping threshold throughout training [3].

3.5 Rationale for Choosing DP-SGD

DP-SGD provides well-documented privacy guarantees [1] while maintaining decent utility for image classification tasks. The addition of noise and gradient clipping help ensure (ϵ, δ)-differential privacy, making it ideal for sensitive applications. Previous work shows that DP-SGD, when optimized, can yield near state-of-the-art accuracy for differentially private models [4].

3.6 Why This Approach Will Outperform Others

Our approach leverages the ResNet-20 model which is often used in conjunction with CIFAR-10 dataset. Similarly, DP-SGD is an often used optimization algorithm, that has proven powerful in balancing the privacy-utility trade-offs posed by differential privacy, as seen in [1]. With the baseline of DP-SGD and ResNet-20, we believe our incremental improvements will yield strong gains in accuracy. Converting from DP-SGD to DP-RMSprop may improve the accuracy by adapting the learning rate as the model trains. This is especially important given the limited amount of times we are able to query a dataset while implementing differential privacy. Additionally, we believe the added gradient normalization introduced by upgrading to DP-ADAM will similarly improve the rate of convergence of our model as seen in the non-private study [2]. Again, this is paramount given the limited number of times that our model can query the dataset. Finally, by experimenting with automatic clipping, we aim to find an optimal trade-off between accuracy and privacy.

3.7 Pseudocode for Non-Private Optimizers

Below are pseudocodes for the non-private SGD, non-private RMSprop, and non-private ADAM algorithms that we plan to privatize, adapted or referenced from [2]:

Algorithm 1 SGD Algorithm

Input: A step size α, initial starting point $\mathbf{x}_1 \in \mathbb{R}^d$, and access to a (possibly noisy) oracle for gradients of $f : \mathbb{R}^d \to \mathbb{R}$.
function SGD($\mathbf{x}_1, \alpha$)
    Initialize: $\mathbf{v}_0 = \mathbf{0}$
    for $t = 1, 2, \ldots$ **do**
        $\mathbf{g}_t = \nabla f(\mathbf{x}_t)$
        $\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \mathbf{g}_t$
    end for
end function

Algorithm 2 RMSProp

Input: A constant vector $\mathbb{R}^d \ni \xi \mathbf{1}_d \geq 0$, parameter $\beta_2 \in [0, 1)$, step size α, initial starting point $\mathbf{x}_1 \in \mathbb{R}^d$, and access to a (possibly noisy) oracle for gradients of $f : \mathbb{R}^d \to \mathbb{R}$.
function RMSPROP($\mathbf{x}_1, \beta_2, \alpha, \xi$)
    Initialize: $\mathbf{v}_0 = \mathbf{0}$
    for $t = 1, 2, \ldots$ **do**
        $\mathbf{g}_t = \nabla f(\mathbf{x}_t)$
        $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)(\mathbf{g}_t^2 + \xi \mathbf{1}_d)$
        $\mathbf{V}_t = \text{diag}(\mathbf{v}_t)$
        $\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \mathbf{V}_t^{-\frac{1}{2}} \mathbf{g}_t$
    end for
end function

Algorithm 3 ADAM

Input: A constant vector $\mathbb{R}^d \ni \xi \mathbf{1}_d > 0$, parameters $\beta_1, \beta_2 \in [0, 1)$, a sequence of step sizes $\{\alpha_t\}_{t=1,2,\ldots}$, initial starting point $\mathbf{x}_1 \in \mathbb{R}^d$, and oracle access to the gradients of $f : \mathbb{R}^d \to \mathbb{R}$.
function ADAM($\mathbf{x}_1, \beta_1, \beta_2, \{\alpha_t\}, \xi$)
  Initialize: $\mathbf{m}_0 = \mathbf{0}$, $\mathbf{v}_0 = \mathbf{0}$
  for $t = 1, 2, \ldots$ **do**
    $\mathbf{g}_t = \nabla f(\mathbf{x}_t)$
    $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t$
    $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2$
    $\mathbf{V}_t = \mathrm{diag}(\mathbf{v}_t)$
    $\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha_t \left(\mathbf{V}_t^{\frac{1}{2}} + \mathrm{diag}(\xi \mathbf{1}_d)\right)^{-1} \mathbf{m}_t$
  end for
end function

4 Experimental Setup

4.1 System Description

We will use PyTorch [11] for model implementation and training. The DP-SGD [1] implementation will be based on the Opacus library [13]. Training will be performed on GPUs available via our departmental server csci-cuda.cs.uga.edu or on Google Colab.

4.2 Dataset

We will use the CIFAR-10 dataset [8], consisting of 60,000 32x32 RGB images, which is commonly used for image classification tasks. The dataset is built-in in PyTorch [11], and we will load it using standard libraries.

4.3 Metrics to Compare

- **Accuracy/Training Loss:** Accuracy and training loss on CIFAR-10 [8].
- **Privacy Cost:** We will measure (ϵ, δ) using RDP [9] to ensure privacy compliance.
- **Training Time:** Time complexity and memory usage will be tracked.

4.4 Design of Experiments

We will be performing a series of experiments to evaluate our modified differentially private optimizers against other baseline models. Table 1 outlines the experimental design, including the optimizer, clipping method, noise mechanism, and other metrics.

Table 1: Experimental Design

Experiment ID	Optimizer	Clipping Method	Noise Mechanism	Accuracy	Training Time	Privacy Cost
1	DP-SGD	Standard	Standard Gaussian	TBD	TBD	TBD
2	DP-RMSprop	Standard	Standard Gaussian	TBD	TBD	TBD
3	DP-Adam	Standard	Standard Gaussian	TBD	TBD	TBD
4	DP-SGD	Automatic Clipping	Standard Gaussian	TBD	TBD	TBD
5	DP-RMSprop	Automatic Clipping	Standard Gaussian	TBD	TBD	TBD
6	DP-Adam	Automatic Clipping	Standard Gaussian	TBD	TBD	TBD

4.5 Baseline Models

We will compare the performance of our modified differentially private models with standard private optimizers, including vanilla DP-SGD [1], DP-RMSprop, and DP-Adam [14]. Additionally, we will benchmark against AdaClip [12] to evaluate the effectiveness of our automatic clipping and noise mechanism modifications.

5 Related Work

Several approaches to differentially private deep learning have been explored in the literature. Abadi et al. [1] introduced DP-SGD, which has become a foundational technique for privacy-preserving model training. Our focus is on making incremental improvements to this framework by adapting it to DP-RMSprop and DP-Adam. Additionally, we aim to incorporate automatic clipping [3].

6 Timeline and Milestones

- October 4, 2024: Best Accuracy Report 1 Due.
- October 18, 2024: Interim Report Due.
- November 8, 2024: Best Accuracy Report 2 Due.
- November 22, 2024: Final Report Due.

7 Best Accuracy Report #1

7.1 Current Implementation Overview

Table 2: Experimental Results
All experiments were conducted with a constant privacy budget $\delta = 10^{-5}$, momentum $\beta = 0.9$, weight decay $\lambda = 10^{-4}$ and a maximum gradient norm of $C = 1.0$.

Experiment ID	Optimizer	Epochs	Accuracy	Training Time (s)	Privacy Cost	Learning Rate	Batch Size	Noise Multiplier
1	SGD	100	87%	-	-	0.1	128	-
2	SGD	200	94%	-	-	0.1	128	-
3	**DP-SGD**	**30**	**41%**	**481.49**	**3**	**0.1**	**128**	**1.1**
4	DP-SGD	30	40%	598.68	3	0.2	128	1.1
5	DP-SGD	30	39%	527.37	3	0.3	128	1.1
6	DP-SGD	30	37%	584.55	3	0.4	128	1.1
7	DP-SGD	30	38%	597.60	3	0.5	128	1.1
8	DP-SGD	30	35%	995.68	3	0.1	64	1.1
9	**DP-SGD**	**30**	**44%**	**473.86**	**3**	**0.1**	**256**	**1.1**
10	DP-SGD	30	44%	597.29	2.99	0.1	512	1.1
11	DP-SGD	30	42%	677.04	3	0.1	1024	1.1
12	DP-SGD	30	43%	519.55	8.01	0.1	128	1.1
13	DP-SGD	30	44%	627.49	10.01	0.1	128	1.1
14	DP-SGD	30	48%	553.12	50.04	0.1	128	0.1
15	**DP-SGD**	**30**	**50%**	**375.37**	**50.04**	**0.1**	**256**	**0.1**

7.2 Best Observed Accuracy and Components/Hyperparameters

In the Experimental testing above, we varied hyperparameters to maximize accuracy of classification on the CIFAR10 dataset. We employed the (ϵ, δ)-Differentially Private - Stochastic Gradient Descent (DP-SGD) as our optimizer. For our loss function, we used the built-in PyTorch CrossEntropy function. The bolded lines, in the table above, indicate the maximum accuracies achieved by varying each of the hyperparameters. As a baseline, we also implemented a non-private SGD to give an upper bound on potential accuracy for the CIFAR10 dataset. We were able to achieve 94% accuracy with non-private SGD over 200 epochs.

- **Learning Rate:** We varied learning rate from 0.1 to 0.5. The smallest learning rate (0.1) yielded the highest accuracy of 41%.
- **Batch Size:** We varied batch size from 64 to 1024. A batch size of 256 yielded the highest accuracy of 44% among its peers.
- **Epsilon/Privacy Budget:** Higher epsilons mean lower privacy, but also can mean higher utility. When the epsilon was adjusted to 50, accuracy peaked at 50%.
- **Noise Multiplier:** Along with Epsilon changes, our highest accuracy (50%) run also coincided with a decrease in noise multiplier (0.1).

7.3 Failed Approaches

Below is discussion of the approaches that either failed to improve accuracy or reduced the accuracy from our first DP-SGD model run (Experiment ID #3).

- **Increasing learning rate:** Adjusting the learning rate from 0.3 to 0.5 still yielded lower accuracy, ranging between 37% and 39%.

- **Adjusting Batch Sizes:** Decreasing the batch size to 64 was a failure on 2 fronts. It first decreased the accuracy of the model to 35% and it was noticeably slower than all other approaches at more than 900 seconds to completion. Higher Batch Sizes, near 1000, also experienced some slowdown (more than 650 seconds). Any batch size other than the 256 or 512 seemed to have no positive effect on accuracy. CUDA also indicated a memory warning when batch size was 1024.

7.4 Implementation Challenges

In the course of completing these tests, we ran into numerous issues. These issues mainly during the implementation of the Opacus Privacy Engine. Our original implementation builds a Residual Network with 20 layers (Resnet20) and uses non-private SGD as its optimizer.

- **BatchNorm to GroupNorm:** Opacus was not compatible with BatchNorm (which was used in our base implementation [6]). To fix this, we employed Opacus' ModuleValidator.fix(model) built-in function, which replaces all the BatchNorms with GroupNorms.

- **Lambda Layer:** Next, we replaced the Lambda layer (from the base implementation [6]) with a Shortcut layer, as the Lambda layer was using Serializable functions which weren't compatible with ModuleValidator.fix(model). This occurs because Lambda layers use unnamed functions which are not serializable.

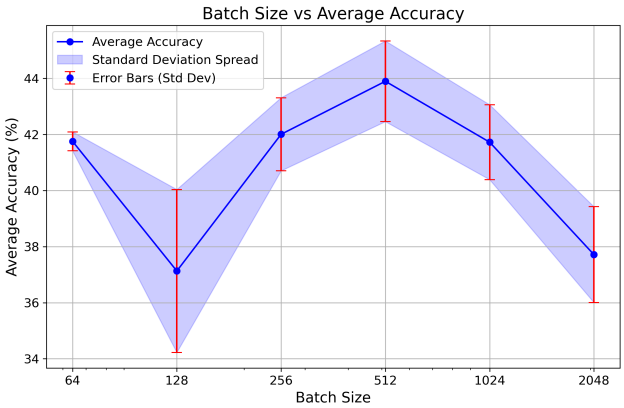
- **Dead Module:** Lastly, we encountered an error with a "dead module". Specifically, while calling the loss.backward() function, our model would return an RuntimeError indicating that we were trying to call the hook of a dead module. This arose due to the base implementations [6] usage of torch.nn.DataParallel. To fix this, we removed this parallelization from the code.

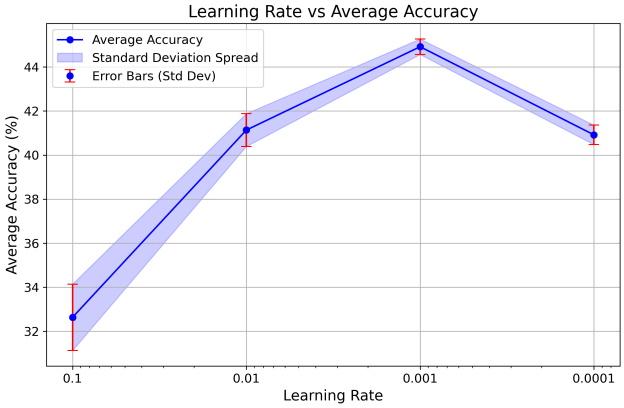
8 Best Accuracy Report #2

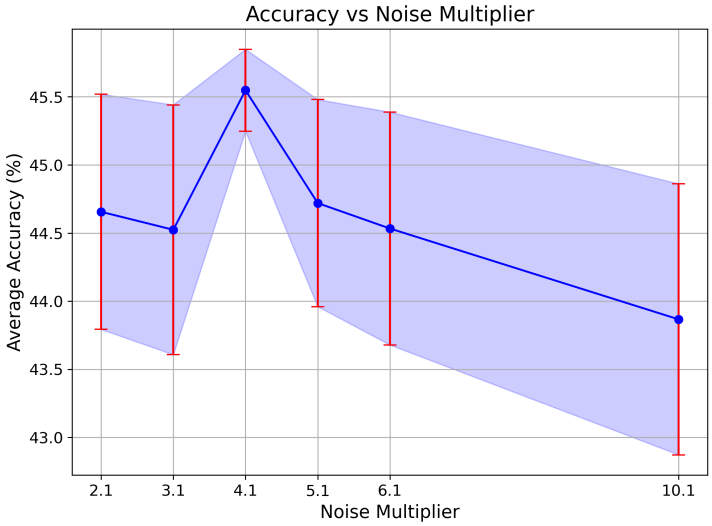
8.1 Current Implementation Overview

We performed multiple experiments to assess the effectiveness of our implementation by various different hyperparameters. All the detailed results from these experiments are presented in Appendix A (see Table 3). This table offers an overview of our findings and supports the analyses discussed in the following sections.

8.2 Visualizations







8.3 Best Observed Accuracy and Components/Hyperparameters

The biggest change from the BA report #1 was the privatization of the Lion Optimizer. To privatize, we made the gradients used in the algorithm "noisy" by introducing clipping at the per example gradient level and then adding noise for the batch. The above results indicate the first round of testing completed with the new DP-Lion Optimizer.

We began our testing with a grid-search of possible hyperparameter combinations for batch size, learning rate, and noise multiplier. The main takeaways were that batch size, when changed indepedently, appears to be optimal around 512 batch size. For Learning rate, it appeared that a learning rate of .001 showed the highest accuracy over a set of 5 runs. Finally, for noise multiplier, we found that raising the multiplier to 4.1 from 1.1 might provide some improvements in accuracy. We also found there to be marginal differences between higher and lower noise multipliers when using large batch sizes.

In this suite of testing, our highest accuracy was roughly 46%. We then turned to the original non-private Lion paper for some guidance on how they adjusted hyperparameters. They recommend large batch sizes and relatively small learning rates (relative to DP-Adam, DP-SGD, DP-RMSprop). They also recommend a $Beta_1$ value of 0.95 and a $Beta_2$ value of 0.98. We tested these beta values also with increasing the epochs from 30 to 200 and achieved a peak accuracy of 54.43%. This is the highest private accuracy we have been able to generate in our testing yet.

In our early testing, it seems that non-private Lion and Private Lion might prefer the same types of hyperparameters, namely large batch sizes, small learning rates, $Beta_1$ value of 0.95, and a $Beta_2$ value of 0.98.

- **Learning Rate:** We varied learning rate from 0.1 to 0.0001. The middle learning rate (0.001) yielded the highest accuracy of 45%.
- **Batch Size:** We varied batch size from 64 to 2048. A batch size of 512 yielded the highest accuracy of 44% among its peers.
- **Noise Multiplier:** Along with Epsilon changes, our highest accuracy, 54.43%, also coincided with an increase in noise multiplier (10.1).

8.4 Failed Approaches

It seems that using smaller batch sizes is not optimal for accuracy of Lion. We also notice that large learning rates like those used in DP-SGD degrade accuracy dramatically. Lastly, we found that $Beta_1$ value of 0.9 and a $Beta_2$ value of 0.999, which are the commonly used values for Adam, are not optimal for DP-Lion.

8.5 Implementation Challenges

We needed to find an implementation of Lion to be used in our model. We used the following non-private Lion implementation from the google automl repository: `https://github.com/google/automl/blob/master/lion/lion_pytorch.py`.

From there, we needed to then pass the optimizer through the opacus make private function to return the noisy gradients optimization. As we're looking to improve the accuracy of our models, we're needing to increase the batch size and the epochs the model runs for. This is causing us to hit a computing limit. Some of our runs take upwards of 3600 seconds (roughly 1 hour) and can use considerable memory as well. This isn't a persistent issue, but sometimes the runs will hit memory allocation or OOM issues on CUDA.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 308–318, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Amitabh Basu, Soham De, Anirbit Mukherjee, and Enayat Ullah. Convergence guarantees for rmsprop and ADAM in non-convex optimization and their comparison to nesterov acceleration on autoencoders. *CoRR*, abs/1807.06766, 2018.

[3] Zhiqi Bu, Yu-Xiang Wang, Sheng Zha, and George Karypis. Automatic clipping: Differentially private deep learning made easier and stronger, 2023.

[4] Soham De, Leonard Berrada, Jamie Hayes, Samuel L. Smith, and Borja Balle. Unlocking high-accuracy differentially private image classification through scale, 2022.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] Yerlan Idelbayev. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. `https://github.com/akamaster/pytorch_resnet_cifar10`. Accessed: 2024-09-17.

[7] Jason Jason Huang. Rmsprop, Oct 2020.

[8] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[9] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, August 2017.

[10] Nicolas Papernot and Thomas Steinke. Hyperparameter tuning with renyi differential privacy, 2022.

[11] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8026–8037, 2019.

[12] Venkatadheeraj Pichapati, Ananda Theertha Suresh, Felix X. Yu, Sashank J. Reddi, and Sanjiv Kumar. Adaclip: Adaptive clipping for private sgd, 2019.

[13] Yandex and Facebook AI Research. Opacus: User-friendly differential privacy library for pytorch. `https://github.com/pytorch/opacus`, 2021.

[14] Yingxue Zhou, Xiangyi Chen, Mingyi Hong, Zhiwei Steven Wu, and Arindam Banerjee. Private stochastic non-convex optimization: Adaptive algorithms and tighter generalization bounds, 2020.

GitHub Contributions

The code and related materials for this project are available at our GitHub repository: `https://github.com/CS8960-Privacy-Preserving-Data-Analysis/final-project`. Contributions, issues, and discussions are welcome.

A Detailed Experiment Results

Exp ID	Opt	Epochs	Avg Acc	Run 1		Run 2		Run 3		Run 4		Run 5		Privacy Cost	LR	Batch	Noise	Beta 1	Beta 2
				Acc	Time	Acc	Time	Acc	Time	Acc	Time	Acc	Time						
1	DP-Lion	30	42.01%	43.35%	489.8	43.67%	690.09	40.29%	677.16	41.62%	678.1	41.10%	677.23	7.99	0.003	256	1.1	0.9	0.999
2	DP-Lion	30	41.41%	41.11%	1093.4	41.95%	1088.26	41.99%	1110.42	42.00%	1116.9	41.72%	1139.94	7.99	0.0003	64	1.1	0.9	0.999
3	DP-Lion	30	37.13%	42.74%	707.22	34.50%	695.22	35.79%	664.77	36.90%	559.75	35.72%	566.71	7.99	0.0003	128	1.1	0.9	0.999
4	DP-Lion	30	43.90%	43.78%	765.43	44.46%	498.97	41.37%	500.11	45.79%	580.04	44.08%	642.37	7.99	0.0003	512	1.1	0.9	0.999
5	DP-Lion	30	41.72%	41.08%	953.57	41.80%	821.57	43.85%	774.07	39.77%	763.44	42.12%	541.74	7.99	0.0003	1024	1.1	0.9	0.999
6	DP-Lion	30	37.71%	37.49%	1364.81	38.20%	718.85	34.78%	688.51	40.08%	840.74	38.02%	694.35	7.99	0.0003	2048	1.1	0.9	0.999
7	DP-Lion	30	32.36%	32.72%	687.02	33.79%	387.59	34.33%	387.38	29.98%	429.58	30.96%	618.34	7.99	0.1	256	1.1	0.9	0.999
8	DP-Lion	30	40.99%	40.52%	693.77	40.62%	621.77	42.58%	404.77	40.97%	625.35	40.26%	390.1	7.99	0.01	256	1.1	0.9	0.999
9	DP-Lion	30	44.92%	45.21%	697.67	45.13%	489.4	45.09%	491.66	44.24%	488.25	44.91%	603.59	7.99	0.001	256	1.1	0.9	0.999
10	DP-Lion	30	40.88%	40.66%	688.58	41.77%	849.46	40.52%	500.54	40.76%	489.32	40.69%	493.58	7.99	0.0001	256	1.1	0.9	0.999
11	DP-Lion	30	44.66%	45.50%	1353.32	44.83%	713.72	43.07%	684.61	45.33%	700.32	44.55%	689.72	7.99	0.001	2048	2.1	0.9	0.999
12	DP-Lion	30	44.52%	45.16%	737.86	44.36%	692.99	45.23%	698.26	42.80%	681.08	45.07%	720.68	7.99	0.001	2048	3.1	0.9	0.999
13	DP-Lion	30	45.55%	46.10%	815.71	45.22%	910.33	45.59%	715.69	45.45%	774.95	45.38%	936.26	7.99	0.001	2048	4.1	0.9	0.999
14	DP-Lion	30	44.72%	45.30%	733.84	45.17%	704.37	44.78%	960.33	43.24%	698.53	45.11%	926.9	7.99	0.001	2048	5.1	0.9	0.999
15	DP-Lion	30	26.72%	45.04%	766.2	45.23%	722.57	43.33%	680.96	–	–	–	–	7.99	0.001	2048	6.1	0.9	0.999
16	DP-Lion	30	26.32%	45.22%	1044.46	42.86%	869.19	43.52%	827.31	–	–	–	–	7.99	0.001	2048	10.1	0.9	0.999
17	DP-Lion	30	42.40%	42.55%	574.75	41.53%	497.16	43.05%	487.76	42.10%	486.32	42.77%	480.27	3	0.001	256	1.1	0.9	0.999
18	DP-Lion	30	45.76%	45.53%	571.52	45.77%	353.65	45.66%	569.54	45.63%	516.23	46.20%	535.55	10	0.001	256	1.1	0.9	0.999
19	DP-Lion	30	50.59%	51.65%	536.85	50.31%	597.86	49.86%	428.46	50.37%	423.37	50.76%	421.25	50	0.001	256	1.1	0.9	0.999
20	DP-Lion	200	10.37%	51.86%	3549.89	–	–	–	–	–	–	–	–	7.99	0.001	2048	10.1	0.9	0.999
21	DP-Lion	200	10.32%	51.59%	3514.54	–	–	–	–	–	–	–	–	7.99	0.001	2048	4.1	0.9	0.999
22	DP-Lion	200	32.55%	54.43%	3544.98	54.09%	3848.46	54.24%	3788.61	–	–	–	–	7.99	0.001	2048	10.1	0.95	0.98

Table 3: Detailed Results of All Experiment Runs