Privacy-Preserving Optimizers for Image Classification

Hari Priya Muppidi
hm34746@uga.edu

Shriya Garlapati
@uga.edu

Abstract

Image classification is a critical task in deep learning, allowing for the identification and categorization of objects within images, and enabling high-accuracy interpretation of visual data. Differential privacy (DP) plays a vital role in safeguarding sensitive information within datasets, ensuring that individual data points cannot be inferred, even when models are exposed to adversarial queries. DP achieves this protection by introducing controlled noise during training, thus obscuring the influence of any single data point. Despite its benefits, past research has shown that DP-SGD, the most widely used DP training method, often leads to significant performance degradation, particularly in image classification tasks. Even with over-parameterized models, previous efforts have struggled to push accuracy beyond 70%, highlighting the need for strategies that can bridge this gap. In contrast, our work demonstrates that careful over-parameterized tuning can still perform effectively under different optimizers when combined with advanced techniques. Our study systematically adapts non-private optimizers such as SGD and RMSprop and evaluates their performance on CIFAR-10 dataset using ResNet-20, WRN-16-4, and EfficientNet models to improve accuracy. Our goal is to surpass the existing state-of-the-art accuracy of 70.7% % ($\varepsilon = 3.0, \delta = 10^{-5}$) and outperforms baseline methods like DP-SGD and DICE-SGD, marking a significant step toward enhanced image classification.

1 Introduction

Machine learning (ML) models have transformed a multitude of domains, from healthcare and finance to autonomous vehicles and natural language processing. Their capability to learn from vast datasets and generalize to unseen examples has made them indispensable. Notably, they are also integral to the development and deployment of large language models (LLMs), which have become central in applications ranging from conversational agents to content generation. As these models gain prominence, they also face significant vulnerabilities, particularly from adversarial attacks. However, this increasing reliance on ML also exposes critical vulnerabilities, particularly to adversarial attacks. These attacks involve subtle manipulations of input data that can significantly disrupt model predictions. As part of red teaming initiatives, researchers actively test these models against potential adversarial scenarios to identify weaknesses and strengthen defenses. The implications of such vulnerabilities are profound, especially when considering the potential for unauthorized access to sensitive individual data. Attackers may exploit these weaknesses to extract private information, compromising user privacy. Consequently, it is not only important to focus on improving model accuracy but also on fortifying models against adversarial threats. As the landscape of machine learning continues to evolve, developing robust defenses against such attacks is essential for maintaining trust and integrity in these systems.

Differential privacy (DP) offers a robust framework for protecting individual data in machine learning models by ensuring that the inclusion or exclusion of a single data point does not significantly impact the output of the model. This is achieved through the introduction of noise, which can obscure the

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

influence of any one individual's data on the model's learning process. However, the addition of noise presents a critical challenge: it introduces a trade-off between privacy and accuracy. On one hand, a higher level of noise can provide stronger privacy guarantees, making it difficult for adversaries to infer information about individuals. On the other hand, excessive noise can lead to reduced model performance, as the fundamental patterns within the data may become obscured. Finding an optimal balance between accuracy and privacy is a key research focus, as it directly influences the usability of differentially private models in practical scenarios. Furthermore, as regulations around data privacy tighten, the demand for effective differential privacy implementations that maintain utility while safeguarding personal information has never been higher.

Training DP-SGD involves gradient computation based on a subset of the training data at each iteration. To ensure differential privacy, noise is added to these gradients before they are used to update the model weights. This noise is typically drawn from a calibrated distribution, such as Gaussian, and its scale is determined by privacy parameters, specifically the privacy budget. One of the primary advantages of DP-SGD is its ability to provide strong privacy guarantees while still allowing the model to learn effectively from the data. A key aspect of DP-SGD is the concept of gradient clipping, which ensures that the influence of any single data point is limited before noise is added. Its straightforward implementation has made it a popular choice for many practitioners looking to incorporate differential privacy into their machine learning workflows. Training DP-RMSProp introduces an adaptive learning rate mechanism based on the moving average of past gradients. This approach helps stabilize training and improve convergence rates, allowing the model to adjust to varying landscape gradients more effectively. By leveraging the history of gradients, this method can lead to more efficient training, particularly in scenarios with complex datasets. Overall, DP-RMSProp represents a promising advancement in the quest for privacy-preserving machine learning.

In this work, we delve into two prominent optimization techniques: Differentially Private Stochastic Gradient Descent (DP-SGD) and Differentially Private RMSProp (DP-RMSProp). DP-SGD is a widely adopted method that integrates differential privacy into the stochastic gradient descent framework by adding noise to the gradients during the training process. This technique typically utilizes mini-batches, allowing for efficient computation while ensuring privacy. The algorithm's effectiveness hinges on the selection of hyperparameters, including batch size, learning rate, and the amount of noise injected. DP-RMSProp is a variant that adjusts the learning rate based on the moving average of past gradients, offering the potential for improved convergence rates. Our work also focuses on enhancing DP-RMSProp by carefully tuning these hyperparameters to achieve optimal performance in privacy-sensitive applications. Historically, DP-RMSProp has been underutilized compared to DP-SGD, primarily due to its complexity and the challenges in parameter tuning. Many practitioners prefer the more straightforward implementation of DP-SGD, but our findings suggest that with the right frameworks and methodologies, DP-RMSProp can provide significant advantages in certain contexts, particularly where adaptive learning rates are beneficial.

2 Related Work

The main purpose of SGD is to determine the model parameters that result in the highest accuracy for both training and test datasets. The gradient represents a vector that points in the direction of the steepest rise of the function at a given point. By moving in the opposite direction of the gradient, the algorithm gradually descends to lower values of the function, continuing until it reaches the function's minimum. The main advantage of DP-SGD is its ability to limit the influence of individual data points on the model, satisfying the principles of Differential privacy. Group normalization[12] and weight standardization[12] ensure DP-SGD can be applied to large-scale deep learning models. Batch normalization is avoided which includes communication between examples leading to violation of privacy principles. Parallel gradient computation is possible by using reweighted loss functions based on gradient norms, increasing training without sacrificing privacy[10]. DP-SGD carefully balances privacy and accuracy. Augmentation multiplicity[12] and larger batch sizes[12] reduce the variance caused by added noise, hence improving accuracy. Better model initialization is possible by using pre-training on non-private data. During private fine-tuning[12], this step can lead to faster convergence and higher accuracy. Usage of exponential moving averages for parameter averaging smoothens the learning process and helps in retaining accuracy[12].

By limiting the magnitude of each gradient, clipping[9] bounds the sensitivity of the optimization process to any individual data point. Noise (typically Gaussian)[9] is added to the clipped gradients to

introduce randomness that helps obscure the contribution of individual data points, making it harder for adversaries to infer any specific sample's influence. SGD fails to converge effectively due to inadequate learning rates or overcoming the problem of minima locations[8]. Adaptive learning rates allow the optimizer to update weights to smooth out noise effects ensuring faster convergence speed[8]. RMSprop maintains a moving average of squared gradients to stabilize training, especially when gradients are noisy. The square is performed elementwise, and thus preserves dimensions[8]. The square slows down the learning rate for weights with large gradients (avoids excessive oscillations) and increases it for weights with small gradients (speed up training)[8]. Here, it prevents oscillations that could harm convergence as it provides a measured and systematic approach to ensure that privacy guarantees are upheld throughout all the steps of the algorithm[8]. RMSProp provides reasonable accuracy when trained on the ResNet[6] model and CIFAR-10 and CIFAR-100 datasets[7]. There has been not much prior work on the use of RMSprop in the context of grouped gradient clipping[13] and AugMix[15] as together. RMSprop's adaptive nature makes it more efficient at balancing model accuracy and privacy, leading to better performance under the same privacy budget. In most of the cases, RMSprop proves to be a consistent optimizer in accuracy results. So, it is seen as a significant and promising direction to use this optimizer in our work.

3 Methodology

ResNet-20 is a deep convolutional neural network architecture that belongs to the Residual Network (ResNet) family, designed primarily for image classification tasks. It was introduced to address the problem of vanishing gradients in deep networks by using residual connections, which allow gradients to flow more easily through the network during backpropagation. ResNet-20 is composed of multiple residual blocks, each containing two convolutional layers. These blocks include skip connections that add the input of the block to its output, helping to mitigate the issues of gradient propagation in deeper architectures. The "20" in ResNet-20 refers to the total number of weight layers in the network. The architecture typically consists of an initial convolutional layer followed by three groups of residual blocks, with each group containing several blocks and increasing the number of filters. The residual block works with Group Normalization instead of the traditional Batch Normalization for better performance in scenarios where mini-batch sizes may vary, especially in differential privacy contexts. A simple CNN with various layers of convolutions implementing DPSGD. It is the standard choice for image classification tasks.

ResNet offers many more advantages compared with CNN- Under the noisy conditions imposed by Differential privacy mechanisms, deeper architectures can offer more stability during training. ResNet's ability to retain and propagate gradient information can help in better representational learning, which can be highly beneficial when the gradients are noisy. Data augmentation techniques work well with deeper architectures like ResNet. This helps in model generalization, which is important in a differentially private setting where the utility of training data might be limited due to added noise. Additional data variability is handled better in ResNets, avoiding the overfitting risks that the CNNs are more prone to.

Resnet is being used to demonstrate the accuracy-privacy trade-off in the RMSProp optimizer which forms the second part of our study. This ensures that both DPSGD and RMSProp are tested on the same architecture which can make it easier to isolate the effects of the optimizer on the privacy-accuracy trade-offs. Consistency in architectures makes sure that any observed difference is due to optimizers rather than differences between the model architectures.

Stochastic Gradient Descent (SGD) is a variation of the Gradient Descent algorithm, designed to optimize machine learning models more efficiently. Gradien Descent is an iterative optimization technique used to find the optimal value of an objective function, whether it's a minimum or maximum.

Pseudocode for SGD:

Input:

- η: Learning rate
- T: Number of iterations
- (X, y): Training dataset, where X is the matrix of features and y is the vector of corresponding labels

- w_0: Initial parameters vector

Output:

- w_T: Learned parameters after T iterations

Pseudo code:

1. Initialize w_0 randomly
2. For $t = 0$ to $T - 1$:
   (a) Randomly select a mini-batch B_t of training examples from the dataset (X, y)
   (b) Compute the stochastic gradient of the loss function L using the mini-batch B_t:

$$\nabla L(w_t; B_t) = \frac{1}{|B_t|} \sum_{(x_i, y_i) \in B_t} (w_t^\top x_i - y_i) x_i$$

   (c) Update the parameter vector w:

$$w_{t+1} = w_t - \eta \nabla L(w_t; B_t)$$

End for
Return the final parameters w_T.

RMSprop is widely known in the machine learning community for training neural networks due to its stable learning[1], faster convergence speed even in non-convex optimization problems[2] and using fewer hyper-parameters[3][4][5].

Pseudocode for RMSProp:

Require:

- Global learning rate η
- Decay rate ρ
- Initial parameter θ

Initialize: accumulation variable $r = 0$

While: Stopping criterion not met do

1. Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$.
2. Set $g = 0$
3. For $i = 1$ to m do
   (a) Compute gradient:

$$g \leftarrow g + \nabla_\theta L\left(f\left(x^{(i)}; \theta\right), y^{(i)}; \theta\right)$$

4. End for
5. Accumulate gradient:

$$r \leftarrow \rho r + (1 - \rho) g^2$$

6. Compute parameter update:
7. Compute parameter update:

$$\Delta\theta \leftarrow -\frac{\eta g}{\sqrt{r}} \quad (where \frac{1}{\sqrt{r}}\ applied\ element\ wise)$$

8. Apply update:

$$\theta \leftarrow \theta + \Delta\theta$$

End while

Privacy proof:
Strong privacy guarantees are ensured by employing DP through DPSGD. Differential privacy is maintained by clipping gradients to a predefined norm and adding Gaussian noise. Our implementation adheres to the privacy composition theorem, through which we can track the cumulative privacy loss across multiple iterations. Models trained on target epsilon values have resulted in a final epsilon value of 2.99 along with the training and testing accuracies.

4 Experiment Setup

The experiments were conducted using a T4 GPU on Google Colab, leveraging the Opacus deep learning framework for privacy-preserving training. The primary model utilized in the experiments was ResNet-20, a variant of the ResNet architecture that employs a residual learning framework, enabling effective training of very deep networks. During the training process, two optimizers were employed: Stochastic Gradient Descent (SGD) and RMSProp, allowing for diverse optimization strategies. The experiments focused on the CIFAR-10 dataset, a widely used benchmark for evaluating machine learning models. To enhance training stability and convergence, Group Normalization was implemented; this technique normalizes the channels by dividing them into groups, which is particularly beneficial for small batch sizes. The entire development environment was set up on Google Colab, chosen for its computational resources and ease of sharing experimental setups, with all code written in Python 3.

5 Results

Our initial experiments focused on building the ResNet architecture using the DP SGD optimizer. We introduced group normalization, which improved our test accuracy to 35% and test loss 1.8052. Subsequently, we implemented several additional techniques: parameter averaging, weight standardization, and various data augmentations. We also introduced a learning rate scheduler(CosineAnnealingLR), hyperparameter tuning, and grouped gradient clipping. As a result of these enhancements, we achieved a training accuracy of 32% and training loss 1.8646. The parameters used for this configuration included a batch size of 512, a learning rate of 0.01, iterations 50 and multiple augmentations.The training was conducted under differential privacy constraints, with a privacy budget defined by epsilon (ϵ) set at 3.0 and delta (δ) at 1×10^{-5} for both optimizers. We also employed the RMSprop optimizer with a learning rate of 1×10^{-3} and a maximum gradient norm of 1.2 to train our model. The noise multiplier is of the magnitude 1.0. The model was trained for different epochs 10, 20 25 epochs using a batch size of 256 for iterations ranging from 250-300. For data normalization, we used a mean and standard deviation of $[0.5]$. AugMix was incorporated as a data augmentation technique, configured with severity 3, mixture width 3, chain depth set to -1, and alpha at 1.0. Our architecture featured a ResNet-20 with Group Normalization, and we applied Grouped Gradient Clipping during training to ensure stability in gradient updates. Throughout the training process, we meticulously tracked accuracy and loss metrics, as well as the privacy epsilon after each epoch, to evaluate the model's performance on the CIFAR-10 dataset. The training process yielded a loss of 1.7544 and an accuracy of 41.25%, while the test performance showed a loss of 0.0069 and an accuracy of 42.18%. The calculated privacy budget indicated an average value of 2.98 for all the epochs (10, 20, 25).

6 Conclusion

Due to the vanishing gradient problem in CNN, the models were experiencing slow learning and poor performance. We wanted to try other deep learning models where adding more layers felt practical. As ResNet architectures use shortcut connections which skip one or more layers, due to which the gradient can flow more easily through the network, we wanted to add experiments with the ResNet model. We have begun enhancing our new architecture by incrementally incorporating various techniques to boost accuracy. Initially, we have used Tensorflow library to train Resnet-20, WRAN-16-4 and EfficientNet models using SGD and RMSProp optimizers and achieved an accuracy ranging from 35-55%. But, it was observed that Tensorflow does not have sufficient documentation for the resultant shape and dimension errors when integrating the optimizers(especially for RMSProp) with automatic clipping and AugMix. So, we have changed the library to Opacus and have started

training the models from scratch again. During this process, we have understood how DPSGD works to maintain privacy while training deep learning models. Experimentation with various models has shown that different noise scales impact the accuracy-privacy trade-off. We have observed how sensitive the model performance is with the choices made of key hyperparameters like noise multiplier, learning rate and batch size. Accuracy improved with the increase in number of epochs. In future, we would like to implement a dynamic noise scaling mechanism that adapts during training based on the performance, automatic clipping to compare with grouped gradient clipping. When the model tries to overfit, noise can be increased and reduced when the training seems stable. Privacy- accuracy trade-off can be optimized throughout the training process. We would like to also tune different learning rates to different layers for the ResNet based on the sensitivity of the layers. We might plan to train multiple ResNet models with DP SGD simultaneously and combine their predictions through a weighted voting system depending on the accuracy. We would like to experiment with a pre-trained ResNet model fine tuned on a new dataset with DPSGD.

7 References

1. S. Ruder. An overview of gradient descent optimization algorithms. ArXiv, 1609.04747, 2016.

2. De, S., Mukherjee, A., and Ullah, E. Convergence guarantees for RMSProp and ADAM in non-convex optimization and an empirical comparison to Nesterov acceleration. arXiv preprint arXiv:1807.06766, 2018.

3. D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl. On empirical comparisons of optimizers for deep learning. arXiv preprint arXiv:1910.05446, 2019.

4. Shi, N., Li, D., Hong, M., and Sun, R. Rmsprop converges with proper hyperparameter. In International Conference on Learning Representations, 2020.

5. Jinlan Liu, Dongpo Xu, Huisheng Zhang, and Danilo Mandic. On hyperparameter selection for guaranteed convergence of RMSProp. Cognitive Neurodynamics, pages 1-11 2022.

6. Verma P, Tripathi V, Pant B (2021) Comparison of different optimizers implemented on the deep learning architectures for COVID-19 classification, Materials Today: Proceedings, p S2214785321013316.

7. Suyesh Pandit and Sushil Kumar. Improvement in convolutional neural network for cifar-10 dataset image classification. International Journal of Computer Applications, 176:25-29, 2020.

8. Rahul Yedida, Snehanshu Saha, and Tejas Prashanth. 2021. LipschitzLR: Using theoretically computed adaptive learning rates for fast convergence. Applied Intelligence 51, 3 (2021), 1460-1478.

9. Abadi, Martin, 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, 308-18. CCS.

10. Lee, Jaewoo, and Daniel Kifer."Scaling up Differentially Private Deep learning with Fast Per-Example Gradient Clipping." Proceedings on Privacy Enhancing Technologies 2021, no. 1 (2021): 128-4.

11. De, Soham, Leonard Berrada, Jamie Hayes, Samuel L. Smith, and Borja Balle. "Unlocking High-Accuracy Differentially Private Image Classification through Scale," https://doi.org/10.48550/arXiv.2204.13650

12. I. Mironov. R'enyi differential privacy. Private communication, 2016

13. Liu H., Li C., Liu B., Wang P., Ge S., Wang W. Differentially private learning with grouped gradient clipping ACM Multimedia Asia (2021), pp. 1-7

14. Loshchilov, I.; Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. arXiv 2016, arXiv:1608.03983.

15. Hendrycks D., Mu N., Cubuk E.D., Zoph B., Gilmer J., Lakshminarayanan B. Augmix: A simple data processing method to improve robustness and uncertainty (2019) arXiv preprint arXiv:1912.02781

Final Report

1 Introduction

Differentially Private Adagrad (DP-Adagrad) integrates the Adagrad optimizer with differential privacy techniques to ensure that the learning process adheres to strict privacy constraints. It employs gradient clipping, noise addition, and adaptive learning rates while maintaining a rigorous accounting of privacy loss. It is well-suited for problems where the data or gradients are sparse, as the adaptive learning rate efficiently handles these scenarios. In this work, we delve into three prominent optimization techniques: Differentially Private Stochastic Gradient Descent (DP-SGD), Differentially Private RMSProp (DP-RMSProp) and Differentially Private AdaGrad(DP-AdaGrad). DP-Adagrad automatically adjusts the learning rate for each parameter based on its historical gradient accumulation. This ensures efficient updates for sparse features, improving convergence in scenarios with uneven data distribution. The algorithm's adaptability to sparse gradients makes it well-suited for datasets with high dimensionality and many sparse features. It is suitable for applications involving sensitive data. Gradient clipping reduces the impact of outlier gradients, ensuring that no single data point disproportionately affects the model updates. This also stabilizes training and improves robustness.

Due to the deviation of the clipped gradient distribution and disparate impact in the noise addition, there is a collective decline in model accuracy. To address these challenges, Grouped Gradient Clipping method is introduced. This approach mitigates clipping bias and disparate impact by dividing the gradients into multiple groups and clipping each group individually. By doing so, the loss of gradient information is significantly reduced, resulting in clipped gradients that more closely resemble the true gradient distribution. Hyperparameters are dynamically adjusted during training based on gradient norms and privacy budget consumption.

2 Related Work

Tighter generalization bounds[1] show how the population loss over an n-dimensional space can be minimized in stochastic non-convex optimization algorithms such as DP-RMSProp and DP-Adam. They provide privacy guarantee for knitting Machinery Spare Classification[2] for both the optimizers. There is limited existing research on hyperparameter tuning for DP-RMSProp and DP-AdaGrad. Existing methods focused on improving the accuracy of the models in different use-cases.

3 Methodology

3.1 Weight Standardization

Weight Standardization (WS)[3] is a normalization technique that smooths the loss landscape by standardizing the weights in convolutional layers. Unlike previous normalization methods that focus on activations, WS emphasizes the smoothing effects of weights beyond just length-direction decoupling. Theoretically, WS reduces the Lipschitz constants of the loss and the gradients, thereby smoothing the loss landscape and improving training.

In Weight Standardization, instead of directly optimizing the loss on the original weights w, the weights are reparameterized as a function of standardized weights $\hat{w}$, i.e., $w = g(\hat{w})$. Similar to Batch Normalization, WS controls the first and second moments of the weights of each output channel individually in convolutional layers. While many initialization methods also initialize weights in similar ways, WS standardizes the weights in a differentiable manner, aiming to normalize gradients during back-propagation.

Unlike initialization methods, WS does not include any affine transformation on $\hat{w}$. This is because it assumes that normalization layers, such as Batch Normalization (BN) or Group Normalization (GN), will normalize this convolutional layer again.So, we have used this method in our training procedure to increase the accuracy.

Wide ResNet-16-4 is a variant of the ResNet architecture that focuses on increasing the width (number of filters) of each layer rather than its depth to improve representational capacity. It consists of 16 layers with a width multiplier of 4, resulting in more parameters per layer and better feature extraction.

This architecture is particularly effective for small datasets like CIFAR-10, offering a balance between computational efficiency and accuracy.EfficientNet is a family of convolutional neural networks designed using a compound scaling method that balances network depth, width, and resolution for optimal performance. It employs depthwise separable convolutions and squeeze-and-excitation layers, making it highly efficient in terms of both computation and memory. EfficientNet achieves state-of-the-art accuracy on several benchmarks while significantly reducing the number of parameters compared to traditional architectures.

3.2 Exponential Moving Average (EMA)

Exponential Moving Average (EMA) is a model averaging technique used in machine learning to improve the stability and generalization of models during training. Instead of using the raw model parameters for evaluation, EMA maintains a smoothed version of the model parameters by calculating an exponentially weighted moving average of these parameters throughout the training process. Firstly, EMA parameters are initialized with the model's initial parameters and a decay rate is set. At each training step, update the EMA parameters. Then the parameters are smoothed by giving less weight to current updates and more to historical values, reducing noise. During evaluation, EMA parameters are used instead of raw parameters for more stable metrics.

3.3 Grouped Gradient Clipping

Grouped gradient clipping[4] is used to modulate the gradient weights. It incorporates the smooth sensitivity mechanism into differentially private deep learning paradigm, which bounds the adding Gaussian noise. In this way, the resulting model can simultaneously provide with strong privacy protection and avoid accuracy degradation, providing a good trade-off between privacy and performance. Grouped Gradient Clipping focuses on stabilizing training by managing gradients across parameter groups, while DP-SGD is explicitly designed for differential privacy by clipping sample-level gradients and adding noise. GGC can be combined with DP-SGD for better control of gradients in privacy-preserving scenarios.

Algorithm:

- **Initialize Parameters:** Initialize the model parameters (ω) randomly. Define the following hyperparameters:
  - L: The group size, determining the number of data points sampled in each batch.
  - η_r: The learning rate used for updating the model parameters.
  - B: The gradient norm bound, ensuring that each group's L2 norm does not exceed this value.
  - σ: The noise scale, determining the standard deviation of the Gaussian noise added to gradients.
  - k: The number of groups into which the gradients are divided.
- **Gradient Computation:** For each batch, compute the gradients $g_t(x_i)$ of the loss function $\mathcal{L}(\omega, x_i)$ with respect to the model parameters:
  - $g_t(x_i)$: The gradient of the loss function for the data point x_i at iteration t.
  - $\mathcal{L}(\omega, x_i)$: The loss function with model parameters ω and input x_i.
- **Grouping and Clipping:** Divide the computed gradients into k groups:
  - Clip each group individually to ensure the L2 norm does not exceed the bound B:

$$\bar{g}_t(x_i) = g_t(x_i) \cdot \min\left(1, \frac{B}{\|g_t(x_i)\|_2}\right).$$

  - $\|g_t(x_i)\|_2$: The L2 norm of the gradient $g_t(x_i)$.
  - $\bar{g}_t(x_i)$: The clipped gradient for the data point x_i.

  Merge the clipped gradients into a single representation for the batch.

- **Adding Noise:** Add Gaussian noise scaled by σ to the averaged clipped gradients to ensure differential privacy:
$$\tilde{g}_t = \bar{g}_t + \mathcal{N}(0, \sigma^2 B^2 \mathbf{I}),$$
where:
  - $\mathcal{N}(0, \sigma^2 B^2 \mathbf{I})$: Gaussian noise with mean 0 and variance $\sigma^2 B^2$.
  - $\tilde{g}_t$: The noisy gradient used for the parameter update.
- **Parameter Update:** Update the model parameters using the noisy gradients in a descent step:
$$\omega_{t+1} = \omega_t - \eta_t \bar{g}_t,$$
where:
  - ω_t: The model parameters at iteration t.
  - ω_{t+1}: The updated model parameters at the next iteration.
  - η_t: The learning rate at iteration t.
  - $\bar{g}_t$: The averaged clipped gradients (with noise).

3.4 AdaGrad Optimizer

Adagrad (Adaptive Gradient) is an optimization algorithm designed to adapt the learning rate for each parameter based on its historical gradients. This approach allows Adagrad to improve the efficiency of machine learning models, especially in scenarios where sparse data is involved or when different parameters have varying rates of convergence. It provides higher learning rates for less frequent features and lower rates for more frequent ones, which makes it well-suited for working with sparse data. Additionally, it reduces the need to manually adjust the learning rate.

DP-AdagradAlgorithm:

- **Initialization:** Initialize the model parameters (ω) randomly. Set the accumulation vector ($\mathbf{v}_{(0)} = 0$) to store the sum of squared noisy gradients for adapting learning rates. Define the following hyperparameters:
  - η: The learning rate.
  - B: Gradient norm bound for clipping.
  - σ: Noise scale for ensuring privacy.
  - ϵ: A small constant for numerical stability.
- **Batch Sampling:** At each iteration, select a random batch S from the dataset with a sampling probability:
$$q = \frac{L}{m},$$
where:
  - L: Size of the batch.
  - m: Total size of the dataset.
- **Gradient Computation:** For each data point $x_i \in S$, compute the gradient of the loss function:
$$g_t(x_i) = \nabla \mathcal{L}(\omega_{(t)}, x_i),$$
where:
  - $\mathcal{L}(\omega, x_i)$: The loss function with model parameters ω and input x_i.
  - $g_t(x_i)$: The gradient of the loss function for the data point x_i at iteration t.
- **Gradient Clipping:** Clip each gradient to ensure its L2 norm does not exceed the bound B:
$$\bar{g}_t(x_i) = g_t(x_i) \cdot \min\left(1, \frac{B}{\|g_t(x_i)\|_2}\right),$$
where:

  - $\|g_t(x_i)\|_2$: The L2 norm of the gradient $g_t(x_i)$.
  - $\bar{g}_t(x_i)$: The clipped gradient for the data point x_i.
- **Aggregate Gradients:** Compute the average of the clipped gradients:

$$\bar{g}_t = \frac{1}{|S|} \sum_{x_i \in S} \bar{g}_t(x_i),$$

  where:
  - $|S|$: The size of the batch.
  - $\bar{g}_t$: The averaged clipped gradient.
- **Add Noise for Privacy:** Add Gaussian noise to the aggregated gradient to ensure differential privacy:

$$\tilde{g}_t = \bar{g}_t + \mathcal{N}(0, \sigma^2 B^2 \mathbf{I}),$$

  where:
  - $\mathcal{N}(0, \sigma^2 B^2 \mathbf{I})$: Gaussian noise with mean 0 and variance $\sigma^2 B^2$.
  - $\tilde{g}_t$: The noisy gradient.
- **Update Accumulation Vector:** Update the accumulation vector to include the squared noisy gradients:

$$\mathbf{v}_{(t)} = \mathbf{v}_{(t-1)} + \tilde{g}_t^2,$$

  where:
  - $\mathbf{v}_{(t)}$: The updated accumulation vector at iteration t.
  - $\tilde{g}_t^2$: The squared noisy gradient.
- **Update Parameters:** Use the Adagrad update rule to adjust the model parameters:

$$\omega_{(t+1)} = \omega_{(t)} - \eta \cdot \frac{\tilde{g}_t}{\sqrt{\mathbf{v}_{(t)}} + \epsilon},$$

  where:
  - $\omega_{(t+1)}$: The updated model parameters.
  - $\sqrt{\mathbf{v}_{(t)}} + \epsilon$: The adaptive learning rate for each parameter.
- **Repeat:** Repeat the steps for a total of T iterations, where T is the maximum number of training iterations.

Per-Layer Noise Scaling

Per-layer noise scaling introduces noise at each layer of a model with a scale that depends on the layer's depth. This is an alternative to adding uniform noise across all layers, which may not account for the varying contributions of different layers to the overall model performance.

The noise scaling is implemented as follows in our DP-RMSProp and DP-Adagrad algorithms when applied in grouped gradient clipping:

- Iterate over the model parameters, identifying each layer by its index (`layer_idx`).
- For each parameter gradient that is not `None`, calculate the noise scale for the layer:

$$\text{layer\_noise} = \frac{\text{NOISE\_MULTIPLIER}}{\text{layer\_idx} + 1}.$$

- Sample noise from a Gaussian distribution with:

$$\text{mean} = 0, \quad \text{std} = \text{layer\_noise} \times \text{bound}.$$

- Add the noise to the parameter gradient (`param.grad`).

4 Experimental Setup

4.1 DP-SGD

A Wide Residual Network (WRN-16-4) model was built from scratch. This model can balance computational efficiency and performance by increasing the width of residual networks. This model was chosen because of its capacity to increase feature learning without increasing the depth significantly.

Differential privacy is incorporated into the model by modifying the SGD optimizer to a differentially private stochastic descent gradient to limit the impact of any individual data point on the gradients of the model through gradient clipping and noise addition.

Modifications introduced into the model can include the removal of batch normalization layers because batch normalization depends on statistics computed over the complete batch. There is a chance that sensitive information is leaked if batch normalization layers are included in the model.

This implementation is inspired by various research papers publisher on Differential privacy like "Deep Learning with Differential Privacy" by Abadi et al. [1]. The WRN-16-4, trained with a batch size of 256 and a learning rate of 0.1 in this approach has resulted in a test accuracy of 26.3

Group normalization layers are added on top of the previous model to improve accuracy while adhering to DP-constraints. Group normalization was selected because it computes normalization statistics independently for predefined groups of channels rather than across the batch, ensuring compliance with DP-constraints while stabilizing and accelerating training. This modification aims to increase the performance of the trained model. GN was applied to all layers where batch normalization existed. Group size was empirically chosen to balance the trade-off between computational cost and performance. The batch size and learning rate remain 256 and 0.1. The resulting model has yielded a test accuracy of 37.74

The batch size is increased to 512 to promote gradient stability. Larger batch sizes can reduce variance in gradient updates. Noise is now distributed over a large set of examples while maintaining a fixed privacy budget. After 50 epochs, these were the model's metrics: Train Loss: 1.6054, Train Accuracy: 45.40

Weight standardization and parameter averaging are further added to the model to increase its performance. Weight standardization technique is used to normalize the weights within each layer, due to which training stability and convergence will improve. Weight standardization helps maintain robust updates, especially when differential privacy noise is added to gradients. Parameter averaging ensures that parameters are periodically averaged to counter instability in training, which is introduced due to the added noise, making the optimization trajectory smoother. The model is trained with weight standardization and parameter averaging with a batch size of 256 and a learning rate of 0.01. This has led to a significant increase in test accuracy of 49.09

Grouped gradient clipping and learning rate scheduler are integrated into the model. Grouped Gradient Clipping is used to control the gradient sensitivity by clipping gradients in groups, improving stability and ensuring adherence to privacy constraints. Cosine Annealing Scheduler is used to periodically decay the learning rate. The learning rate began at 0.1 and was progressively reduced using a cosine function, reaching a value of 0.000199 by epoch 50. The model is trained with all these features at a batch size of 128 and has yielded a test accuracy of: 48.29A dynamic noise multiplier is further added to this model which aims to increase the noise multiplier when the training loss is low and decrease the noise multiplier when the training loss is high. By updating the noise multiplier based on the training loss, the function adapts the training process to focus on learning (low noise) when the model is undertrained and prioritize privacy (high noise) as the model converges. The final model with all the above features is trained with a batch size of 256 and learning rate of 0.01. These values for the batch size and learning rate have been chosen after experimenting with many values of batch sizes and learning rates with the intermediate and final models. Some of the results of the intermediate models are mentioned with different learning rates and batch sizes to give insights into how the intermediate models are performing with different learning rates and batch sizes. The final model with all the above features has yielded a test accuracy of 48.29The intermediate model with group normalization, weight standardization, and parameter averaging, trained with a batch size of 256 and a learning rate of 0.01 has yielded the best test accuracy of: 49.09

4.2 DP-RMSProp Optimizer and DP-AdaGrad Optimizer

CIFAR-10 dataset consists of 50,000 training samples and 10,000 test samples. Images are normalized using a mean of $[0.4914, 0.4822, 0.4465]$ and a standard deviation of $[0.2023, 0.1994, 0.2010]$. Data is batched with a size of 4096 for both training and testing, while smaller physical batches of 128 are used to optimize memory usage. ResNet-20 is implemented with three stages, each containing three blocks. Each block is a BasicBlockGN, which incorporates Weight Standardized Convolutional Layers (WSConv2d) that normalize the convolutional kernel weights for stability, Group Normalization that normalizes intermediate feature maps by dividing them into groups, and Residual Connections, which create shortcut connections to skip convolutional layers for efficient gradient flow. The activation function used is SiLU (Sigmoid Linear Unit), which introduces non-linearity. An average pooling layer reduces feature map dimensions before passing them to a fully connected layer for classification.

WRN-16-4 extends standard ResNets by increasing the width (number of filters) rather than depth. WRN-16-4 has 16 layers, each composed of three stages and WideBasicGN blocks. The width multiplier quadruples the number of filters in each layer compared to the original ResNet. Dropout is applied within each block to improve generalization, especially on datasets like CIFAR-10. Weight Standardization (WSConv2d) is utilized for stable training, and Group Normalization ensures better handling of smaller batch sizes. EfficientNet-B0 is the base model pretrained on ImageNet, adapted for CIFAR-10. It uses depthwise separable convolutions and squeeze-and-excitation layers to reduce computation. The final fully connected layer is replaced with a custom classifier for 10 classes.

The optimizers used are RMSProp and Adagrad, both configured with a learning rate of 8×10^{-4}. RMSProp additionally employs a momentum of 0.9 and a weight decay of 1×10^{-4}. A cosine annealing scheduler reduces the learning rate from 8×10^{-4} to 9×10^{-5} over 50 epochs. Differential privacy is enforced using Opacus's PrivacyEngine, targeting a privacy budget of $\epsilon = 3.0$ and $\delta = 1 \times 10^{-5}$. Gradients are clipped to a maximum L2-norm of 1.5, and Gaussian noise is added with a noise multiplier of 1.1 to ensure privacy. Additionally, gradients are divided into 40 groups and clipped individually using a Grouped Gradient Clipping strategy, with each group scaled by a bound of 1.0.

To stabilize training, Exponential Moving Average (EMA) with a decay factor of 0.95 is applied to maintain smoothed model weights. Training involves calculating cross-entropy loss and applying grouped gradient clipping with 40 groups and noise addition during the backward pass, followed by an optimizer step and EMA update. Evaluation uses the EMA-smoothed model, computing metrics such as test accuracy and loss. Metrics logged per epoch include train/test accuracy and the privacy parameter (ϵ). After training for 50 epochs, the final privacy parameter is printed to confirm adherence to the privacy budget. Visualizations include plots of test accuracy against the privacy parameter (ϵ) and train/test accuracy against epochs, highlighting the trade-off between privacy and utility. The number of iterations for each epoch are in the range of 390 – 499.

5 Results and Discussions

Table 1: Variability across hyper-parameters. For each model, we report the minimum, maximum, median, and median absolute deviation (MAD) in test accuracy (%) achieved for a DP budget of $(\epsilon = 3, \delta = 10^{-5})$. These accuracies are the averages of five runs.

	DP-RMSProp				**DP-SGD**				**DP-AdaGrad**			
Model	**Min**	**Max**	**Median**	**MAD**	**Min**	**Max**	**Median**	**MAD**	**Min**	**Max**	**Median**	**MAD**
Resnet-20	58.6	59.5	59.0	0.4	47.1	47.6	47.6	0.2	35.4	37.3	36.9	0.7
WRN-16-4	50.8	51.1	50.9	0.1	49.2	50.1	49.1	0.4	30.2	31.6	30.6	0.5
EfficientNet	48.4	49.6	49.1	0.6	24.6	25.4	25.1	0.4	17.3	19.6	18.2	0.8

Highest accuracy observed was for the Resnet-20 model in DP-RMSProp with 60%. It is mainly due to the adaptive learning rate of RMSProp, grouped gradient clipping,parameter averaging and weight standardization of Resnet-20 architecture. It was observed that Efficient Net has lower accuracy when compared to that of other models due to absence of multiple layers and weight standardization. The three models are also trained using baselines DP-SGD and DiceSGD. It was observed that

baseline DP-SGD produced an accuracy of 26.3 % for Resnet-20 and 29.58 % for WRN-16-4. DiceSGD generated an accuracy of 40.7% for Resnet-20 and 41.3% for WRN-16-4. Figures 1 and 2 illustrate the training and testing accuracies across epochs and the test accuracy against privacy budgets (ϵ-DP) for ResNet-20 and WRN-16-4 models under RMSProp and AdaGrad optimization. Subfigure (a) in both figures shows how training and testing accuracies improve steadily over epochs, reflecting learning stability. Subfigure (b) demonstrates the impact of increasing ϵ-DP on test accuracy, highlighting the trade-off between privacy and performance. These graphs emphasize the effectiveness of grouped gradient clipping, weight standardization, and exponential moving average in improving model accuracy while maintaining privacy.

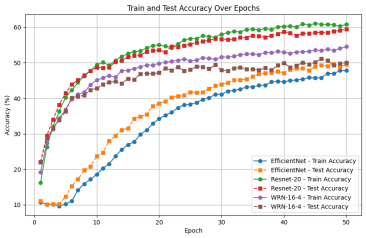
DP-SGD

5.1 ResNet

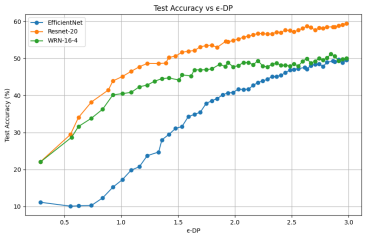
A Resnet model is built from scratch. Differential Privacy is incorporated into the model through gradient clipping and Noise addition. Batch Normalization layers are removed from the model to adhere to DP-constraints. The test accuracy of this model is found to be 29.58%. This model is trained with a batch size of 256 and a learning rate of 0.1. Group Normalization layers are added to this model to stabilize training and improve performance. This resulted in a test accuracy of 38.11% Further, weight standardization is added to the model which resulted in an accuracy of 38.13%. Parameter averaging is added to the same model and the batch size is increased to 512. This resulted in a test accuracy of 44.04%. Further, the batch size is increased to 1024, which increased the test accuracy to 46.76%. The batch size is maintained at 1024 and the addition of grouped gradient clipping resulted in an accuracy of 47.61%, which is the best accuracy obtained for ResNet.

5.2 WRN-16-4

A WRN-16-4 model is built from scratch and Differential Privacy is incorporated into it. Batch Normalization layers are removed. The accuracyof this model is 26.3%. Group Normalization layers are added to stabilize learning which resulted in a test accuracy of 37.74%. When the batch siza increased from 256 to 512, the accuracy is found to increase to 45.47%. Different learning rates are empirically evaluated and the batch size of 256 and a learning rate of 0.01 is found to give good results across the experiments. While deciding upon these values, the trade-off between performance and computational requirements is considered. Though the batch size of 512 yielded better test accuracy in few cases, we found it to be computationally very expensive in the other cases. Further, weight standardization and parameter averaging are added to the same model which increased the test accuracy to 49.09%. The batch size is fixed at 256 and the learning rate considered is 0.01. This is the best accuracy we obtained for WRN-16-4 model with Differential Privacy trained on the CIFAR-10 dataset. We have continued our experiments by incrementally adding grouped gradient clipping and learning rate scheduler which has resulted in an accuracy of 48.47%. We have added a dynamic noise multiplier which would add less noise when the training loss is high to give scope for the model to learn and add more noise when the training loss is less to maintain the privacy constraints. This addition has resulted in an accuracy of 48.29%.



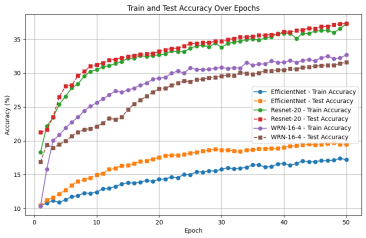
(a) Train and test accuracy against epochs



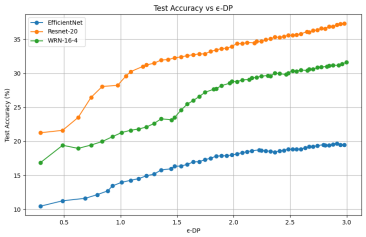
(b) Test Accuracy against ϵ-DP

(c) Test Accuracy against ϵ-DP

Figure 1: RMSProp

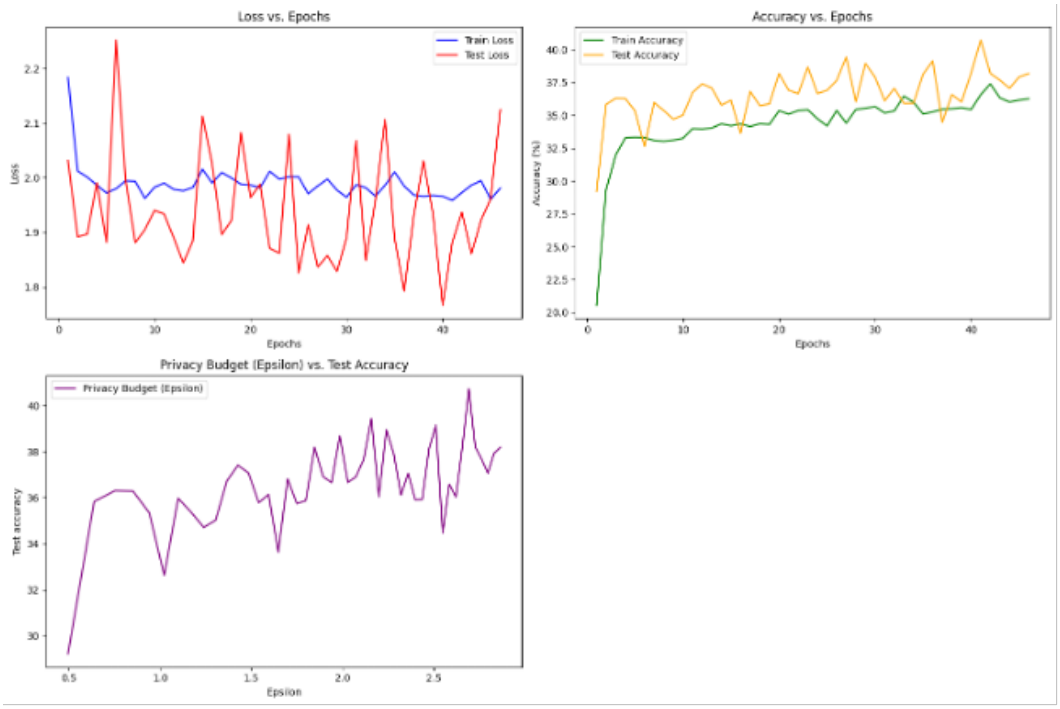


(a) Train and test accuracy against epochs

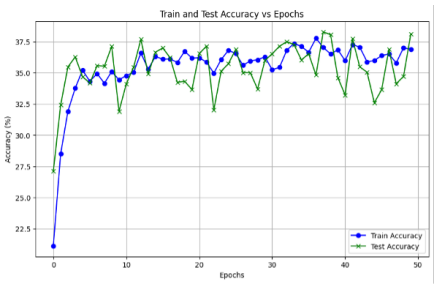


(b) Test Accuracy against ϵ-DP

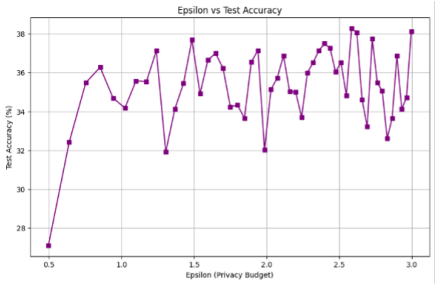
Figure 2: AdaGrad



(a) RESNET with DP before the addition of Group normalization



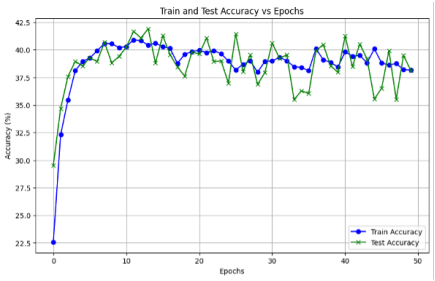
(a) Test Accuracy against epochs



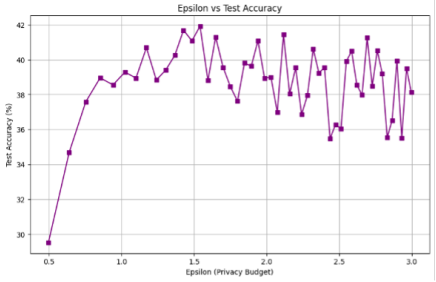
(b) Test Accuracy against ϵ-DP

(c) Test and Train loss against epochs

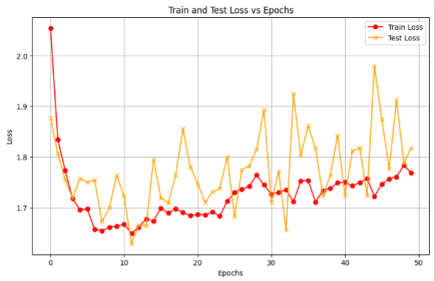
Figure 4: Resnet with DP after the addition of Group Normalization



(a) Test and Train Accuracy against epochs

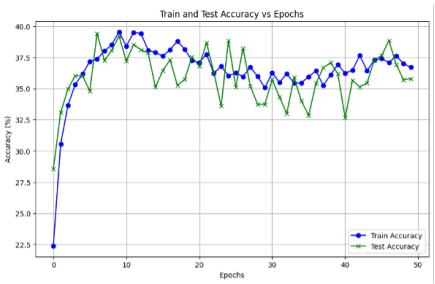


(b) Test Accuracy against ϵ-DP

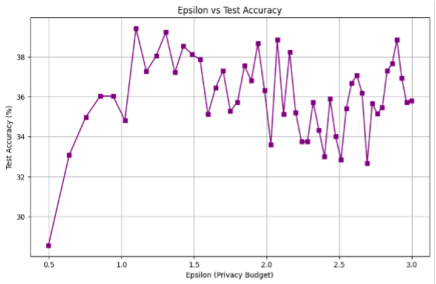


(c) Test and Train loss against epochs

Figure 5: Resnet with DP after the addition of Weight standarization



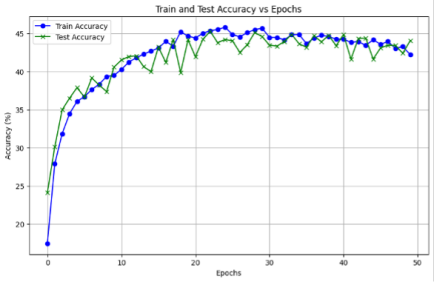
(a) Test and Train Accuracy against epochs



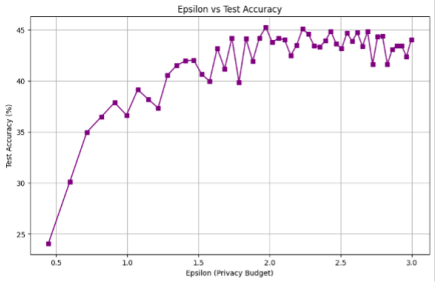
(b) Test Accuracy against ϵ-DP

(c) Test and Train loss against epochs

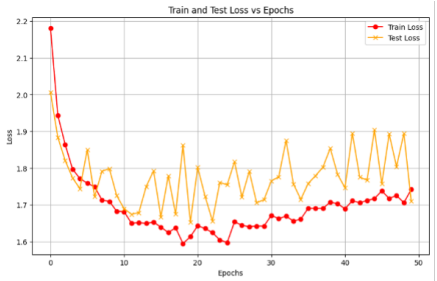
Figure 6: Resnet with DP after the addition of Parameter addition: Batch size: 256



(a) Test and Train Accuracy against epochs

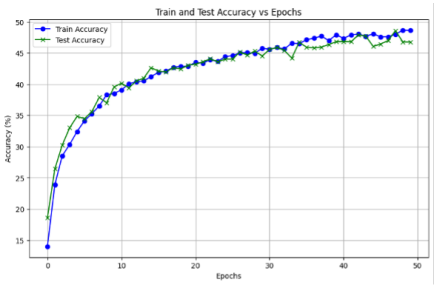


(b) Test Accuracy against ϵ-DP

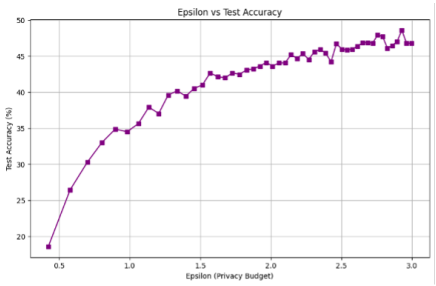


(c) Test and Train loss against epochs

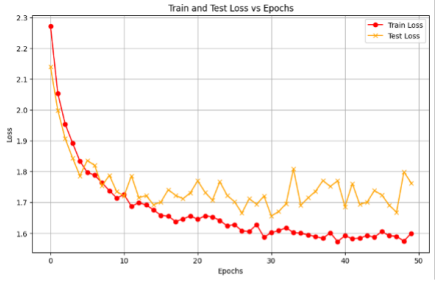
Figure 7: Resnet with DP after the addition of Parameter addition: Batch size: 512



(a) Test and Train Accuracy against epochs

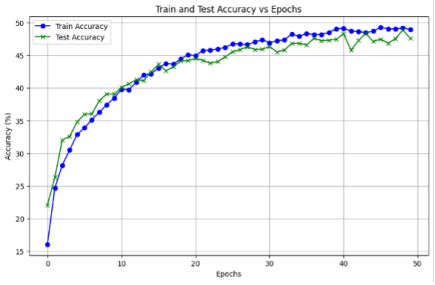


(b) Test Accuracy against ϵ-DP

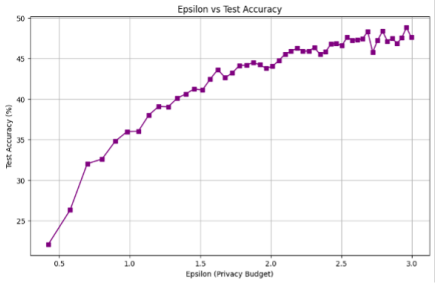


(c) Test and Train loss against epochs

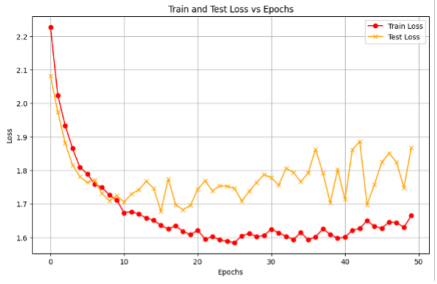
Figure 8: Resnet with DP after the addition of Parameter addition: Batch size: 1024



(a) Test and Train Accuracy against epochs

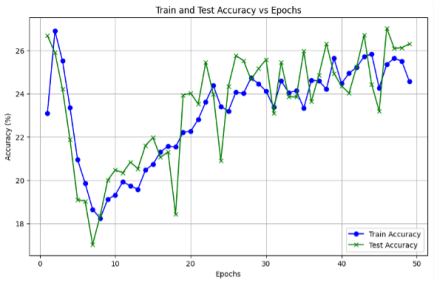


(b) Test Accuracy against ϵ-DP

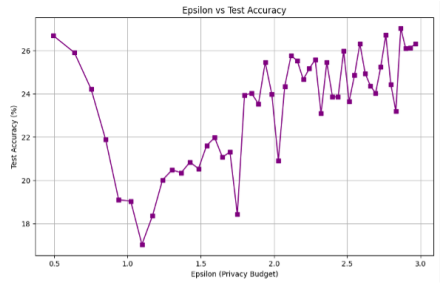


(c) Test and Train loss against epochs

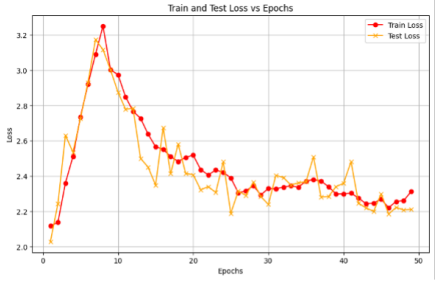
Figure 9: Resnet with DP after the addition of Grouped Gradient clipping.



(a) Test and Train Accuracy against epochs

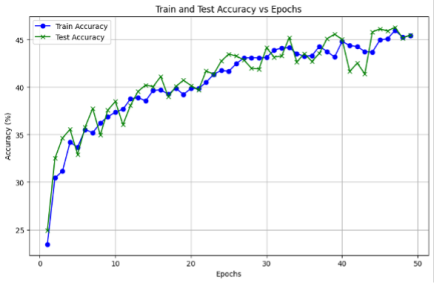


(b) Test Accuracy against ϵ-DP

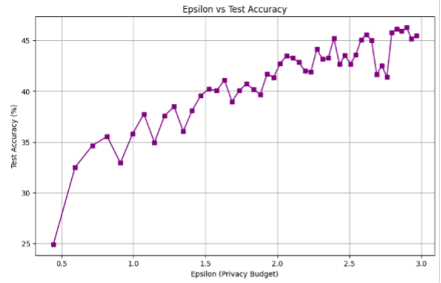


(c) Test and Train loss against epochs

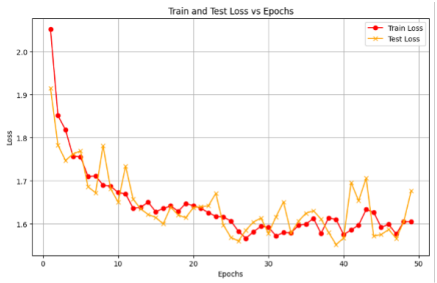
Figure 10: WRN-16-4 with DP - Baseline (Without Batch Normalization)



(a) Test and Train Accuracy against epochs

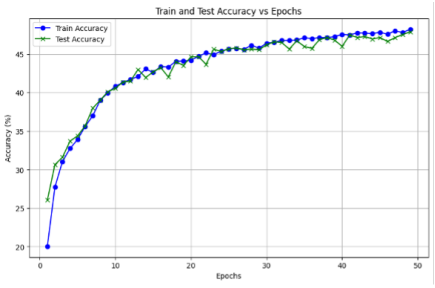


(b) Test Accuracy against ϵ-DP

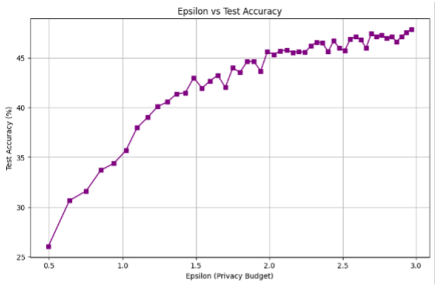


(c) Test and Train loss against epochs

Figure 11: WRN-16-4 with DP after the addition of Group Normalization



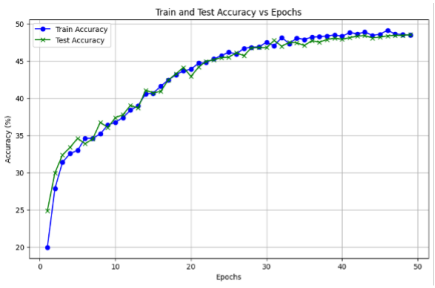
(a) Test and Train Accuracy against epochs



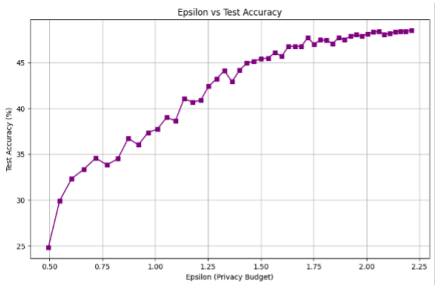
(b) Test Accuracy against ϵ-DP

(c) Test and Train loss against epochs

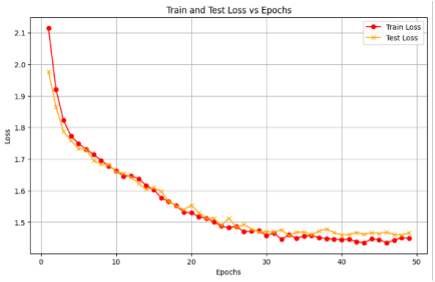
Figure 12: WRN-16-4 with DP after the addition of Weight Standardization, Parameter Averaging, Grouped gradient clipping, Learning rate scheduler



(a) Test and Train Accuracy against epochs



(b) Test Accuracy against ϵ-DP



(c) Test and Train loss against epochs

Figure 13: WRN-16-4 with DP after the addition of Dynamic Noise Multiplier

6 Learning

The training process combines various techniques and hyperparameters, each influencing the training and testing accuracies in different ways. Gradient clipping with a maximum norm of 1.2 ensures that no single gradient dominates the update process, stabilizing training and improving test accuracy by preventing large updates that could lead to overfitting for DP-RMSProp. However, increasing its value to 2.0 did not show any significant improvement in the accuracy for DP-RMSProp and DP-AdaGrad. Grad Norm of 1.5 is the highest value for DP-AdaGrad to increase the accuracy.

Lowering the learning rate value has a significant change for DP-RMSProp and DP-AdaGrad due to their inherent property of adaptive learning rates. A cosine annealing scheduler adjusts the learning rate from 8×10^{-4} to 9×10^{-5} over the course of training, allowing for faster initial convergence and finer adjustments later, which improves test accuracy.

Adding noise to gradients, controlled by a noise multiplier of 1.1, provides differential privacy but reduces both training and test accuracies due to the randomness introduced. It was observed that changing the value of the noise multiplier (i.e., between 0.8 and 1.8) did not have a major improvement in the accuracy. Grouped Gradient Clipping, which divides gradients into 40 groups before clipping, ensures better control over individual gradient components and improves generalization by preserving gradient information better than traditional clipping. The accuracy also changes when the groups are decreased to 4, while there is no change in accuracy when groups are increased to 50. The noise multiplier generated during training was 2.8.

Group Normalization, weight standardization, and Grouped Gradient Clipping increased the accuracy by more than 10%. Group Normalization (GN) further stabilizes training, particularly with smaller batch sizes, by normalizing activations within predefined groups, leading to consistent test accuracy improvements. Weight standardization also plays an important role in model architectures. The accuracy of EfficientNet is significantly lower compared to ResNet-20 and WRN-16-4 due to the absence of weight standardization.

Increasing the number of epochs from 50 to 100 did not change the value of accuracy by a large margin (less than 1%). The use of large logical batches (4096), split into smaller physical batches (128) for memory efficiency, contributes to smoother gradient updates, enhancing training stability and test performance. SGD failed to execute when batch sizes were set to high values (4096).

The Exponential Moving Average (EMA) technique smoothens model updates by averaging parameters across epochs, significantly reducing noise in test accuracy metrics and resulting in better generalization. Without the addition of EMA and per-layer noise addition, the accuracy decreased by 2% and loss increased by 0.5%.

The momentum parameter (momentum $= 0.9$) has an effect on the increase in accuracy for DP-RMSProp, but DP-AdaGrad showed no change in accuracy with the momentum parameter. Model architectures and the SiLU activation function are crucial to maintaining the balance between accuracy and privacy. The final epsilon value generated is 3.0 after 50 epochs.Future work could focus on refining gradient clipping strategies, such as adaptive grouping or layer-specific clipping bounds, to enhance both privacy guarantees and model accuracy.

Incorporating Differential Privacy into deep learning models is challenging and requires adjustments like removing Batch Normalization and adding Group Normalization to stabilize training while adhering to privacy constraints. Incremental modifications such as weight standardization, parameter averaging, grouped gradient clipping, and dynamic noise multipliers significantly improve performance, highlighting the importance of fine-tuning these methods for privacy-preserving learning. Larger batch sizes generally enhance accuracy, but they introduce computational challenges, underscoring the need for a balance between performance gains and resource efficiency. Dynamic noise adjustment based on training loss has potential for optimizing the privacy-utility trade-off by adapting noise levels to the model's learning phase. Future work can explore advanced regularization techniques, alternate model architectures, and adaptive optimization strategies to further improve accuracy while maintaining privacy guarantees.

7 References

1. Abadi, Martin, 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, 308–18. CCS

2. Yingxue Zhou, Xiangyi Chen, Mingyi Hong, Zhiwei Steven Wu, and Arindam Banerjee. *Private Stochastic Non-Convex Optimization: Adaptive Algorithms and Tighter Generalization Bounds.* Proceedings of the 38th International Conference on Machine Learning (ICML), 2021.
3. Canan Tastimur, Songul Kasap, and Erhan Akin. *Knitting Machinery Spare Classification using Deep Learning with Differential Privacy.*
4. Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. *Micro-Batch Training with Batch-Channel Normalization and Weight Standardization.*
5. Haolin Liu, Chenyu Li, Bochao Liu, Pengju Wang, Shiming Ge, and Weiping Wang. *Differentially Private Learning with Grouped Gradient Clipping.*

8 APPENDIX

All the code files are available in Github(3 hyphens before HS below): https://github.com/HARIPRIYA02/DPOptimizers—HS