CSCI 8960: Privacy-Preserving Data Analysis
Interim & Final Report

Nishchay Kumar Radhakrishna
Department of Computer Science
University of Georgia
Athens, GA 30605
`nishchay.kumarr@uga.edu`

Karishma Hegde
Department of Computer Science
University of Georgia
Athens, GA 30605
`karishma.hegde@uga.edu`

Abstract

This project aims to develop a highly efficient, scalable, and differentially private optimizer. The main objective is to compete with the current state-of-the-art accuracy of 70.7% while maintaining $(\epsilon = 3.0, \delta = 10^{-5})$ differential privacy guarantees. The project focuses on optimizing privacy-preserving techniques such as Differentially Private Stochastic Gradient Descent (DP-SGD) using Adaptive Gradient Clipping, and Weight Standardization to improve both accuracy and computational performance. With a focus on reproducibility, the outcomes are compared to privacy and accuracy requirements. Contributions to the field will be included in the final report, which will show enhanced model performance while maintaining differential privacy.

1 Introduction

As organizations increasingly employ machine learning models, ensuring the privacy of individuals' data used to train these models has become essential. Differential privacy (DP) offers a robust solution to this challenge that ensures an adversary cannot predict data or the presence of an individual in training data from released models. The central goal of this project is to develop a differentially private optimizer that balances accuracy and privacy, addressing the limitations of current approaches.

The project focuses on developing models leveraging the DP-SGD (Differentially Private Stochastic Gradient Descent) method through Opacus. As DP-SGD presents trade-offs between accuracy and privacy, this work seeks to refine DP-SGD by introducing key optimizations, including Adaptive Gradient Clipping and Weight Standardization. The project uses ResNet-20 and WRN-16-4 (Wide Residual Networks) architectures for image classification on the CIFAR-10 dataset. Both models have been modified to incorporate the proposed privacy-preserving techniques, with a focus on achieving high accuracy while maintaining privacy budgets.

This report details the models used, the modifications made, and the results achieved in the initial phases of the project. Additionally, it outlines the next steps for further optimizing the system, including the implementation of data augmentation techniques compatible with differential privacy.

2 Models

2.1 ResNet-20: The Initial Model

The concept of Residual Learning was first presented in the 2015 landmark paper by K. He et al.[1] to address the vanishing gradient problem that occurs when very deep networks perform worse than their shallower counterparts.

The main concept is to reframe deep network layers as learning residual functions with respect to the layer inputs instead than learning unreferenced functions directly. This is accomplished through "shortcuts" between each layer, in which the input of a block is kept and added to the transformation's output inside the block. As the network develops deeper, this aids in the retention of crucial information.

ResNet-20 is a variation of the Residual Network (ResNet) architecture that belongs to a family of models including more complex models like ResNet-50 and ResNet-101, and is especially made for jobs like image classification.

Architecture

The implemented ResNet-20 architecture consists of an initial convolutional layer followed by three layers of residual blocks, each containing three blocks. Each residual block comprises two convolutional layers, interleaved with Group Normalization and ReLU activation functions, to facilitate effective learning through skip connections. The first layer maintains 16 channels, while the subsequent layers increase to 32 and 64 channels, with downsampling applied in the second and third layers using a stride of 2. This downsampling reduces the spatial dimensions from 32x32 in the first layer to 16x16 in the second layer and finally to 8x8 in the third layer, enabling the model to focus on increasingly abstract representations of the input data while managing computational efficiency. The model concludes with an adaptive average pooling layer, which feeds into a fully connected layer for classification, outputting predictions for a specified number of classes. Figure 1 lays out the sequence of the layers, along with the filter and output channel dimensions.

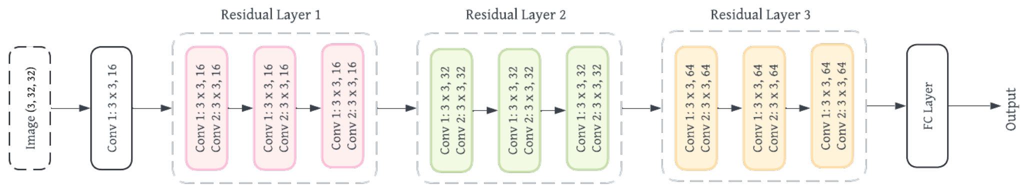

Figure 1: ResNet-20 Layer Architecture

Reasons for choosing ResNet-20

Our primary intuition to begin with ResNet-20 as a baseline model was due to the popularity of ResNets in image classification tasks[1]. The 20 layer model is most feasible within our resource limits, while still leveraging the advantages of ResNet architectures. Following ResNet-20, we attempted to implement ResNet-34 as a more complex baseline. However, due to hardware constraints, specifically the available GPU memory, the model encountered a CUDA `OutOfMemoryError`, pushing us to proceed with ResNet-20.

A. Kurakin et al. in their work[2] suggest that the size of the model is not proportional to the accuracy, in the context of differential privacy. Further, DP-SGD requires adding a predetermined amount of noise, σ, to each parameter in order to achieve the appropriate level of privacy, ϵ. Therefore, using DP-SGD on big models results in a greater parameter shift than on smaller models, which may have a negative impact on accuracy. This reassured us that working with a smaller model will not restrict the project from achieving good results.

Algorithm

The implemented ResNet-20 model is written by taking reference from the algorithm by Yerlan[3]. Algorithm 1 gives the baseline model, based on the building blocks provided by the PyTorch `torch.nn` module, which offers a flexible and efficient framework for constructing deep learning models.

The Adaptive Gradient Clipping method implemented in Algorithm 2 is derived from G. Andrew et al.'s work[4]. The clipping norm is dynamically adjusted to track a quantile (e.g., the median) of the

Algorithm 1 ResNet-20 Forward Pass

```
Input: Image $x$ (e.g., $32 \times 32$ RGB image)
Apply initial convolution: $x = \text{Conv2D}(x, \text{filters} = 16, \text{kernel size} = 3, \text{stride} = 1, \text{padding} = 1)$
Stage 1: Apply 3 residual blocks (16 filters each)
for each residual block in Stage 1 do
    Apply WSConv2D($x$, 16 filters, stride 1)
    Apply GroupNorm and ReLU activation
    Apply WSConv2D($x$, 16 filters, stride 1)
    Apply GroupNorm
    Add skip connection
    Apply ReLU activation
end for
Stage 2: Apply 3 residual blocks (32 filters)
First block: downsampling with stride 2
for each residual block in Stage 2 do
    Apply WSConv2D($x$, 32 filters)
    Add skip connection if downsampling
end for
Stage 3: Apply 3 residual blocks (64 filters)
Apply WSConv2D, GroupNorm, ReLU, skip connections
Apply Average Pooling
Fully Connected Layer
=0
```

gradient norms over time. The clipping threshold is updated using a geometric update rule, which adjusts the clipping norm in an exponential manner to ensure fast convergence to the desired quantile. We used the same target quantile value as the paper uses, as it strikes a good balance and helps ensure that the majority of gradients are not excessively clipped or under-clipped. The update rule for the clipping norm C_t is:

$$C_{t+1} = C_t \cdot \exp\left(-\eta_C \left(\bar{b}_t - \gamma\right)\right)$$

Where,

- η_C is the learning rate for the clipping norm.
- $\bar{b}_t$ is the fraction of updates that are clipped at step t.
- γ is the target quantile (e.g., 0.5 for the median).

Algorithm 2 Adaptive Gradient Clipping

```
Input: Model parameters $W$, gradient norms $grad\_norms$, initial clipping norm $C_0$, target quantile $\gamma$, learning rate $\eta_C$
Initialize clipping norm $C \leftarrow C_0$
for each training step $t$ do
    Compute the current gradient norms for all parameters $grad\_norms$
    Calculate the fraction of gradients below the current clipping norm:
    $clipped\_fraction \leftarrow \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(grad\_norms_i \leq C)$
    Update the clipping norm using the geometric update rule:
    $C \leftarrow C \cdot \exp(-\eta_C \cdot (clipped\_fraction - \gamma))$
    for each parameter $W_i$ do
        Clip the gradient:
        $grad(W_i) \leftarrow grad(W_i) \cdot \min\left(1, \frac{C}{\|grad(W_i)\|}\right)$
    end for
    Update model parameters using the clipped gradients
end for
return Updated model parameters $W$ =0
```

The Weight Standardization procedure in Algorithm 3 involves normalizing the weights of each convolutional layer by subtracting the mean and dividing by the standard deviation across the filter dimensions before the convolution operation. This ensures that the weights have zero mean and unit variance, improving gradient flow and training stability.

Algorithm 3 Weight Standardization

```
Input: Weights $W$ of the convolutional layer
for each convolutional filter $W_i$ do
    Compute the mean of the weights:
    $\mu_W \leftarrow \frac{1}{n} \sum_{i=1}^{n} W_i$
    Compute the standard deviation of the weights:
    $\sigma_W \leftarrow \sqrt{\frac{1}{n} \sum_{i=1}^{n} (W_i - \mu_W)^2}$
    Standardize the weights:
    $W_{\text{standardized}} \leftarrow \frac{W - \mu_W}{\sigma_W + \epsilon}$
end for
Use the standardized weights for the convolution operation
return Convolutional output with standardized weights =0
```

Modifications

The modifications made to the vanilla ResNet model were aimed towards optimizing the utility of the system. We came across A. Yousefpour et al.'s work[4] and modified the gradient calculation method. As per their work (on federated learning systems), the **Adaptive Gradient Clipping** method applied to the median update norm performs effectively without the need to adjust any clipping hyperparameters. The approach clips updates depending on a certain quantile (such as the median) of the gradient norm distribution rather than clipping to a preset value. As a result, the clipping becomes adaptable to the inherent diversity in the data over the course of several training cycles. In our initial training, after the application of Adaptive Gradient Clipping, the accuracy increased from 41.32% to 50.36%. On repeating the training process, we found that the accuracy stabilized around a value of 46.94%.

Next, we implemented **Weight Standardization** in the model, which was also implemented by several authors including S. De et al.'s work[5]. Weight Standardization is a normalization technique used to stabilize the loss landscape by normalizing the weights of each convolutional layer before applying the convolution operation[6]. It was first introduced as a solution to memory constraints encountered during Batch Normalization [7]. Implementing this modification increased the test accuracy to 54.98%.

Our final step towards optimization for this phase of the project was to increase the **Logical Batch Size** along with the **Learning Rate**. The batch size was increased from 512 to **2048** and the Learning rate from 0.001 to **0.5**. We also increased the number of epochs from 15 to **25**. This modification pushed the test accuracy to **60.97%**, the best accuracy obtained so far.

The benefits of increasing the batch size was highlighted in the work by R. Anil et al.[8]. The paper describes how training with DP-SGD enhances the gradient signal-to-noise ratio (SNR) as batch size increases. In particular, stronger gradient aggregation made possible by larger batch sizes reduces the effect of the additional noise required for differential privacy. Using a fixed batch size can slow down convergence by allowing noise to become more dominant as training goes on and gradients get smaller. The authors suggest an increased batch size plan to combat this, which increases training efficiency without sacrificing accuracy.

Related Work

A. Kurakin et al.[2] implemented ResNet-18 with DP-SGD.

2.2 WRN-16-4

Wide Residual Networks (Wide-ResNet/WRN) are a variation of ResNets in which the residual networks' width is increased and their depth is decreased with the help of Wide residual blocks. The idea was first proposed by S. Zagoruyko et al.[9] to combat network training issues to increase accuracy. WRNs have fewer layers but more parameters per layer since they trade off extreme depth for greater width. Compared to highly deep ResNets, this is computationally more efficient and frequently results in better performance.

Architecture

The implemented Wide Residual Network (WRN-16-4) architecture consists of an initial convolutional layer, followed by three layers of residual blocks, each containing $n = 2$ blocks. Each residual block comprises two convolutional layers, interleaved with Group Normalization and ReLU activation functions, facilitating effective learning through skip connections. The widening factor of 4 increases the number of channels in each block, significantly boosting the model's capacity. Specifically, the first layer starts with 64 channels (16 channels widened by a factor of 4), the second layer increases to 128 channels (32 channels widened by 4), and the third layer has 256 channels (64 channels widened by 4). Downsampling is applied in the second and third layers using a stride of 2, reducing the spatial dimensions from 32×32 to 16×16 and finally to 8×8. This allows the network to efficiently capture more abstract representations of the input data. The model concludes with a global average pooling layer and a fully connected layer for classification. The use of weight standardization and group normalization helps stabilize training and improves generalization across varying batch sizes.

Reasons for choosing WRN-16-4

As we were capped out at ResNet-20 with our current computational resources, we looked for models that would work towards increasing the accuracy without requiring substantial increase in processing. The paper [5] also uses a WRN-16-4 as a the baseline model as it has shown promising results for the CIFAR-10 dataset in both private and non-private settings.

Algorithm

Algorithm 4 outlines the structure of the WRN-16-4 model implemented.

Algorithm 4 Wide-Residual-Network (WRN)-16-4

```
Input: Image $x$ (e.g., $32 \times 32$ RGB image)
Apply initial WSConv2d with 16 filters, $3 \times 3$ kernel, stride 1, and padding 1
Apply GroupNorm and ReLU activation
Stage 1: Apply $n$ residual blocks with 16 and 64 filters
for each residual block in Stage 1 do
    Apply WSConv2d (stride = 1)
    Apply GroupNorm and ReLU activation
    Apply second WSConv2d
    Add skip connection (identity mapping) and apply ReLU activation
end for
Stage 2: Apply $n$ residual blocks with 32 and 128 filters
for each residual block in Stage 2 do
    Apply WSConv2d with stride = 2 (downsampling)
    Apply GroupNorm and ReLU activation
    Apply second WSConv2d
    Add skip connection (WSConv2d with stride = 2) and apply ReLU activation
end for
Stage 3: Apply $n$ residual blocks with 64 and 256 filters
for each residual block in Stage 3 do
    Apply WSConv2d with stride = 2 (downsampling)
    Apply GroupNorm and ReLU activation
    Apply second WSConv2d
    Add skip connection (WSConv2d with stride = 2) and apply ReLU activation
end for
Apply average pooling with kernel size 8
Flatten the pooled output
Apply fully connected layer to produce output (class probabilities)
=0
```

Modifications

Tthe WRN-16-4 is not directly available as a pre-built model in PyTorch's standard torchvision models library. PyTorch offers WRN-50-2 and WRN-101-2 but were too large for processing. Hence, we implemented WRN-16-4 by modifying the existing ResNet architecture. We brought over the Group normalization, adaptive gradient clipping and weight standardization implementations from the ResNet-20 model.

3 Privacy Proof

To provide a formal privacy proof for the differentially private training setup in both ResNet-20 and WRN-16-4 implementations, we use the core properties of DP as they apply to machine learning. Specifically, it leverages DP-SGD, which builds upon these concepts:

If the output distributions for any contiguous datasets D and D differ by one data point, then a mechanism M is (ϵ, δ) - differentially private, i.e.

$$\Pr[M(D) \in S] \leq e^{\epsilon} \Pr[M(D') \in S] + \delta[10]$$

Here, the mechanism M is the DP-SGD algorithm, with $\epsilon = 3.0$ and $\delta = 10^{-5}$.

The adptive gradient clipping method adds a Gaussian noise with standard deviation proportional to σ to the clipped gradients. This mechanism ensures privacy by obfuscating individual data points. The equation for nosiy gradient for weight update gradients $\tilde{g}$ is given as:

$$\tilde{g} = \frac{1}{n} \sum_{i=1}^{n} g'_i + \mathcal{N}(0, \sigma^2 C^2 I)$$

where,

n - batch size.

g'_i - the clipped gradient for the i-th data point in the batch.

Therefore,

$\frac{1}{n} \sum_{i=1}^{n} g'_i$ is the average clipped gradient over the entire batch.

σ^2 is the Gaussian noise added to the averaged gradient.

C^2 is the Clipping norm for the gradients.

I is the identity matrix.

Therefore,

$\mathcal{N}(0, \sigma^2 C^2 I)$ is the random noise sampled from a Gaussian distribution with mean 0 and variance $\sigma^2 C^2$.

Opacus's PrivacyEngine tracks the privacy loss over multiple steps using the moments accountant method, which accumulates the privacy loss efficiently over the course of training. The final privacy budget ϵ is computed as:

$$\epsilon = f(\sigma, T, B, N, \delta)$$

where, σ is the noise multiplier.

is the number of training iterations (epochs).

is the batch size.

is the dataset size.

δ is the privacy failure probability.

Related Work

S. De et al. [5] implement the WRN-16-4 model with DP-SGD as their baseline model.

4 Preliminary Results and Training Details

4.1 Optimizer Details

Our initial model used RMSProp as the optimizer. RMSProp adapts the learning rate during training, so it typically works well with smaller learning rates. However, to accommodate the larger batch size and learning rate, we decided to replace the optimizer with DP-SGD, which requires more manual tuning, and larger learning rates are common.

Additionally, the Differentially Private PyTorch library Opacus that is being used favours gradient calculation by the Stochastic Gradient Descent (SGD) method. In each iteration, after computing the gradient in the back propagation, some noise is added to the value. This masks the original data from the model and avoids memorization of examples. This is especially important for outliers, that are at a higher chance of being leaked, as they have a greater influence on gradient calculation.

Taking into account the computation, Opacus handles per-sample gradient by implementing an effective technique. This involves the algorithm re-uses the already computed gradients and further processes them to obtain per-sample gradients In a traditional ML model, the per-sample gradient would require O(mnp2) operations, whereas Opacus's technique does this with O(mnp) operations[11]. It involves computing matrices Z (pre-activation) and H (activation) for the minibatch, the algorithm computes the per-sample gradient norms without to re-running the backpropagation multiple times.

The hyperparameter values in relation to the privacy for our baseline mechanism with ResNet-20 are as follows:

- **Privacy Budget ϵ:** 3.0
- **Privacy Loss Probability δ:** 1×10^{-5}
- **Batch Size :**
  - **Logical Batch Size:** 512
  - **Physical Batch Size:** 128
- **Learning Rate:** 1×10^{-3}

As part of our effort towards improving the accuracy, we retained the (ϵ, δ) values, but increased the logical batch size and the learning rate in particular to support the DP-SGD optimizer:

- **Logical Batch Size:** 2048
- **Learning Rate:** 0.1

The same updated values were carried on for implementing the WRN-16-4 model. The models were trained on Google Colab leveraging the Tesla T4 GPU.

Before training our models on the CIFAR-10 dataset, we applied normalization to the input images using the mean and standard deviation values[1] precomputed for the dataset. Normalization ensures that the data is centered and scaled to a consistent range, which improves model training stability and efficiency.

4.2 Differential Privacy Details

The graph in Figure 2 shows the privacy budget ϵ accounting over **25 epochs**. The noise scale calculated by Opacus's `make_private_with_epsilon` method is **1.61**.

5 Discussion on Results

Both models were able to successfully run with differential privacy guarantees using and $\epsilon = 3.0$ $\delta = 10^5$ as their ultimate privacy budgets. The ResNet-20 model stands at a best test accuracy of **60.97%**. The WRN-16-4 model achieved a best test accuracy of **55.61%**.

The Privacy Budget Graph in Figure 2 demonstrates that privacy is consumed linearly over time.

[1]Mean = (0.4914, 0.4822, 0.4465) and Standard Deviation = (0.2023, 0.1994, 0.2010) where the 3 values correspond to the color channels RGB

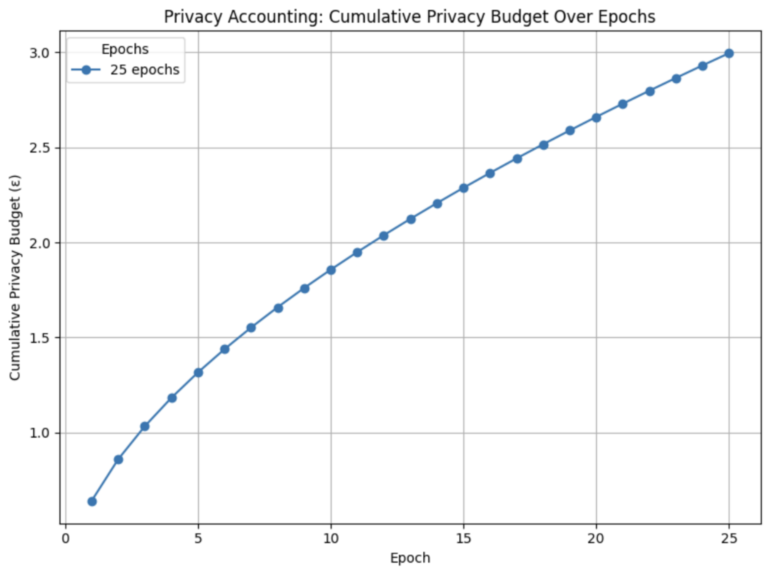

Figure 2: Privacy Accounting

- X-axis (Epochs): Represents the number of epochs, where each epoch is a complete pass through the training dataset.
- Y-axis (Cumulative Privacy Budget - ϵ): Shows the cumulative privacy budget (ϵ), which increases as more training epochs are completed.
- Data Points: Each point represents the cumulative at the end of a particular epoch.

Each epoch contributes an equal amount of privacy loss, and by the end of the 25th epoch, the privacy budget is completely used up at $\epsilon = 3$. This steady increase suggests that our training process is efficiently managing the privacy budget, ensuring that the model does not overspend or underspend on privacy in any particular phase of the training.

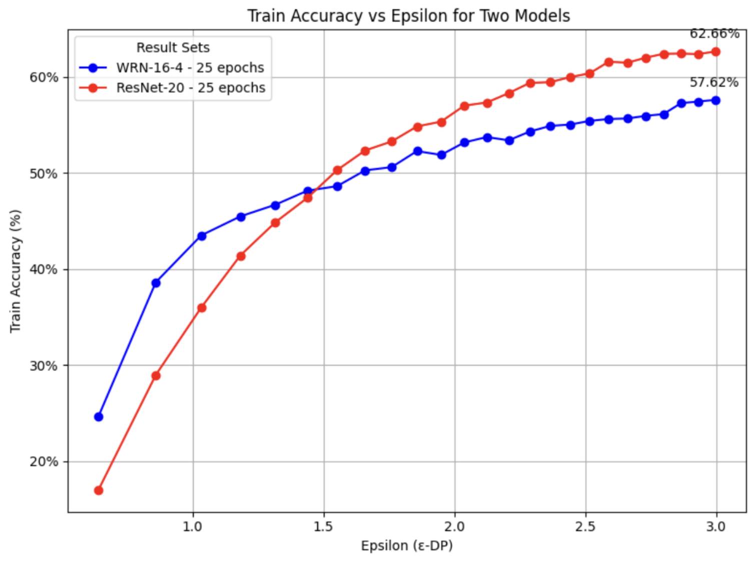

Figure 3: Train Accuracy vs Privacy Budget for ResNet-20 and WRN-16-4

The graph shown in Figure 3 compares training accuracy against the privacy budget (ϵ) for two models, ResNet-20 and WRN-16-4.

- X-axis (ϵ): This represents the differential privacy budget
- Y-axis (Train Accuracy %): This shows the training accuracy of the models as a percentage.

ResNet-20 consistently outperforms WRN-16-4 across all ϵ values. At $\epsilon = 3$, the final accuracy for ResNet-20 reaches approximately 62.66%, while WRN-16-4 reaches 57.62%. This suggests that ResNet-20 is able to extract more useful patterns from the data, leading to better training accuracy while consuming the same amount of privacy budget. WRN-16-4 starts with a higher initial accuracy than ResNet-20 at lower ϵ values (below 1.5). This might suggest that WRN-16-4 is slightly more robust to noise in the early stages when privacy constraints are stronger, but as privacy is relaxed (with increasing ϵ), ResNet-20 takes the lead and ends up performing better.

The ResNet-20 model appears to be more privacy-efficient compared to WRN-16-4. Given the same privacy budget ($\epsilon = 3$), ResNet-20 achieves a higher accuracy than WRN-16-4. WRN-16-4 could be a more appropriate model in situations where stronger privacy constraints are needed (i.e., lower ϵ), since it shows better accuracy at the early stages ($\epsilon < 1.5$). However, it does not scale as well as ResNet-20 when the privacy budget is increased.

Table 1 summarizes the average accuracy over multiple executions against the standard deviation values.

Model	**Privacy Parameters**	**Average Accuracy ± Standard Deviation**
ResNet-20	$\epsilon = 3.0, \delta = 10^{-5}$, 25 epochs	60.82 ± 0.65
WRN-16-4	$\epsilon = 3.0, \delta = 10^{-5}$, 25 epochs	54.59 ± 1.17

Table 1: Results of models with privacy parameters and accuracy

6 Summary and Future Steps

In this project, we applied Differential Privacy using Opacus to existing machine learning models. We witnessed how different parameters affect the privacy utility trade-offs. We were also able to observe how certain modifications like Adaptive Gradient Clipping and Weight Standardization can optimize the accuracy, without compromising the privacy.

In the next phase of the project, we aim to improve the differential privacy learning of the system by implementing data augmentation. As traditional data augmentation techniques are not compatible with DP mechanisms, we plan on referring to the work by W. Bao et al.[12] on multi-sample data augmentation technique and other similar techniques. For the WRN model, we shall work towards adjusting the hyperparameters to improve the accuracy.

We also aim to set up a more reliable computing resource by leveraging Apple's MPS (Metal Performance Shaders) for PyTorch[13] on our local systems, as Google Colab's free tier T4 GPU, although fast and powerful, offers limited usage limits.

7 Final Report

In the subsequent phases of the project, we explored new approaches to improve model performance under differential privacy constraints, focusing on areas such as fine-tuning the gradient clipping technique and addressing challenges related to accuracy saturation. These efforts aimed to strike a balance between achieving robust privacy guarantees and maximizing model utility.

7.1 New Approaches

In our implementation, we integrated the Dynamic Differential Privacy-Preserving Stochastic Gradient Descent (Dynamic DP-SGD) algorithm from J. Du et al.'s work[14] to enhance the privacy-utility trade-off in differentially private training. By dynamically adjusting the gradient clipping threshold and noise multiplier, the paper claimed to stabilize gradient updates and help achieve improved performance compared to static DP-SGD approaches. For our model, it did not contribute to increasing the accuracy. Hence we did not move forward with this approach as the complexity added did not justify the impact it made on the accuracy.

During the training process, we observed the accuracy stagnating over multiple epochs, which slowed down the increase in accuracy. To address the challenge of accuracy plateauing, we implemented a **dynamic learning rate adjustment** using the `ReduceLROnPlateau` scheduler. This approach monitors validation accuracy and reduces the learning rate if no improvement is observed for a predefined number of epochs, allowing the model to make finer weight updates and potentially escape local minima[15]. With `patience` set to 2 epochs and a reduction `factor` of 0.5, this method improved convergence stability in the differential privacy setting, though the gains were modest (increased the accuracy by 1%) motivating further exploration of additional optimization techniques.

In our efforts to further optimize the gradient clipping process, we experimented with the approach described in J. He. et al.[16]'s work. It discusses the implementation of **per-layer clipping** in differentially private optimization. The authors demonstrate that clipping the gradient of each neural network layer separately allows for efficient backpropagation, resulting in private learning that is as memory-efficient and nearly as fast per training update as non-private learning for many workflows. It refined the clipping process, but the gain in terms of accuracy and performance were not too significant.

7.2 Training Details

Optimizer Details

We continued to use DP-SGD as the optimizer for this project, as it is the foundational optimization algorithm implemented in the Opacus library. This made it straightforward to integrate the enhancements we explored.

The batch size was further pushed from 2048 to 4096. Accompanied by the increased epochs, per-layer clipping and `ReduceLROnPlateau` scheduler, this setup resulted in giving the best accuracy of **62.39%**.

The hyperparameter values for the setup obtaining best accuracy are as follows:

- **Privacy Budget ϵ:** 3.0
- **Privacy Loss Probability δ:** 1×10^{-5}
- **Batch Size :**
  - **Logical Batch Size:** 4096
  - **Physical Batch Size:** 128
- **Learning Rate:** 1×10^{-3}

Differential Privacy Details

As directly configuring the noise multiplier using `make_private` did not improve the performance for our model (experimented in BA-Report II), we continued using `make_private_with_epsilon` allowing Opacus' built-in mechanism for calculating the noise multiplier[17]. In our final mechanism, this value was around **2.88**. Opacus calculates this based on ϵ, δ, batch size, dataset size, number of epochs, and sampling rate. Using the Rényi Differential Privacy (RDP) framework, it tracks the cumulative privacy loss and employs numerical methods to solve for the noise multiplier, ensuring that the desired privacy guarantees are met while maintaining model utility.

The training code was executed for 25, 35, and 50 epochs to evaluate the impact of training duration on model performance. Our results indicated that the model showed most of its accuracy improvements within 25–35 epochs, after which the accuracy plateaued. The increase after this range was very slow, taking around 5-10 epochs for an increase in 1%. Hence, we determined that ideally, **35** epochs was sufficient to get the best performance. For the final results, we executed 50 epochs to provide the model sufficient opportunity to maximize its accuracy.

7.3 Experimental Results and Discussions

In this section, we present the experimental results obtained during the training of the ResNet-20 model under differential privacy constraints. We evaluate the model's performance in terms of accuracy and privacy budget across epochs and explore the effects of techniques like dynamic learning rate adjustments and clipping technique modifications. Additionally, we analyze trends in the training process to identify the trade-offs between privacy guarantees and model utility.

Privacy Budget (ϵ) and Accuracy

Graph 4 illustrates the relationship between the train accuracy and the privacy budget (ϵ) over 50 epochs. As ϵ increases, the model's accuracy improves significantly, with diminishing returns beyond $\epsilon \approx 2.0$. This trend highlights the fundamental trade-off between privacy and utility in differentially private machine learning. The graph underscores the importance of carefully selecting ϵ to balance model utility with privacy guarantees, especially for scenarios where utility cannot be compromised significantly.

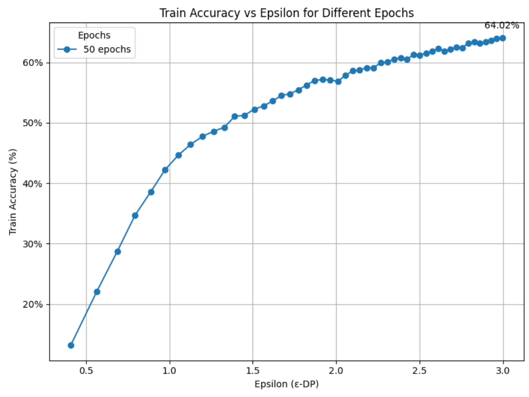

Figure 4: Privacy Budget vs. Train Accuracy

Next, we analyzed the loss trends over the different epochs with respect to the ϵ in graph 5. As ϵ increases, the train loss decreases, particularly in the range of ϵ = 0.5 to 1.5, indicating that reducing the noise required for stronger privacy enables the model to better minimize the loss. This behavior reinforces the trade-off between privacy and utility in differentially private training: higher noise (smaller ϵ) introduces greater instability in gradient updates, leading to higher loss.

In order to understand how the learning rate scheduler (`ReduceLROnPlateau`) impacted the optimization, we plotted graph 6. The significant drops in the learning rate are observed around epochs 20 and 30 align with the plateau phases of the accuracy curve. These reductions help the model stabilize in convergence while avoiding large updates that could disturb the optimization process. By the end of training, the learning rate approaches zero, reflecting the diminishing gradient updates as the model reaches its final state.

Finally, we trained the model over two other ϵ values – 5 and 8 to observe its behavior and its impact on test accuracy. Graph 7 shows how the accuracy improves noticeably from ϵ = 3 to ϵ = 5. However, the improvement from ϵ = 5 to ϵ = 8 is minimal, indicating that the model has reached a saturation point where further relaxation of privacy constraints provides diminishing returns.

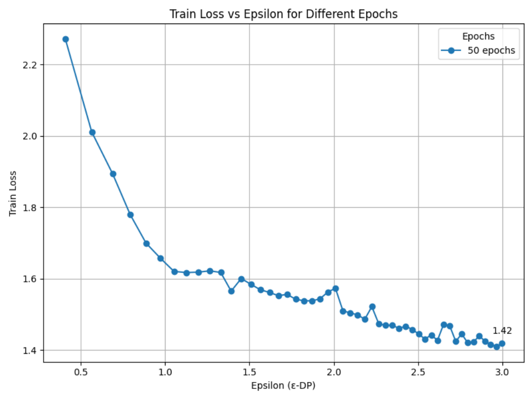

Figure 5: Privacy Budget vs. Train Loss

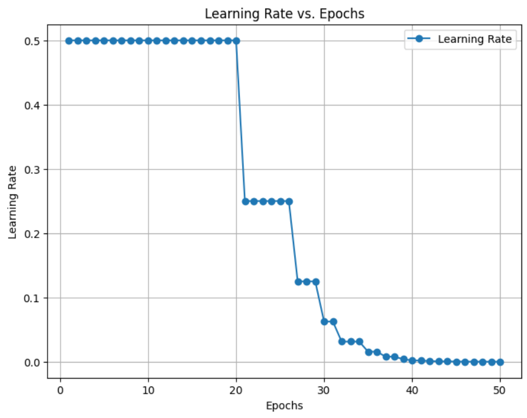

Figure 6: Learning Rate adjusted by ReduceLROnPlateau scheduler over the epochs

To summarize this section, our experiments have demonstrated the balance between privacy and utility in DP training. The interplay of learning rate adjustments and noise reduction helped achieve optimal accuracy without compromising privacy guarantees.

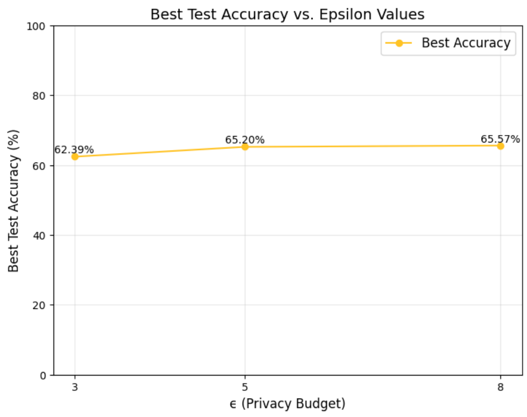

Figure 7: Best Test Accuracy over different Privacy Budgets (ϵ)

7.4 Conclusion

This project has provided valuable insights into the challenges and trade-offs of training machine learning models under differential privacy constraints. By implementing techniques such as dynamic learning rate adjustments, adaptive gradient clipping, and experimentation across varying privacy budgets, we observed the delicate balance between achieving strong privacy guarantees and maintaining model utility.

In future work, we propose exploring the use of pre-trained models and fine-tuning it under differential privacy constraints as done in S. De. et al's work [5]. This approach could significantly enhance performance by leveraging the rich feature representations learned during pre-training while reducing computational costs. The behavior of accuracy saturation warrants further investigation, as understanding it more deeply could help optimize computational efficiency while achieving improved accuracy.

Github

Repository - /karishmahegde/dp-optimizer

References

[1] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385`. URL: `http://arxiv.org/abs/1512.03385`.

[2] Alexey Kurakin et al. "Toward Training at ImageNet Scale with Differential Privacy". In: *CoRR* abs/2201.12328 (2022). arXiv: `2201.12328`. URL: `https://arxiv.org/abs/2201.12328`.

[3] Yerlan Idelbayev. *Proper ResNet Implementation for CIFAR10/CIFAR100 in PyTorch*. `https://github.com/akamaster/pytorch_resnet_cifar10`. Accessed: 2024-10-03.

[4] Galen Andrew et al. *Differentially Private Learning with Adaptive Clipping*. 2022. arXiv: `1905.03871 [cs.LG]`. URL: `https://arxiv.org/abs/1905.03871`.

[5] Soham De et al. *Unlocking High-Accuracy Differentially Private Image Classification through Scale*. 2022. arXiv: `2204.13650 [cs.LG]`. URL: `https://arxiv.org/abs/2204.13650`.

[6] Papers with Code. *Weight Standardization*. `https://paperswithcode.com/method/weight-standardization`. Accessed: 2024-10-15. 2024.

[7] Siyuan Qiao et al. "Micro-Batch Training with Batch-Channel Normalization and Weight Standardization". In: *CoRR* abs/1903.10520 (2019). arXiv: `1903.10520`. URL: `http://arxiv.org/abs/1903.10520`.

[8] Rohan Anil et al. "Large-Scale Differentially Private BERT". In: *CoRR* abs/2108.01624 (2021). arXiv: `2108.01624`. URL: `https://arxiv.org/abs/2108.01624`.

[9] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks". In: *CoRR* abs/1605.07146 (2016). arXiv: `1605.07146`. URL: `http://arxiv.org/abs/1605.07146`.

[10] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Now Publishers Inc, 2014.

[11] Ian Goodfellow. *Efficient Per-Example Gradient Computations*. 2015. arXiv: `1510.01799 [stat.ML]`. URL: `https://arxiv.org/abs/1510.01799`.

[12] Wenxuan Bao et al. *DP-Mix: Mixup-based Data Augmentation for Differentially Private Learning*. 2023. arXiv: `2311.01295 [cs.LG]`. URL: `https://arxiv.org/abs/2311.01295`.

[13] PyTorch. *MPS Backend*. `https://pytorch.org/docs/stable/notes/mps.html`. Accessed: 2024-10-17. 2023.

[14] Jian Du et al. "Dynamic Differential-Privacy Preserving SGD". In: *CoRR* abs/2111.00173 (2021). arXiv: `2111.00173`. URL: `https://arxiv.org/abs/2111.00173`.

[15] PyTorch. *MPS Backend*. `https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html`.

[16] Jiyan He et al. *Exploring the Limits of Differentially Private Deep Learning with Group-wise Clipping*. 2022. arXiv: `2212.01539 [cs.LG]`. URL: `https://arxiv.org/abs/2212.01539`.

[17] Opacus. *Noise Multiplier*. `https://opacus.ai/api/accounting/utils.html`.