

329 Contents

| | | |
|-----|---|-----------|
| 330 | 1 Introduction | 1 |
| 331 | 1.1 Differential Privacy and Lipschitz Networks | 3 |
| 332 | 2 Clipless DP-SGD with l-Lipschitz networks | 4 |
| 333 | 2.1 Backpropagation for bounds | 5 |
| 334 | 2.2 Privacy accounting for Clipless DP-SGD | 5 |
| 335 | 3 From theory to practice | 6 |
| 336 | 3.1 Gradient Norm Preserving networks | 7 |
| 337 | 3.2 Lip-dp library | 8 |
| 338 | 4 Experimental results | 8 |
| 339 | 5 Limitations and future work | 9 |
| 340 | 6 Concluding remarks and broader impact | 9 |
| 341 | A Definitions and Methods | 12 |
| 342 | A.1 Additionnal background | 12 |
| 343 | A.1.1 Lipschitz neural networks background | 12 |
| 344 | A.1.2 Gradient Norm Preserving networks | 12 |
| 345 | A.1.3 Differential Privacy background | 13 |
| 346 | A.2 More about Clipless DP-SGD | 13 |
| 347 | B Lipdp tutorial | 13 |
| 348 | B.1 Prerequisite: building a l -Lipschitz network | 14 |
| 349 | B.2 Getting started | 15 |
| 350 | B.3 Image spaces and input clipping | 16 |
| 351 | B.3.1 Color space representations | 16 |
| 352 | B.3.2 Input clipping | 16 |
| 353 | B.4 Practical implementation of Residual connections | 17 |
| 354 | B.5 Loss-logits gradient clipping | 18 |
| 355 | B.5.1 Possible improvements | 19 |
| 356 | C Computing Sensitivity Bounds | 19 |
| 357 | C.1 Losses bounds | 19 |
| 358 | C.2 Layer bounds | 21 |
| 359 | C.2.1 Dense layers | 21 |
| 360 | C.2.2 Convolutions | 22 |
| 361 | C.2.3 Layer normalizations | 23 |
| 362 | C.2.4 MLP Mixer architecture | 24 |

| | | |
|-----|--|-----------|
| 363 | D Experimental setup | 24 |
| 364 | D.1 Pareto fronts | 24 |
| 365 | D.1.1 Hyperparameters configuration for MNIST | 24 |
| 366 | D.1.2 Hyperparameters configuration for FASHION-MNIST | 25 |
| 367 | D.1.3 Hyperparameters configuration for CIFAR-10 | 25 |
| 368 | D.2 Configuration of speed experiment | 26 |
| 369 | D.3 Drop-in replacement with Lipschitz networks in vanilla DPSGD | 26 |
| 370 | D.4 Extended limitations | 27 |
| 371 | E Proofs of general results | 28 |
| 372 | E.1 Main result | 28 |
| 373 | E.2 Proof of main result | 30 |
| 374 | E.3 Variance of the gradient | 33 |

A Definitions and Methods

A.1 Additionnal background

The purpose of this appendix is to provide additionnal definitions and properties regarding Lipschitz Neural Networks, their possible GNP properties and Differential Privacy.

A.1.1 Lipschitz neural networks background

For simplicity of the exposure, we will focus on feedforward neural networks with densely connected layers: the affine transformation takes the form of a matrix-vector product $h \mapsto Wh$. In section C.2 we tackle the case of convolutions $h \mapsto \Psi * h$ with kernel Ψ .

Definition 4 (Feedforward neural network). *A feedforward neural network of depth T , with input space $\mathcal{X} \subset \mathbb{R}^n$, and with parameter space $\Theta \subset \mathbb{R}^p$, is a parameterized function $f : \Theta \times \mathcal{X} \rightarrow \mathcal{Y}$ defined by the following recursion:*

$$\begin{aligned} h_0(x) &:= x, & z_t(x) &:= W_t h_{t-1}(x) + b_t, \\ h_t(x) &:= \sigma(z_t(x)), & f(\theta, x) &:= z_{T+1}(x). \end{aligned} \quad (8)$$

The set of parameters is denoted as $\theta = (W_t, b_t)_{1 \leq t \leq T+1}$, the output space as $\mathcal{Y} \subset \mathbb{R}^K$ (e.g logits), and the layer-wise activation as $\sigma : \mathbb{R}^n \rightarrow \mathbb{R}^n$.

Definition 5 (Lipschitz constant). *The function $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ is said l -Lipschitz for l_2 norm if for every $x, y \in \mathbb{R}^m$ we have:*

$$\|f(x) - f(y)\|_2 \leq l \|x - y\|_2. \quad (9)$$

Per Rademacher's theorem [10], its gradient is bounded: $\|\nabla f\| \leq l$. Reciprocally, continuous functions gradient bounded by l are l -Lipschitz.

Definition 6 (Lipschitz neural network). *A Lipschitz neural network is a feedforward neural network with the additional constraints:*

- the activation function σ is S -Lipschitz. This is a standard assumption, frequently fulfilled in practice.
- the affine functions $x \mapsto Wx + b$ are U -Lipschitz, i.e $\|W\|_2 \leq U$. This is achieved in practice with spectrally normalized matrices [12] [13]. The feasible set is the ball $\{\|W\|_2 \leq U\}$ of radius U (which is convex), or a subset of thereof (not necessarily convex).

As a result, the function $x \mapsto f(\theta, x)$ is $U(US)^T$ -Lipschitz for all $\theta \in \Theta$.

Two strategies are available to enforce Lipschitzness:

1. With a differentiable reparametrization $\Pi : \mathbb{R}^p \rightarrow \Theta$ where $\tilde{\theta} = \Pi(\theta)$: the weights $\tilde{\theta}$ are used during the forward pass, but the gradients are back-propagated to θ through Π . This turns the training into an unconstrained optimization problem on the landscape of $\mathcal{L} \circ f \circ \Pi$.
2. With a suitable projection operator $\Pi : \mathbb{R}^p \rightarrow \Theta$: this is the celebrated Projected Gradient Descent (PGD) algorithm [58] applied on the landscape of $\mathcal{L} \circ f$.

For arbitrary re-parametrizations, option 1 can cause some difficulties: the Lipschitz constant of Π is generally unknown. However, if Θ is convex then Π is 1-Lipschitz (with respect to the norm chosen for the projection). To the contrary, option 2 elicits a broader set of feasible sets Θ . For simplicity, option 2 will be the focus of our work.

A.1.2 Gradient Norm Preserving networks

Definition 7 (Gradient Norm Preserving Networks). *GNP networks are 1-Lipschitz neural networks with the additional constraint that the Jacobian of layers consists of orthogonal matrices:*

$$\left(\frac{\partial f_d}{\partial x_d} \right)^T \left(\frac{\partial f_d}{\partial x_d} \right) = I. \quad (10)$$

This is achieved with GroupSort activation [8, 39], Householder activation [40], and orthogonal weight matrices [17, 41] or orthogonal convolutions (see [44, 45, 46] and references therein). Without biases these networks are also norm preserving: $\|f(\theta, x)\| = \|x\|$.

416 The set of orthogonal matrices (and its generalization the Stiefel manifold [20]) is not convex, and not
 417 even connected. Hence projected gradient approaches are mandatory: for re-parametrization methods
 418 the Jacobian $\frac{\partial \Pi}{\partial \theta}$ may be unbounded which could have uncontrollable consequences on sensitivity.

419 A.1.3 Differential Privacy background

420 **Definition 8** (Neighboring datasets). *A labelled dataset \mathcal{D} is a finite collection of input/label pairs*
 421 *$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$. Two datasets \mathcal{D} and \mathcal{D}' are said to be neighbouring if they*
 422 *differ by at most one sample: $\mathcal{D}' = \mathcal{D} \cup \{(x', y')\} - \{(x_i, y_i)\}$.*

423 The sensitivity is also referred as algorithmic stability [59], or bounded differences property in other
 424 fields [60]. We detail below the building of a Gaussian mechanism from an arbitrary query of known
 425 sensitivity.

426 **Definition 9** (Gaussian Mechanism). *Let $f : \mathcal{D} \rightarrow \mathbb{R}^p$ be a query accessing the dataset of known*
 427 *l_2 -sensitivity $\Delta(f)$, a Gaussian mechanism adds noise sampled from $\mathcal{N}(0, \sigma, \Delta(f))$ to the query f .*

428 **Property 1** (DP of Gaussian Mechanisms). *Let $\mathcal{G}(f)$. be a Gaussian mechanism of l_2 -sensitivity*
 429 *$S_2(f)$ adding the noise $\mathcal{N}(0, \sigma, S_2(f))$ to the query f . The DP guarantees of the mechanism are*
 430 *given by the following continuum: $\sigma = \sqrt{2 \cdot \log(1.25/\delta)}/\epsilon$.*

431 SGD is a composition of queries. Each of those query consists of sampling a minibatch from the
 432 dataset, and computing the gradient of the loss on the minibatch. The sensitivity of the query is
 433 proportional to the maximum gradient norm l , and inversely proportional to the batch size b . By
 434 perturbing the gradient with a Gaussian noise of variance $\sigma^2 \frac{l^2}{b^2}$ the query is transformed into a
 435 Gaussian mechanism. By composing the Gaussian mechanisms we obtain the DPSGD variant, that
 436 enjoy (ϵ, δ) -DP guarantees.

437 **Proposition 1** (DP guarantees for SGD, adapted from [1]). *Assume that the loss fulfills*
 438 *$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 \leq l$, and assume that the network is trained on a dataset of size N with SGD*
 439 *algorithm for T steps with noise scale $\mathcal{N}(\mathbf{0}, \sigma^2)$ such that:*

$$\sigma \geq \frac{16K \sqrt{T \log(2/\delta) \log(1.25T/\delta N)}}{N\epsilon}. \quad (11)$$

440 Then the SGD training of the network is (ϵ, δ) -DP.

441 A.2 More about Clipless DP-SGD

442 **About the “local” strategy.** Illustrated in Figures 3. We dissect the DP-mechanism that consists
 443 of the SGD steps applied on each layer. Indeed, we are able to characterise the sensitivity Δ_d of
 444 every layer of the network. Therefore, we give (ϵ_d, δ_d) -DP guarantees on a per-layer basis. Finally
 445 however, we would still have to be able to guarantee (ϵ, δ) -DP on the whole model.

- 446 1. On each layer, we apply a Gaussian mechanism [9] with noise variance $\sigma^2 \Delta_d^2$.
- 447 2. Their composition yields an other Gaussian mechanism with non isotropic noise.
- 448 3. The Gaussian mechanism benefits from privacy amplification via subsampling [31] thanks
 449 to the stochasticity in the selection of batches of size $b = pN$.
- 450 4. Finally an epoch is defined as the composition of $T = \frac{1}{p}$ sub-sampled mechanisms.

451 Different layers exhibit different maximum gradient bounds - and in turn this implies different
 452 sensitivities. This also suggests that different noise multipliers σ_d can be used for each layer. This
 453 open extensions for future work.

454 We detail in Algorithm 3 the global variant of our approach.

455 B Lipdp tutorial

456 This section gives advice on how to start your DP training processes using the framework. Moreover,
 457 it provides insights into how input pre-processing, building networks with Residual connections and
 458 Loss-logits gradient clipping could help offer better utility for the same privacy budget.

Algorithm 3 Clipless DP-SGD with **global** sensitivity accounting

Input: Feed-forward architecture $f(\cdot, \cdot) = f_{T+1}(\Theta_{T+1}, \cdot) \circ f_T(\Theta_T, \cdot) \circ \dots \circ f_0(\Theta_0, \cdot)$

Input: Initial weights θ_0 , learning rate scheduling η_t , noise multiplier σ .

1: **repeat**

2: $\Delta_0 \dots \Delta_{T+1} \leftarrow \text{compute_gradient_bounds}(f, X)$.

3: Sample a batch $\mathcal{B}_t = \{(x_1, y_1), (x_2, y_2), \dots, (x_b, y_b)\}$.

4: Compute the mean gradient of the batch for each layer t :

$$\tilde{g}_t := \frac{1}{b} \sum_{i=1}^b \nabla_{W_t} \mathcal{L}(\hat{y}_i, y_i).$$

5: For each layer t of the model, get the theoretical bound of the gradient:

$$\forall 1 \leq i \leq b, \quad \|\nabla_{W_t} \mathcal{L}(\hat{y}_i, y_i)\|_2 \leq \Delta_t.$$

6: Update Moment accountant state with **global** sensitivity $\Delta = \frac{2}{b} \sqrt{\sum_{t=1}^{T+1} \Delta_t^2}$.

7: Add global noise $\zeta \sim \mathcal{N}(0, 2\sigma\Delta/b)$ to each weights and perform projected gradient step:

$$W_t \leftarrow \Pi(W_t - \eta(\tilde{g}_t + \zeta)).$$

8: Report new (ϵ, δ) -DP guarantees with accountant.

9: **until** privacy budget (ϵ, δ) has been reached.

459 B.1 Prerequisite: building a l -Lipschitz network

460 As per def 3 a l -Lipschitz neural network of depth d can be built by composing $\sqrt[d]{l}$ layers. In
461 the rest of this section we will focus on 1-Lipschitz networks (rather than controlling l we control
462 the loss to obtain the same effects [18]). In order to do so the strategy consists in choosing only
463 1-Lipschitz activations, and to constrain the weights of parameterized layers such that it can only
464 express 1-Lipschitz functions. For instance normalizing the weight matrix of a dense layer by its
465 spectral norm yield a 1-Lipschitz layer (this, however cannot be applied trivially on convolution's
466 kernel). In practice we used the layers available in the open-source library *deel-lip*. In practice, when
467 building a Lipschitz network, the following block can be used:

- 468 • Dense layers: are available as *SpectralDense* or *QuickSpectralDense* which apply spectral
469 normalization and Björck orthogonalization (for GNP layers).
- 470 • *Relu* activations are replaced by *Groupsort2* activations, and such activations are GNP,
471 preventing vanishing gradient.
- 472 • pooling layers are replaced with *ScaledL2NormPooling*, which is GNP.
- 473 • normalization layers like *BatchNorm* are not K -Lipschitz. We did not accounted these
474 layers since they can induce a privacy leak (as they keep a rolling mean and variance). A
475 1-Lipschitz drop-in replacement is studied in C.2.3. The relevant literature also propose
476 drop-in replacement for this layer with proper sensitivity accounting.

477 Originally, this library relied on differentiable re-parametrizations, since it yields higher accuracy
478 outside DP training regime (clean training without noise). However, our framework does not
479 account for the Lipschitz constant of the re-parametrization operator. This is why we provide
480 *QuickSpectralDense* and *QuickSpectralConv2D* layers to enforce Lipschitz constraints, where the
481 projection is enforced with a tensorflow constraint. Note that *SpectralDense* and *SpectralConv2D*
482 can still be used with a *CondenseCallack* to enforce the projection, and bypass the back-propagation
483 through the differentiable re-parametrization. However this last solution, while being closer to the
484 original spirit of *deel-lip*, is also less efficient in speed benchmarks.

485 The Lipschitz constant of each layer is bounded by 1, since each weight matrix is divided by
486 the largest singular value σ_{\max} . This singular value is computed with Power Iteration algorithm.
487 Power Iteration computes the largest singular value by repeatedly computing a Rayleigh quotient
488 associated to a vector u , and this vector eventually converges to the eigenvector u_{\max} associated to
489 the largest eigenvalue. These iterations can be expensive. However, since gradient steps are smalls,

the weight matrices W_t and W_{t+1} remain close to each other after a gradient step. Hence their largest eigenvectors tend to be similar. Therefore, the eigenvector u_{\max} can be memorized at the end of each train step, and re-used as high quality initialization at the next train step, in a “lazy” fashion. This speed-up makes the overall projection algorithm very efficient.

B.2 Getting started

The framework we propose is built to allow DP training of neural networks in a fast and controlled approach. The tools we provide are the following :

1. **A pipeline** to efficiently load and pre-process the data of commonly used datasets like MNIST, FashionMNIST and CIFAR10.
2. **Configuration objects** to correctly account DP events we provide config objects to fill in that will
3. **Model objects** on the principle of Keras’ model classes we offer both a `DP_Model` and a `DP_Sequential` class to streamline the training process.
4. **Layer objects** where we offer a readily available form of the principal layers used for DNNs. These layers are already Lipschitz constrained and possess class specific methods to access their Lipschitz constant.
5. **Loss functions**, identically, we offer DP loss functions that automatically compute their Lipschitz constant for correct DP enforcing.

We highlight below an example of a full training loop on Mnist with **lip-dp** library. Refer to the “examples” folder in the library for more detailed explanations in a jupyter notebook.

```

510 dp_parameters = DPPParameters(
511     noisify_strategy="global",
512     noise_multiplier=2.0,
513     delta=1e-5,
514 )
515
516
517 epsilon_max = 3.0
518
519 input_upper_bound = 20.0
520 ds_train, ds_test, dataset_metadata = load_and_prepare_data(
521     "mnist",
522     batch_size=1000,
523     drop_remainder=True,
524     bound_fct=bound_clip_value(
525         input_upper_bound
526     ),
527 )
528
529 # construct DP_Sequential
530 model = DP_Sequential(
531     layers=[
532         layers.DP_BoundedInput(
533             input_shape=dataset_metadata.input_shape, upper_bound=
534                 input_upper_bound
535         ),
536         layers.DP_QuickSpectralConv2D(
537             filters=32,
538             kernel_size=3,
539             kernel_initializer="orthogonal",
540             strides=1,
541             use_bias=False,
542         ),
543         layers.DP_GroupSort(2),
544         layers.DP_ScaledL2NormPooling2D(pool_size=2, strides=2),
545         layers.DP_QuickSpectralConv2D(
546             filters=64,

```

```

547         kernel_size=3,
548         kernel_initializer="orthogonal",
549         strides=1,
550         use_bias=False,
551     ),
552     layers.DP_GroupSort(2),
553     layers.DP_ScaledL2NormPooling2D(pool_size=2, strides=2),
554
555     layers.DP_Flatten(),
556
557     layers.DP_QuickSpectralDense(512),
558     layers.DP_GroupSort(2),
559     layers.DP_QuickSpectralDense(dataset_metadata.nb_classes),
560 ],
561 dp_parameters=dp_parameters,
562 dataset_metadata=dataset_metadata,
563 )
564
565 model.compile(
566     loss=losses.DP_TauCategoricalCrossentropy(18.0),
567     optimizer=tf.keras.optimizers.SGD(learning_rate=2e-4, momentum
568         =0.9),
569     metrics=["accuracy"],
570 )
571 model.summary()
572
573 num_epochs = get_max_epochs(epsilon_max, model)
574
575 hist = model.fit(
576     ds_train,
577     epochs=num_epochs,
578     validation_data=ds_test,
579     callbacks=[
580         # accounting is done thanks to a callback
581         DP_Accountant(log_fn="logging"),
582     ],
583 )
584

```

585 B.3 Image spaces and input clipping

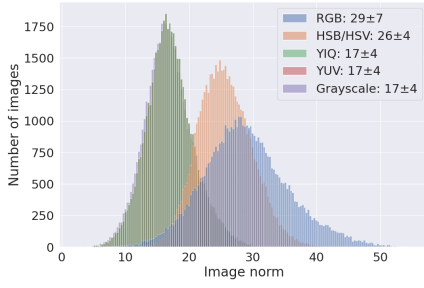
586 Input preprocessing can be done in a completely dataset agnostic way and may yield positive results
587 on the models utility. We explore here the choice of the color space, and the norm clipping of the
588 input.

589 B.3.1 Color space representations

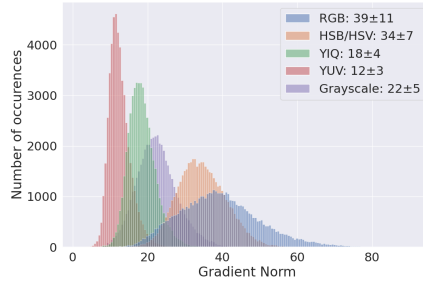
590 The color space representation of the color images of the CIFAR10 dataset for example can yield
591 very different gradient norms during the training process. Therefore, we can take advantage of this to
592 train our DP models more efficiently. Empirically, Figures 6a and 6b show that some color spaces
593 yield narrower image norm distributions that happen to be more advantageous to maximise the mean
594 gradient norm to noise ratio across all samples during the DP training process of GNP networks.

595 B.3.2 Input clipping

596 A clever way to narrow down the distribution of the dataset's norms would be to clip the norms of the
597 input of the model. This may result in improved utility since a narrower distribution of input norms
598 might maximise the mean gradient norm to noise ratio for misclassified examples. Also, we advocate
599 for the use of GNP networks as their gradients usually turn out to be closer to the upper bound we are
600 able to compute for the gradient. See Figure 7a and 7b

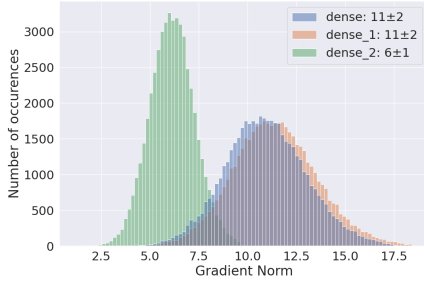


(a) Input norms.

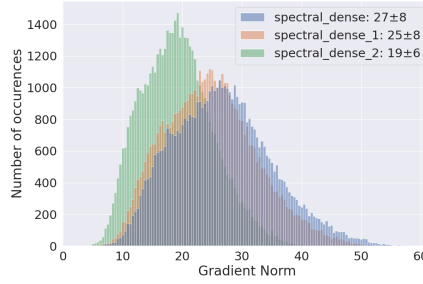


(b) Gradient norms.

Figure 6: **Histogram of norms for different image space on CIFAR-10 images.** We see that for GNP networks the distribution of dataset norms have a strong influence on the norm of the individual parameterwise gradient norms of misclassified examples.



(a) Conventional network.



(b) Gradient Norm Preserving network.

Figure 7: **Gradient norms of fully-connected networks on CIFAR-10.** We see that GNP networks exhibit a qualitatively different profile of gradient norms with respect to parameters, sticking closer to the upper bound we are able to compute for the gradient norm.

601 B.4 Practical implementation of Residual connections

602 The implementation of skip connections is made relatively straightforward in our framework by the
 603 `make_residuals` method. This function splits the input path in two, and wraps the layers inside the
 604 residual connections, as illustrated in figure 8.

```

605 from lipdp.layers import make_residuals
606 ## Manual implementation of residual connection:
607 layers = [DP_SplitResidual(),
608           DP_WrappedResidual(DP_QuickSpectralConv2D(16, (3, 3)),
609                               DP_WrappedResidual(DP_GroupSort(2)),
610                               DP_MergeResidual('1-lip-add'))]
611 ## Or equivalently, with helper function:
612 layers = make_residuals([
613     DP_QuickSpectralConv2D(16, (3, 3),
614     DP_GroupSort(2)
615 ])
616 
```

618 By using this implementation, the sensitivity of the gradient computation and the input bounds
 619 to each layer are correctly computed when the model's path is split. This allows for fairly easy
 620 implementations of models like the MLP-Mixer and ResNets. Since convolutional models may suffer
 621 from gradient vanishing and that dense based models are relatively restrictive in terms of architecture,
 622 implementing skip connections could be a useful feature for our framework.

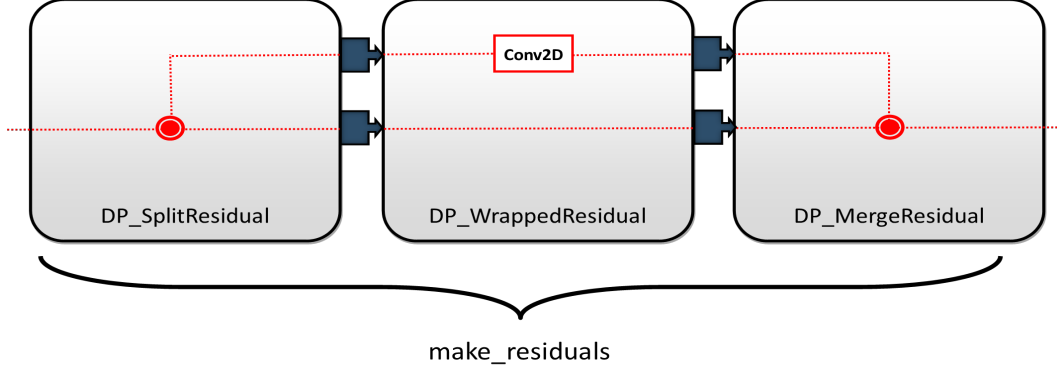


Figure 8: Implementation of a skip connection in the **lip-dp** framework. The meta block **DP_WrappedResidual** handle the forward propagation and the backward propagation of pairs of bounds (one for each computation path) by leveraging the forward and the backward of sub-blocks. **DP_SplitResidual** handle the creation of a tuple of input bounds at the forward, and collapse the tuple of gradient bounds into a scalar at the backward, while **DP_MergeResidual** does the opposite. All those operations are wrapped under the convenience function **make_residuals**.

623 B.5 Loss-logits gradient clipping

624 The advantage of the traditional DP-SGD approach is that through hyperparameter optimization
 625 on the global gradient clipping constant, we indirectly optimize the mean signal to noise ratio on
 626 misclassified examples. However, this clipping constant is not really explainable and rather just an
 627 empirical result of optimization on a given architecture and loss function.

628 Importantly, our framework is compatible with a more efficient and explainable form of clipping.
 629 Indeed, by introducing a gradient clipping layer in our framework we are able to clip the gradient of
 630 the loss on the final logits for a minimal cost.

631 Indeed, in this case the size of the clipped vector is the output dimension $|\hat{y}|$, which is small for a
 632 lot of practical regression and classification tasks. For example in CIFAR-10 the output vector is
 633 of length 10. This scales better with the batch size than the weight matrices that are typically of
 634 sizes 64×64 or 128×128 .

635 Note that this clipping value can also follow a scheduling during training, in the spirit of [61] - but
 636 care must be taken that the scheduling is either data independent, or in the case it is data dependant
 637 the privacy leaks must be taken into account. The implementation of loss gradient clipping scheduling
 638 is yet to be implemented in our framework. However, it is expected to be a part of future works.

639 Furthermore, if the gradient clipping layer is inserted on the tail of the network (between the logits
 640 and the loss) we can characterize its effects on the training, in particular for classification tasks with
 641 binary cross-entropy loss.

642 We denote by $\mathcal{L}(\text{Clip}_{\nabla}^C(\hat{y}))$ the loss wrapped under a **DP_ClipGradient** layer that behaves like
 643 identity $x \mapsto x$ at the forward, and clips the gradient norm to $C > 0$ in the backward pass:

$$\nabla_{\hat{y}} (\mathcal{L}(\text{Clip}_{\nabla}^C(\hat{y}))) := \min \left(1, \frac{C}{\|\nabla_{\hat{y}} \mathcal{L}(\hat{y})\|} \right) \nabla_{\hat{y}} \mathcal{L}(\hat{y}) \quad (12)$$

$$= \min (\|\nabla_{\hat{y}} \mathcal{L}(\hat{y})\|, C) \frac{\nabla_{\hat{y}} \mathcal{L}(\hat{y})}{\|\nabla_{\hat{y}} \mathcal{L}(\hat{y})\|}. \quad (13)$$

644 We denote by $\overrightarrow{g(\hat{y})}$ the unit norm vector $\frac{\nabla_{\hat{y}} \mathcal{L}(\hat{y})}{\|\nabla_{\hat{y}} \mathcal{L}(\hat{y})\|}$. Then:

$$\nabla_{\hat{y}} (\mathcal{L}(\text{Clip}_{\nabla}^C(\hat{y}))) = \min (\|\nabla_{\hat{y}} \mathcal{L}(\hat{y})\|, C) \overrightarrow{g(\hat{y})}.$$

645 **Proposition 2** (Clipped binary cross-entropy loss is the Wasserstein dual loss). *Let $\mathcal{L}_{BCE}(\hat{y}, y) =$
 646 $-\log(\sigma(\hat{y}y))$ be the binary cross-entropy loss, with $\sigma(\hat{y}y) = \frac{1}{1+\exp(-\hat{y}y)}$ the sigmoid activation,
 647 assuming discrete labels $y \in \{-1, +1\}$. Assume examples are sampled from the dataset \mathcal{D} . Let*

648 $\hat{y}(\theta, x) = f(\theta, x)$ be the predictions at input $x \in \mathcal{D}$. Then for every $C > 0$ sufficiently small, a
649 gradient descent step with the clipped gradient $\nabla_{\theta} \mathbb{E}_{\mathcal{D}}[\mathcal{L}(\text{Clip}_{\nabla}^C(\hat{y}, y))]$ is identical to the gradient
650 ascent step obtained from Kantorovich-Rubinstein loss $\mathcal{L}_{KR}(\hat{y}, y) = \hat{y}y$.

651 *Proof.* In the following we use the short notation \mathcal{L} in place of \mathcal{L}_{BCE} . Assume that examples with
652 labels $+1$ (resp. -1) are sampled from distribution P (resp. Q). By definition:

$$\nabla_{\theta} \mathbb{E}_{\mathcal{D}}[\mathcal{L}(\text{Clip}_{\nabla}^C(\hat{y}, y))] = \nabla_{\theta} (\mathbb{E}_{x \sim P}[\mathcal{L}(\text{Clip}_{\nabla}^C(f(\theta, x), +1))] + \mathbb{E}_{x \sim Q}[\mathcal{L}(\text{Clip}_{\nabla}^C(f(\theta, x), -1))]).$$

653 Observe that the output of the network is a single scalar, hence $\overrightarrow{g(\hat{y})} \in \{-1, +1\}$. We ap-
654 ply the chainrule $\nabla_{\theta} \mathcal{L} = \nabla_{\hat{y}} \mathcal{L} \frac{\partial \hat{y}}{\partial \theta} = \overrightarrow{g(\hat{y})} \min(\|\nabla_{\hat{y}} \mathcal{L}(\hat{y})\|, C) \nabla_{\theta} f(\theta, x)$. We note $R(\hat{y}, C) :=$
655 $\min(\|\nabla_{\hat{y}} \mathcal{L}(\hat{y})\|, C) > 0$ and we obtain:

$$\nabla_{\theta} \mathbb{E}_{\mathcal{D}}[\mathcal{L}(\text{Clip}_{\nabla}^C(\hat{y}, y))] = \nabla_{\theta} (\mathbb{E}_{x \sim P}[\overrightarrow{g(\hat{y})} R(\hat{y}, C) \nabla_{\theta} f(\theta, x)] + \mathbb{E}_{x \sim Q}[\overrightarrow{g(\hat{y})} R(\hat{y}, C) \nabla_{\theta} f(\theta, x)]). \quad (14)$$

656 Observe that the value of $\overrightarrow{g(\hat{y})}$ can actually be deduced from the label y , which gives:

$$\nabla_{\theta} \mathbb{E}_{\mathcal{D}}[\mathcal{L}(\hat{y}, y)] = -\mathbb{E}_{x \sim P}[R(\hat{y}, C) \nabla_{\theta} f(\theta, x)] + \mathbb{E}_{x \sim Q}[R(\hat{y}, C) \nabla_{\theta} f(\theta, x)]. \quad (15)$$

657 Observe that the function $x \mapsto \nabla_{\hat{y}} \mathcal{L}(\hat{y})$ is piecewise-continuous when the loss $\hat{y} \mapsto \mathcal{L}(\hat{y}, y)$ is
658 piecewise continuous. Observe that $|\nabla_{\hat{y}} \mathcal{L}(\hat{y})|$ is non zero, since the loss \mathcal{L} does not achieve its
659 minimum over the open set $(-\infty, +\infty)$, since $\sigma(\hat{y}) \in (0, 1)$. Assuming that the data $x \in \mathcal{D}$ live in
660 a compact (or equivalently that P and Q have compact support), since $x \mapsto |\nabla_{\hat{y}} \mathcal{L}(\hat{y})|$ is piecewise
661 continuous (with finite number of pieces for finite neural networks) it attains its minimum $C' > 0$.
662 Choosing any $C < C'$ implies that $R(\hat{y}, C) = C$, which yields:

$$\begin{aligned} \nabla_{\theta} \mathbb{E}_{\mathcal{D}}[\mathcal{L}(\text{Clip}_{\nabla}^C(\hat{y}, y))] &= -\mathbb{E}_{x \sim P}[C \nabla_{\theta} f(\theta, x)] + \mathbb{E}_{x \sim Q}[C \nabla_{\theta} f(\theta, x)] \\ &= -C (\mathbb{E}_{x \sim P}[\nabla_{\theta} f(\theta, x)] - \mathbb{E}_{x \sim Q}[\nabla_{\theta} f(\theta, x)]). \end{aligned}$$

663 This corresponds to a gradient ascent step of length C on the Kantorovich-Rubinstein (KR) objective
664 $\mathcal{L}_{KR}(\hat{y}, y) = \hat{y}y$. This loss is named after the Kantorovich Rubinstein duality that arises in optimal
665 transport, than states that the Wasserstein-1 distance is a supremum over 1-Lipschitz functions:

$$\mathcal{W}_1(P, Q) := \sup_{f \in \text{Lip}(\mathcal{D}, \mathbb{R})} \mathbb{E}_{x \sim P}[f(x)] - \mathbb{E}_{x \sim Q}[f(x)]. \quad (16)$$

666 Hence, with clipping C small enough the gradient steps are actually identical to the ones performed
667 during the estimation of Wasserstein-1 distance. \square

668 In future works, other multi-class losses can be studied through the lens of per-example clipping.
669 Our framework permits the use of gradient clipping while at the same time facilitating the theoretical
670 analysis.

671 B.5.1 Possible improvements

672 Our framework is compatible with possible improvements in methods of data pre-processing. For
673 instance some works suggest that feature engineering is the key to achieve correct utility/privacy
674 trade-off [50] some other work rely on heavily over-parametrized networks, coupled with batch size
675 [51]. While we focused on providing competitive and reproducible baselines (involving minimal
676 pre-processing and affordable compute budget) our work is fully compatible with those improvements.
677 Secondly the field of GNP networks (also called orthogonal networks) is still an active field, and new
678 methods to build better GNP networks will improve the efficiency of our framework (for instance
679 orthogonal convolutions [44],[41],[17],[42],[43] are still an active topic). Finally some optimizations
680 specific to our framework can also be developed: a scheduling of loss-logits clipping might allow for
681 better utility scores by following the declining value of the gradient of the loss, therefore allowing for
682 a better mean signal to noise ratio across a diminishing number of miss-classified examples.

683 C Computing Sensitivity Bounds

684 C.1 Losses bounds

685 This section contains the proofs related to the content of table 1. Our framework wraps over some
686 losses found in deal-lip library, that are wrapped by our framework to provide Lipschitz constant
687 automatically during backpropagation for bounds.

| Loss | Hyper-parameters | $\mathcal{L}(\hat{y}, y)$ | Lipschitz bound L |
|------------------------------|---|---|---------------------|
| Softmax Cross-entropy | temperature $\tau > 0$ | $y^T \log \text{softmax}(\hat{y}/\tau)$ | $\sqrt{2}/\tau$ |
| Cosine Similarity | bound $X_{\min} > 0$ | $\frac{y^T \hat{y}}{\max(\ \hat{y}\ _2, X_{\min})}$ | $1/X_{\min}$ |
| Multiclass Hinge | margin $m > 0$ | $\{\max(0, \frac{m}{2} - \hat{y}_i \cdot y_i)\}_{1 \leq i \leq K}$ | 1 |
| Kantorovich-Rubenstein | N/A | $\{\hat{y}, y\}$ | 1 |
| Hinge Kantorovich-Rubenstein | margin $m > 0$ regularization $\alpha > 0$ | $\alpha \cdot \mathcal{L}_{MH}(\hat{y}, y) + \mathcal{L}_{MKR}(\hat{y}, y)$ | $1 + \alpha$ |

Table 1: Lipschitz constant of common supervised classification losses used for the training of Lipschitz neural networks with k classes. Proofs in section C.1.

Multiclass Hinge This loss, with min margin m is computed in the following manner for a one-hot encoded ground truth vector y and a logit prediction \hat{y} :

$$\mathcal{L}_{MH}(\hat{y}, y) = \{\max(0, \frac{m}{2} - \hat{y}_1 \cdot y_1), \dots, \max(0, \frac{m}{2} - \hat{y}_k \cdot y_k)\}.$$

688 And $\|\frac{\partial}{\partial y} \mathcal{L}_{MH}(\hat{y}, y)\|_2 \leq \|\hat{y}\|_2$. Therefore $L_H = 1$.

689 **Multiclass Kantorovich Rubenstein** This loss, is computed in a one-versus all manner, for a
690 one-hot encoded ground truth vector y and a logit prediction \hat{y} :

$$\mathcal{L}_{MKR}(\hat{y}, y) = \{\hat{y}_1 - y_1, \dots, \hat{y}_k - y_k\}.$$

691 Therefore, by differentiating, we also get $L_{KR} = 1$.

692 **Multiclass Hinge - Kantorovitch Rubenstein** This loss, is computed in the following manner for
693 a one-hot encoded ground truth vector y and a logit prediction \hat{y} :

$$\mathcal{L}_{MHKR}(\hat{y}, y) = \alpha \mathcal{L}_{MH}(\hat{y}, y) + \mathcal{L}_{MKR}(\hat{y}, y).$$

694 By linearity we get $L_{HKR} = \alpha + 1$.

695 **Cosine Similarity** Cosine Similarity is defined in the following manner element-wise :

$$\mathcal{L}_{CS}(\hat{y}, y) = \frac{\hat{y}^T y}{\|\hat{y}\|_2 \|y\|_2}.$$

And y is one-hot encoded, therefore $\mathcal{L}_{CS}(\hat{y}, y) = \frac{\hat{y}_i}{\|\hat{y}\|_2}$. Therefore, the Lipschitz constant of this loss is dependant on the minimum value of \hat{y} . A reasonable assumption would be $\forall x \in \mathcal{D} : X_{\min} \leq \|x\|_2 \leq X_{\max}$. Furthermore, if the networks are Norm Preserving with factor K , we ensure that:

$$K X_{\min} \leq \|\hat{y}\|_2 \leq K X_{\max}.$$

696 Which yields: $L_{CS} = \frac{1}{K X_{\min}}$. The issue is that the exact value of K is never known in advance
697 since Lipschitz networks are rarely purely Norm Preserving in practice due to various effects (lack of
698 tightness in convolutions, or rectangular matrices that can not be perfectly orthogonal).

699 Realistically, we propose the following loss function in replacement:

$$\mathcal{L}_{K-CS}(\hat{y}, y) = \frac{\hat{y}_i}{\max(K X_{\min}, \|\hat{y}\|_2)}.$$

700 Where K is an input given by the user, therefore enforcing $L_{K-CS} = \frac{1}{K X_{\min}}$.

| Layer | Hyper parameters | $\ \frac{\partial f_t(\theta_t, x)}{\partial \theta_t}\ _2$ |
|-------------------|---------------------------------------|---|
| 1-Lipschitz dense | none | 1 |
| Convolution | window s | \sqrt{s} |
| RKO convolution | window s image size $H \times W$ | $\sqrt{1/((1 - \frac{(h-1)}{2H})(1 - \frac{(w-1)}{2W}))}$ |

Table 2: Lipschitz constant with respect to parameters in common Lipschitz layers. We report only the multiplicative factor that appears in front of the input norm $\|x\|_2$.

| Layer | Hyper parameters | $\ \frac{\partial f_t(\theta_t, x)}{\partial x}\ _2$ |
|--|------------------|--|
| Add bias | none | 1 |
| 1-Lipschitz dense | none | 1 |
| RKO convolution | none | 1 |
| Layer centering | none | 1 |
| Residual block | none | 2 |
| ReLU, GroupSort softplus, sigmoid, tanh | none | 1 |

Table 3: Lipschitz constant with respect to intermediate activations.

701 **Categorical Cross-entropy from logits** The logits are mapped into the probability simplex with
702 the *Softmax* function $\mathbb{R}^K \rightarrow (0, 1)^K$. We also introduce a temperature parameter $\tau > 0$, which hold
703 significance importance in the accuracy/robustness tradeoff for Lipschitz networks as observed by [18].
704 We assume the labels are discrete, or one-hot encoded: we do not cover the case of label smoothing.

$$S_j = \frac{\exp(\tau \hat{y}_j)}{\sum_i \exp(\tau \hat{y}_i)}. \quad (17)$$

705 We denote the prediction associated to the true label j^+ as S_{j^+} . The loss is written as:

$$\mathcal{L}(\hat{y}) = -\log(S_{j^+}). \quad (18)$$

706 Its gradient with respect to the logits is:

$$\nabla_{\hat{y}} \mathcal{L} = \begin{cases} \tau(S_{j^+} - 1) & \text{if } j = j^+, \\ \tau S_j & \text{otherwise} \end{cases} \quad (19)$$

The temperature factor τ is a multiplication factor than can be included in the loss itself, by using $\frac{1}{\tau} \mathcal{L}$ instead of \mathcal{L} . This formulation has the advantage of facilitating the tuning of the learning rate: this is the default implementation found in *deolip* library. The gradient can be written in vectorized form:

$$\nabla_{\hat{y}} \mathcal{L} = S - 1_{\{j=j^+\}}.$$

707 By definition of Softmax we have $\sum_{j \neq j^+} S_j^2 \leq 1$. Now, observe that $S_j \in (0, 1)$, and as a
708 consequence $(S_{j^+} - 1)^2 \leq 1$. Therefore $\|\nabla_{\hat{y}} \mathcal{L}\|_2^2 = \sum_{j \neq j^+} S_j^2 + (S_{j^+} - 1)^2 \leq 2$. Finally
709 $\|\nabla_{\hat{y}} \mathcal{L}\|_2 = \sqrt{2}$ and $L_{CCE} = \sqrt{2}$.

710 C.2 Layer bounds

711 The Lipschitz constant (with respect to input) of each layer of interest is summarized in table 3, while
712 the Lipschitz constant with respect to parameters is given in table 2.

713 C.2.1 Dense layers

714 Below, we illustrate the basic properties of Lipschitz constraints and their consequences for gradient
715 bounds computations. While for dense layers the proof is straightforward, the main ideas can be
716 re-used for all linear operations which includes the convolutions and the layer centering.

717 **Property 2. Gradients for dense Lipschitz networks.** Let $x \in \mathbb{R}^C$ be a data-point in space of
718 dimensions $C \in \mathbb{N}$. Let $W \in \mathbb{R}^{C \times F}$ be the weights of a dense layer with F features outputs. We
719 bound the spectral norm of the Jacobian as

$$\left\| \frac{\partial(W^T x)}{\partial W} \right\|_2 \leq \|x\|_2. \quad (20)$$

Proof. Since $W \mapsto W^T x$ is a linear operator, its Lipschitz constant is exactly the spectral radius:

$$\frac{\|W^T x - W'^T x\|_2}{\|W - W'\|_2} = \frac{\|(W - W')^T x\|_2}{\|W - W'\|_2} \leq \frac{\|W - W'\|_2 \|x\|_2}{\|W - W'\|_2} = \|x\|_2.$$

720 Finally, observe that the linear operation $x \mapsto W^T x$ is differentiable, hence the spectral norm of its
721 Jacobian is equal to its Lipschitz constant with respect to l_2 norm. \square

722 C.2.2 Convolutions

723 **Property 3. Gradients for convolutional Lipschitz networks.** Let $x \in \mathbb{R}^{S \times C}$ be an data-point
724 with channels $C \in \mathbb{N}$ and spatial dimensions $S \in \mathbb{N}$. In the case of a time serie S is the length of the
725 sequence, for an image $S = HW$ is the number of pixels, and for a video $S = HWN$ is the number
726 of pixels times the number of frames. Let $\Psi \in \mathbb{R}^{s \times C \times F}$ be the weights of a convolution with:

- 727 • window size $s \in \mathbb{N}$ (e.g $s = hw$ in 2D or $s = hwn$ in 3D),
- 728 • with C input channels,
- 729 • with $F \in \mathbb{N}$ output channels.
- 730 • we don't assume anything about the value of strides. Our bound is typically tighter for
731 strides=1, and looser for larger strides.

732 We denote the convolution operation as $(\Psi * \cdot) : \mathbb{R}^{S \times C} \rightarrow \mathbb{R}^{S \times F}$ with either zero padding, either
733 circular padding, such that the spatial dimensions are preserved. Then the Jacobian of convolution
734 operation with respect to parameters is bounded:

$$\left\| \frac{\partial(\Psi * x)}{\partial \Psi} \right\|_2 \leq \sqrt{s} \|x\|_2. \quad (21)$$

735 *Proof.* Let $y = \Psi * x \in \mathbb{R}^{S \times F}$ be the output of the convolution operator. Note that y can be uniquely
736 decomposed as sum of output feature maps $y = \sum_{f=1}^F y^f$ where $y^f \in \mathbb{R}^{S \times F}$ is defined as:

$$\begin{cases} (y^f)_{if} = y_{if} & \text{for all } 1 \leq i \leq S, \\ (y^f)_{ij} = 0 & \text{if } j \neq f. \end{cases}$$

737 Observe that $(y^f)^T y^{f'} = 0$ whenever $f \neq f'$. As a consequence Pythagorean theorem yields
738 $\|y\|_2^2 = \sum_{f=1}^F \|y^f\|_2^2$. Similarly we can decompose each output feature map as a sum of pixels
739 $y^f = \sum_{p=1}^S y^{pf}$. where $y^{pf} \in \mathbb{R}^{S \times F}$ fulfill:

$$\begin{cases} (y^{pf})_{ij} = 0 & \text{if } i \neq p, j \neq f, \\ (y^{pf})_{pf} = y_{pf} & \text{otherwise.} \end{cases}$$

Once again Pythagorean theorem yields $\|y^f\|_2^2 = \sum_{p=1}^S \|y^{pf}\|_2^2$. It remains to bound y^{pf} appropri-
ately. Observe that by definition:

$$y^{pf} = (\Psi * x)_{pf} = (\Psi^f)^T x^p[s].$$

where $\Psi^f \in \mathbb{R}^{s \times C}$ is a slice of Ψ corresponding to output feature map f , and $x^p[s] \in \mathbb{R}^{s \times C}$ denotes
the patch of size s centered around input element p . For example, in the case of images with $s = 3 \times 3$,
 p are the coordinates of a pixel, and $x^p[s]$ are the input feature maps of 3×3 pixels around it. We
apply Cauchy-Schwartz:

$$\|y^{pf}\|_2^2 \leq \|\Psi^f\|_2^2 \times \|x^p[s]\|_2^2.$$

740 By summing over pixels we obtain:

$$\|y^f\|_2^2 \leq \|\Psi^f\|_2^2 \sum_{p=1}^S \|x^p[s]\|_2^2, \quad (22)$$

$$\implies \|y\|_2^2 \leq \left(\sum_{f=1}^F \|\Psi^f\|_2^2 \right) \left(\sum_{p=1}^S \|x^p[s]\|_2^2 \right), \quad (23)$$

$$\implies \|y\|_2^2 \leq \|\Psi\|_2^2 \times \left(\sum_{p=1}^S \|x^p[s]\|_2^2 \right). \quad (24)$$

The quantity of interest is $\sum_{p=1}^S \|x^p[s]\|_2^2$ whose squared norm is the squared norm of all the patches used in the computation. With zero or circular padding, the norm of the patches cannot exceed those of input image. Note that each pixel belongs to atmost s patches, and even exactly s patches when circular padding is used:

$$\sum_{p=1}^S \|x^p[s]\|_2^2 \leq s \sum_{p=1}^S \|x_p\|_2^2 = s \|x\|_2^2.$$

Note that when $\text{strides} > 1$ the leading multiplicative constant is typically smaller than s , so this analysis can be improved in future work to take into account strided convolutions. Since Ψ is a linear operator, its Lipschitz constant is exactly its spectral radius:

$$\frac{\|(\Psi * x) - (\Psi' * x)\|_2}{\|\Psi - \Psi'\|_2} = \frac{\|(\Psi - \Psi') * x\|_2}{\|\Psi - \Psi'\|_2} \leq \frac{\sqrt{s} \|\Psi - \Psi'\|_2 \|x\|_2}{\|\Psi - \Psi'\|_2} = \sqrt{s} \|x\|_2.$$

Finally, observe that the convolution operation $\Psi * x$ is differentiable, hence the spectral norm of its Jacobian is equal to its Lipschitz constant with respect to l_2 norm:

$$\left\| \frac{\partial(\Psi * x)}{\partial \Psi} \right\|_2 \leq \sqrt{s} \|x\|_2.$$

741

□

742 An important case of interest are the convolutions based on Reshaped Kernel Orthogonalization
 743 (RKO) method introduced by [17]. The kernel Ψ is reshaped into 2D matrix of dimensions $(sC \times F)$
 744 and this matrix is orthogonalised). This is not sufficient to ensure that the operation $x \mapsto \Psi * x$ is
 745 orthogonal - however it is 1-Lipschitz and only *approximately* orthogonal under suitable re-scaling
 746 by $\mathcal{N} > 0$.

747 **Corollary 1** (Loss gradient for RKO convolutions.). *For RKO methods in 2D used in [48], the*
 748 *convolution kernel is given by $\Phi = \mathcal{N}\Psi$ where Ψ is an orthogonal matrix (under RKO) and $\mathcal{N} > 0$*
 749 *a factor ensuring that $x \mapsto \Phi * x$ is a 1-Lipschitz operation. Then, for RKO convolutions without*
 750 *strides we have:*

$$\left\| \frac{\partial(\Psi * x)}{\partial \Psi} \right\|_2 \leq \sqrt{\frac{1}{(1 - \frac{(h-1)}{2H})(1 - \frac{(w-1)}{2W})}} \|x\|_2. \quad (25)$$

751 where (H, W) are image dimensions and (h, w) the window dimensions. For large images with small
 752 receptive field (as it is often the case), the Taylor expansion in $h \ll H$ and $w \ll W$ yields a factor of
 753 magnitude $1 + \frac{(h-1)}{4H} + \frac{(w-1)}{4W} + \mathcal{O}(\frac{(w-1)(h-1)}{8HW}) \approx 1$.

754 C.2.3 Layer normalizations

755 **Property 4. Bounded loss gradient for layer centering.** *Layer centering is defined as $f(x) =$*
 756 *$x - (\frac{1}{n} \sum_{i=1}^n x_i) \mathbf{1}$ where $\mathbf{1}$ is a vector full of ones, and acts as a “centering” operation along some*
 757 *channels (or all channels). Then the singular values of this linear operation are:*

$$\sigma_1 = 0, \quad \text{and} \quad \sigma_2 = \sigma_3 = \dots = \sigma_n = 1. \quad (26)$$

758 In particular $\left\| \frac{\partial f}{\partial x} \right\|_2 \leq 1$.

759 *Proof.* It is clear that layer normalization is an affine layer. Hence the spectral norm of its Jacobian
760 coincides with its Lipschitz constant with respect to the input, which itself coincides with the
761 spectral norm of f . The matrix M associated to f is symmetric and diagonally dominant since
762 $|\frac{n-1}{n}| \geq \sum_{i=1}^{n-1} |\frac{-1}{n}|$. It follows that M is semi-definite positive. In particular all its eigenvalues
763 $\lambda_1 \leq \dots \leq \lambda_n$ are non negative. Furthermore they coincide with its singular values: $\sigma_i = \lambda_i$.
764 Observe that for all $r \in \mathbb{R}$ we have $f(r\mathbf{1}) = \mathbf{0}$, i.e the operation is null on constant vectors. Hence
765 $\lambda_1 = 0$. Consider the matrix $M - I$: its kernel is the eigenspace associated to eigenvalue 1. But the
766 matrix $M - I = \frac{-1}{n}\mathbf{1}\mathbf{1}^T$ is a rank-one matrix. Hence its kernel is of dimension $n - 1$, from which it
767 follows that $\lambda_2 = \dots = \lambda_n = \sigma_2 \dots = \sigma_n = 1$. \square

768 C.2.4 MLP Mixer architecture

769 The MLP-mixer architecture introduced in [54] consists of operations named *Token mixing* and
770 *Channel mixing* respectively. For token mixing, the input feature is split in disjoint patches on
771 which the same linear operation is applied. It corresponds to a convolution with a stride equal
772 to the kernel size. convolutions on a reshaped input, where patches of pixels are “collapsed” in
773 channel dimensions. Since the same linear transformation is applied on each patch, this can be
774 interpreted as a block diagonal matrix whose diagonal consists of W repeated multiple times. More
775 formally the output of Token mixing takes the form of $f(x) := [W^T x_1, W^T x_2, \dots W^T x_n]$ where
776 $x = [x_1, x_2, \dots, x_n]$ is the input, and the x_i ’s are the patches (composed of multiple pixels). Note
777 that $\|f(x)\|_2^2 \leq \sum_{i=1}^n \|W\|_2^2 \|x_i\|_2^2 = \|W\|_2^2 \sum_{i=1}^n \|x_i\|_2^2 = \|W\|_2^2 \|x\|_2^2$. If $\|W\|_2 = 1$ then the
778 layer is 1-Lipschitz - it is even norm preserving. Same reasoning apply for Channel mixing. Therefore
779 the MLP_Mixer architecture is 1-Lipchitz and the weight sensitivity is proportional to $\|x\|$.

780 **Lipschitz MLP mixer:** We adapted the original architecture in order to have an efficient 1-Lipschitz
781 version with the following changes:

- 782 • Relu activations were replaced with GroupSort, allowing a better gradient norm preservation,
- 783 • Dense layers were replaced with their GNP equivalent,
- 784 • Skip connections are available (adding a 0.5 factor to the output in order to ensure 1-lipschitz
- 785 condition) but architecture perform as well without these.

786 Finally the architecture parameters were selected as following:

- 787 1. The number of layer is reduced to a small value (between 1 and 4) to take advantage of the
- 788 theoretical sensitivity bound.
- 789 2. The patch size and hidden dimension are selected to achieve a sufficiently expressive network
- 790 (a patch size between 2 and 4 achieve accuracy without over-fitting and a hidden dim of
- 791 128-512 unlocks very large batch size).
- 792 3. The channel dim and token dim were set to value such that weight matrices are square
- 793 matrices (it is harder to guarantee that a network is GNP when the weight matrices are not
- 794 square).

795 D Experimental setup

796 D.1 Pareto fronts

797 We rely on Bayesian optimization [62] with Hyper-band [63] heuristic for early stopping. The
798 influence of some hyperparameters has to be highlighted to facilitate training with our framework,
799 therefore we provide a table that provides insights into the effects of principal hyperparameters
800 in Figure 9. Most hyper-parameters extend over different scales (such as the learning rate), so
801 they are sampled according to log-uniform distribution, to ensure fair covering of the search space.
802 Additionnaly, the importance of the softmax cross-entropy temperature τ has been demonstrated in
803 previous work [18].

804 D.1.1 Hyperparameters configuration for MNIST

805 Experiments are run on NVIDIA GeForce RTX 3080 GPUs. The losses we optimize are either the
806 Multiclass Hinge Kantorovich Rubinstein loss or the $\tau - \text{CCE}$.

| Hyperparameter Tuning | Influence on utility | Influence on privacy leakage per step |
|------------------------|---|--|
| Increasing Batch Size | Beneficial: Decreases the sensitivity of the gradient computation mechanism. | Detrimental: Reduces the privacy amplification by subsampling effects. |
| Loss Gradient Clipping | Mixed: - Beneficial: Tighter sensitivity bounds. - Detrimental: Biases the direction of the gradient. | No influence |
| Clipping Input Norms | Mixed: Limited Knowledge Could be the subject of future work | No influence |

Figure 9: **Hyperparameter table:** Here, we give insights on the effects of intervention on hyperparameters of the method.

| Hyper-Parameter | Minimum Value | Maximum Value |
|------------------------|---------------|-------------------|
| Input Clipping | 10^{-1} | 1 |
| Batch Size | 512 | 10^4 |
| Loss Gradient Clipping | 10^{-2} | NO CLIPPING |
| α (HKR) | 10^{-2} | $2,0 \times 10^3$ |
| τ (CCE) | 10^{-2} | $1,8 \times 10^1$ |

The sweeps were run with MLP or ConvNet architectures yielding the results presented in Figure 5a.

D.1.2 Hyperparameters configuration for FASHION-MNIST

Experiments are run on NVIDIA GeForce RTX 3080 GPUs. The losses we optimize are the Multiclass Hinge Kantorovich Rubinstein loss, the τ – CCE or the K-CosineSimilarity custom loss function.

| Hyper-Parameter | Minimum Value | Maximum Value |
|------------------------|-------------------|-------------------|
| Input Clipping | 10^{-1} | 1 |
| Batch Size | 5.0×10^3 | 10^4 |
| Loss Gradient Clipping | 10^{-2} | NO CLIPPING |
| α (HKR) | 10^{-2} | 2.0×10^3 |
| τ (CCE) | 10^{-2} | 4.0×10^1 |
| K (K-CS) | 10^{-2} | 1.0 |

A simple ConvNet architecture was chosen to run all sweeps yielding the results we present in Figure 5b.

D.1.3 Hyperparameters configuration for CIFAR-10

Experiments are run on NVIDIA GeForce RTX 3080 or 3090 GPUs. The losses we optimize are the Multiclass Hinge Kantorovich Rubinstein loss, the τ – CCE or the K-CosineSimilarity custom loss function. They yield the results of Figure 5c.

| Hyper-Parameter | Minimum Value | Maximum Value |
|------------------------|----------------------|-------------------|
| Input Clipping | 10^{-2} | 1 |
| Batch Size | 512 | 10^4 |
| Loss Gradient Clipping | 2.0×10^{-2} | NO CLIPPING |
| α (HKR) | 10^{-2} | 2.0×10^3 |
| τ (CCE) | 10^{-3} | 3.2×10^1 |
| K (K-CS) | 10^{-2} | 1.0 |

The sweeps have been done on various architectures such as Lipschitz VGGs, Lipschitz ResNets and Lipschitz MLP_Mixer. We can also break down the results per architecture, in figure 10. The

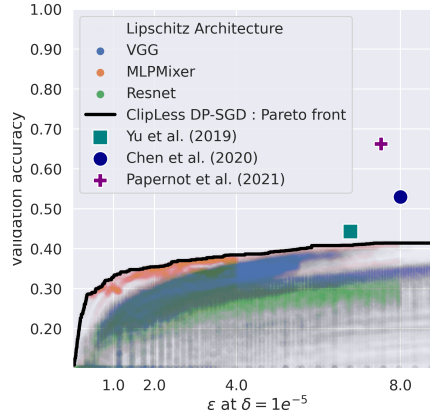


Figure 10: Accuracy/Privacy tradeoff on Cifar-10, split down per architecture used. While some architectures seems to perform better than others, we don’t advocate for the use of one over another. The results may not translate to all datasets, and may be highly dependant on the range chosen for hyper-parameters. While this figure provides valuable insights, identifying the best architecture is left for future works.

MLP_Mixer architecture seems to yield the best results. This architecture is exactly GNP since the orthogonal linear transformations are applied on disjoint patches. To the contrary, VGG and Resnets are based on RKO convolutions which are not exactly GNP. Hence those preliminary results are compatible with our hypothesis that GNP layers should improve performance. Note that these results are expected to change as the architectures are further improved. It is also dependant of the range chosen for hyper-parameters. We do not advocate for the use of an architecture over another, and we believe many other innovations found in literature should be included before settling the question definitively.

D.2 Configuration of speed experiment

We detail below the environment version of each experiment, together with Cuda and cudnn versions. We rely on machine with 32GB RAM and a NVIDIA Quadro GTX 8000 graphic card with 48GB memory. The GPU uses driver version 495.29.05, cuda 11.5 (October 2021) and cudnn 8.2 (June 7, 2021). We use Python 3.8.

- For Jax, we used jax 0.3.17 (Aug 31, 2022) with jaxlib 0.3.15 (July 23, 2022), flax 0.6.0 (Aug 17, 2022) and optax 1.4.0 (Nov 21, 2022).
- For Tensorflow, we used tensorflow 2.12 (March 22, 2023) with tensorflow_privacy 0.7.3 (September 1, 2021).
- For Pytorch, we used Opacus 1.4.0 (March 24, 2023) with Pytorch (March 15, 2023).
- For lip-dp we used deel-lip 1.4.0 (January 10, 2023) on Tensorflow 2.8 (May 23, 2022).

For this benchmark, we used among the most recent packages on pypi. However the latest version of tensorflow privacy could not be forced with pip due to broken dependencies. This issue arise in clean environments such as the one available in google colaboratory.

D.3 Drop-in replacement with Lipschitz networks in vanilla DPSGD

Thanks to the gradient clipping of DP-SGD (see Algorithm 4), Lipschitz networks can be readily integrated in traditional DP-SGD algorithm with gradient clipping. The PGD algorithm is not mandatory: the back-propagation can be performed within the computation graph through iterations of Björck algorithm (used in RKO convolutions). This does not benefit from any particular speed-up over conventional networks - quite to the contrary there is an additional cost incurred by enforcing

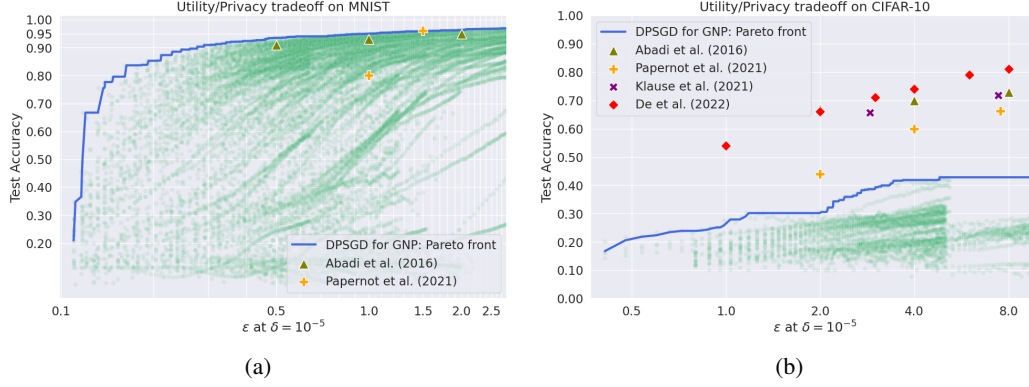


Figure 11: Privacy/utility trade-off for Gradient Norm Preserving networks trained under “vanilla” DP-SGD (**with** gradient clipping). Each green dot corresponds to a single epoch of one of the runs. Trajectories that end abruptly are due to the automatic early stopping of unpromising runs. Note that clipping + orthogonalization have a high runtime cost, which limits the number of epochs that reported.

850 Lipschitz constraints in the graph. Some layers of deeplip library have been recoded in Jax/Flax, and
 851 the experiment was run in Jax, since Tensorflow was too slow.

852 We use the Total Amount of Noise (TAN) heuristic introduced in [64] to heuristically tune
 853 hyper-parameters jointly. This ensures fair covering of the Pareto front. Results are exposed in
 854 Figures 11a and 11b.

Algorithm 4 Differentially Private Stochastic Gradient Descent : **DP-SGD**

Input: Neural network architecture $f(\cdot, \cdot)$

Input: Initial weights θ_0 , learning rate scheduling η_t , number of steps N , noise multiplier σ , L2 clipping value C .

1: **repeat**

2: **for all** $1 \leq t \leq N - 1$ **do**

3: Sample a batch

$$\mathcal{B}_t = (x_1, y_1), (x_2, y_2), \dots, (x_b, y_b).$$

4: Create microbatches, compute and clip the per-sample gradient of cost function:

$$\tilde{g}_{t,i} := \max(C, \nabla_{\theta_t} \mathcal{L}(\hat{y}_i, y_i)).$$

5: Perturb each microbatch with carefully chosen noise distribution $b \sim \mathcal{N}(0, \sigma C)$:

$$\hat{g}_{t,i} \leftarrow \tilde{g}_{t,i} + b_i.$$

6: Perform projected gradient step:

$$\theta_{t+1} \leftarrow \Pi(\theta_t - \eta_t \hat{g}_{t,i}).$$

7: **end for**

8: **until** privacy budget (ϵ, δ) has been reached.

855 D.4 Extended limitations

856 The main weakness of our approach is that it crucially rely on accurate computation of the sensitiv-
 857 ity Δ . This task faces many challenges in the context of differential privacy: floating point arithmetic
 858 is not associative, and summation order can have dramatic consequences regarding numerical
 859 stability [55]. This is further amplified on the GPUs, where some operations are intrinsically non
 860 deterministic [56]. This well known issue is already present in vanilla DP-SGD algorithm. Our
 861 framework adds an additional point of failure: the upper bound of spectral Jacobian must be com-

puted accurately. Hence Power Iteration must be run with sufficiently high number of iterations to ensure that the projection operator Π works properly. The (ϵ, δ) -DP certificates only hold under the hypothesis that all computations are correct, as numerical errors can induce privacy leakages. Hence we check empirically the effective norm of the gradient in the training loop at the end of each epoch. No certificate violations were reported during ours experiments, which suggests that the numerical errors can be kept under control.

E Proofs of general results

This section contains the proofs of results that are either informally presented in the main paper, either formally defined in section A.2.

The informal Theorem 1 requires some tools that we introduce below.

Additional hypothesis for GNP networks. We introduce convenient assumptions for the purpose of obtaining tight bounds in Algorithm 2.

Assumption 1 (Bounded biases). *We assume there exists $B > 0$ such that for all biases b_d we have $\|b_d\| \leq B$. Observe that the ball $\{\|b\|_2 \leq B\}$ of radius B is a **convex** set.*

Assumption 2 (Zero preserving activation). *We assume that the activation fulfills $\sigma(0) = 0$. When σ is S -Lipschitz this implies $\|\sigma(x)\| \leq S\|x\|$ for all x . Examples of activations fulfilling this constraints are ReLU, Groupsort, GeLU, ELU, tanh. However it does not work with sigmoid or softplus.*

We also propose the assumption 3 for convenience and exhaustivity.

Assumption 3 (Bounded activation). *We assume it exists $G > 0$ such that for every $x \in \mathcal{X}$ and every $1 \leq d \leq D + 1$ we have:*

$$\|h_d\| \leq G \quad \text{and} \quad \|z_d\| \leq G. \quad (27)$$

Note that this assumption is implied by requirement 2, assumption 1-2, as illustrated in proposition 3.

In practice assumption 3 can be fulfilled with the use of input clipping and bias clipping, bounded activation functions, or layer normalization. This assumption can be used as a “shortcut” in the proof of the main theorem, to avoid the “propagation of input bounds” step.

E.1 Main result

We rephrase in a rigorous manner the informal theorem of section 3.1. The proofs are given in section B. In order to simplify the notations, we use $X := X_0$ in the following.

Proposition 3. Norm of intermediate activations. *Under requirement 2, assumptions 1-2 we have:*

$$\|h_t\| \leq S\|z_t\| \leq \begin{cases} (US)^t \left(X - \frac{SB}{1-SU} \right) + \frac{SB}{1-SU} & \text{if } US \neq 1, \\ SX + tSB & \text{otherwise.} \end{cases} \quad (28)$$

In particular if there are no biases, i.e if $B = 0$, then $\|h_t\| \leq S\|z_t\| \leq SX$.

Proposition 3 can be used to replace assumption 3.

Proposition 4. Lipschitz constant of dense Lipschitz networks with respect to parameters. *Let $f(\cdot, \cdot)$ be a Lipschitz neural network. Under requirement 2, assumptions 1-2 we have for every $1 \leq t \leq T + 1$:*

$$\left\| \frac{\partial f(\theta, x)}{\partial b_t} \right\|_2 \leq (SU)^{T+1-t}, \quad (29)$$

$$\left\| \frac{\partial f(\theta, x)}{\partial W_t} \right\|_2 \leq (SU)^{T+1-t} \|h_{t-1}\|. \quad (30)$$

In particular, for every $x \in \mathcal{X}$, the function $\theta \mapsto f(\theta, x)$ is Lipschitz bounded.

Proposition 4 suggests that the scale of the activation $\|h_t\|$ must be kept under control for the gradient scales to distribute evenly along the computation graph. It can be easily extended to a general result on the *per sample* gradient of the loss, in theorem 1.

Theorem 1. Bounded loss gradient for dense Lipschitz networks. Assume the predictions are given by a Lipschitz neural network f :

$$\hat{y} := f(\theta, x). \quad (31)$$

Under requirements 1-2, assumptions 1-2, there exists a $K > 0$ for all $(x, y, \theta) \in \mathcal{X} \times \mathcal{Y} \times \Theta$ the loss gradient is bounded:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 \leq K. \quad (32)$$

Let $\alpha = SU$ be the maximum spectral norm of the Jacobian between two consecutive layers.

If $\alpha = 1$ then we have:

$$K = \mathcal{O} \left(LX + L\sqrt{T} + LSX\sqrt{T} + L\sqrt{BX}ST + LBST^{3/2} \right). \quad (33)$$

The case $S = 1$ is of particular interest since it covers most activation function (i.e ReLU, GroupSort):

$$K = \mathcal{O} \left(L\sqrt{T} + LX\sqrt{T} + L\sqrt{BX}T + LBT^{3/2} \right). \quad (34)$$

Further simplification is possible if we assume $B = 0$, i.e a network without biases:

$$K = \mathcal{O} \left(L\sqrt{T}(1 + X) \right). \quad (35)$$

If $\alpha > 1$ then we have:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L \frac{\alpha^T}{\alpha - 1} \left(\sqrt{T}(\alpha X + SB) + \frac{\alpha(SB + \alpha)}{\sqrt{\alpha^2 - 1}} \right) \right). \quad (36)$$

Once again $B = 0$ (network with no bias) leads to useful simplifications:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L \frac{\alpha^{T+1}}{\alpha - 1} \left(\sqrt{T}X + \frac{\alpha}{\sqrt{\alpha^2 - 1}} \right) \right). \quad (37)$$

We notice that when $\alpha \gg 1$ there is an **exploding gradient** phenomenon where the upper bound become vacuous.

If $\alpha < 1$ then we have:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L\alpha^T \left(X\sqrt{T} + \frac{1}{(1 - \alpha^2)} \left(\sqrt{\frac{XSB}{\alpha^T}} + \frac{SB}{\sqrt{(1 - \alpha)}} \right) \right) + \frac{L}{(1 - \alpha)\sqrt{1 - \alpha}} \right). \quad (38)$$

For network without biases we get:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L\alpha^T X\sqrt{T} + \frac{L}{\sqrt{(1 - \alpha)^3}} \right). \quad (39)$$

The case $\alpha \ll 1$ is a **vanishing gradient** phenomenon where $\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2$ is now independent of the depth T and of the input scale X .

The informal theorem of section 3.1 is based on the aforementioned bounds, that have been simplified. Note that the definition of network differs slightly: in definition 3 the activations and the affine layers are considered independent and indexed differently, while the theoretical framework merge them into z_t and h_t respectively, sharing the same index t . This is without consequences once we realize that if $K = U = S$ and $2T = D$ then $(US)^2 = \alpha^2 = K^2$ leads to $\alpha^{2T} = K^D$. The leading constant factors based on α value have been replaced by 1 since they do not affect the asymptotic behavior.

Main paper submission erratum: at time of submission the informal theorem of section 3.1 contains a small mistake (a missing \sqrt{D} factor in case **3.**, and a missing constant in the case **2.**), which has no consequences on the code, the framework or the theoretical analysis. This will be corrected in the revised version.

922 E.2 Proof of main result

923 Propositions 3 and 4 were introduced for clarity. They are a simple consequence of the Lemmas 1-3-4
 924 used in the proof of Theorem 1. See the proof below for a complete exposition of the arguments.

925 **Theorem 1. Bounded loss gradient for dense Lipschitz networks.** Assume the predictions are
 926 given by a Lipschitz neural network f :

$$\hat{y} := f(\theta, x). \quad (31)$$

927 Under requirements 1-2, assumptions 1-2, there exists a $K > 0$ for all $(x, y, \theta) \in \mathcal{X} \times \mathcal{Y} \times \Theta$ the
 928 loss gradient is bounded:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 \leq K. \quad (32)$$

929 Let $\alpha = SU$ be the maximum spectral norm of the Jacobian between two consecutive layers.

If $\alpha = 1$ then we have:

$$K = \mathcal{O} \left(LX + L\sqrt{T} + LSX\sqrt{T} + L\sqrt{BXST} + LBS T^{3/2} \right). \quad (33)$$

930 The case $S = 1$ is of particular interest since it covers most activation function (i.e ReLU, GroupSort):

$$K = \mathcal{O} \left(L\sqrt{T} + LX\sqrt{T} + L\sqrt{BXT} + LBT^{3/2} \right). \quad (34)$$

931 Further simplification is possible if we assume $B = 0$, i.e a network without biases:

$$K = \mathcal{O} \left(L\sqrt{T}(1 + X) \right). \quad (35)$$

If $\alpha > 1$ then we have:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L \frac{\alpha^T}{\alpha - 1} \left(\sqrt{T}(\alpha X + SB) + \frac{\alpha(SB + \alpha)}{\sqrt{\alpha^2 - 1}} \right) \right). \quad (36)$$

932 Once again $B = 0$ (network with no bias) leads to useful simplifications:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L \frac{\alpha^{T+1}}{\alpha - 1} \left(\sqrt{T}X + \frac{\alpha}{\sqrt{\alpha^2 - 1}} \right) \right). \quad (37)$$

933 We notice that when $\alpha \gg 1$ there is an **exploding gradient** phenomenon where the upper bound
 934 become vacuous.

If $\alpha < 1$ then we have:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L\alpha^T \left(X\sqrt{T} + \frac{1}{(1 - \alpha^2)} \left(\sqrt{\frac{XSB}{\alpha^T}} + \frac{SB}{\sqrt{(1 - \alpha)}} \right) \right) + \frac{L}{(1 - \alpha)\sqrt{1 - \alpha}} \right). \quad (38)$$

935 For network without biases we get:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L\alpha^T X\sqrt{T} + \frac{L}{\sqrt{(1 - \alpha)^3}} \right). \quad (39)$$

936 The case $\alpha \ll 1$ is a **vanishing gradient** phenomenon where $\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2$ is now independent of
 937 the depth T and of the input scale X .

938 *Proof.* The control of gradient implicitly depend on the scale of the output of the network at every
 939 layer, hence it is crucial to control the norm of each activation.

940 **Lemma 1** (Bounded activations). If $US \neq 1$ for every $1 \leq t \leq T + 1$ we have:

$$\|z_t\| \leq U^t S^{t-1} \left(X - \frac{SB}{1 - SU} \right) + \frac{B}{1 - SU}. \quad (40)$$

941 If $US = 1$ we have:

$$\|z_t\| \leq X + tB. \quad (41)$$

942 In every case we have $\|h_t\| \leq S\|z_t\|$.

943 *Lemma proof.* From assumption 2, if we assume that σ is S -Lipschitz, we have:

$$\|h_t\| = \|\sigma(z_t)\| = \|\sigma(z_t) - \sigma(\mathbf{0})\| \leq S\|z_t\|. \quad (42)$$

944 Now, observe that:

$$\|z_{t+1}\| = \|W_{t+1}h_t + b_{t+1}\| \leq \|W_{t+1}\|\|h_t\| + \|b_{t+1}\| \leq US\|z_t\| + B. \quad (43)$$

945 Let $u_1 = UX + B$ and $u_{t+1} = SUu_t + B$ be a linear recurrence relation. The translated sequence
 946 $u_t - \frac{B}{1-SU}$ is a geometric progression of ratio SU , hence $u_t = (SU)^{t-1}(UX + B - \frac{B}{1-SU}) + \frac{B}{1-SU}$.
 947 Finally we conclude that by construction $\|z_t\| \leq u_t$. ■

948 The activation jacobians can be bounded by applying the chainrule. The recurrence relation obtained
 949 is the one automatically computed with back-propagation.

950 **Lemma 2** (Bounded activation derivatives). *For every $T + 1 \geq s \geq t \geq 1$ we have:*

$$\left\| \frac{\partial z_s}{\partial z_t} \right\| \leq (SU)^{s-t}. \quad (44)$$

951 *Lemma proof.* The chain rule expands as:

$$\frac{\partial z_s}{\partial z_t} = \frac{\partial z_s}{\partial h_{s-1}} \frac{\partial h_{s-1}}{\partial z_{s-1}} \frac{\partial z_{s-1}}{\partial z_t}. \quad (45)$$

952 From Cauchy-Schwartz inequality we get:

$$\left\| \frac{\partial z_s}{\partial z_t} \right\| \leq \left\| \frac{\partial z_s}{\partial h_{s-1}} \right\| \cdot \left\| \frac{\partial h_{s-1}}{\partial z_{s-1}} \right\| \cdot \left\| \frac{\partial z_{s-1}}{\partial z_t} \right\|. \quad (46)$$

953 Since σ is S -Lipschitz, and $\|W_s\| \leq U$, and by observing that $\left\| \frac{\partial z_t}{\partial z_t} \right\| = 1$ we obtain by induction
 954 that:

$$\left\| \frac{\partial h_s}{\partial h_t} \right\| \leq (SU)^{s-t}. \quad (47)$$

955 ■

956 The derivatives of the biases are a textbook application of the chainrule.

957 **Lemma 3** (Bounded bias derivatives). *For every t we have:*

$$\|\nabla_{b_t} \mathcal{L}(\hat{y}, y)\| \leq L(SU)^{T+1-t}. \quad (48)$$

958 *Lemma proof.* The chain rule yields:

$$\nabla_{b_t} \mathcal{L}(\hat{y}, y) = (\nabla_{\hat{y}} \mathcal{L}(\hat{y}, y)) \frac{\partial z_{T+1}}{\partial z_t} \frac{\partial z_t}{\partial b_t}. \quad (49)$$

959 Hence we have:

$$\|\nabla_{b_t} \mathcal{L}(\hat{y}, y)\| = \|\nabla_{\hat{y}} \mathcal{L}(\hat{y}, y)\| \cdot \left\| \frac{\partial z_{T+1}}{\partial z_t} \right\| \cdot \left\| \frac{\partial z_t}{\partial b_t} \right\|. \quad (50)$$

960 We conclude with Lemma 2 that states $\left\| \frac{\partial z_{T+1}}{\partial z_t} \right\| \leq (US)^{T+1-t}$, with requirement 1 that states
 961 $\|\nabla_{\hat{y}} \mathcal{L}(\hat{y}, y)\| \leq L$ and by observaing that $\left\| \frac{\partial z_t}{\partial b_t} \right\| = 1$. ■

962 We can now bound the derivative of the affine weights:

963 **Lemma 4** (Bounded weight derivatives). *For every $T + 1 \geq t \geq 2$ we have:*

$$\|\nabla_{W_t} \mathcal{L}(\hat{y}, y)\| \leq L(SU)^T \left(X - \frac{SB}{1-SU} \right) + L(SU)^{T+1-t} \frac{SB}{1-SU} \text{ when } SU \neq 1, \quad (51)$$

$$\|\nabla_{W_t} \mathcal{L}(\hat{y}, y)\| \leq LS(X + (t-1)B) \text{ when } SU = 1. \quad (52)$$

$$(53)$$

964 *In every case:*

$$\|\nabla_{W_1} \mathcal{L}(\hat{y}, y)\| \leq L(SU)^T X. \quad (54)$$

965 *Lemma proof.* We proceed like in the proof of Lemma 3 and we get:

$$\|\nabla_{W_t} \mathcal{L}(\hat{y}, y)\| \leq \|\nabla_{\hat{y}} \mathcal{L}(\hat{y}, y)\| \cdot \left\| \frac{\partial z_{T+1}}{\partial z_t} \right\| \cdot \left\| \frac{\partial z_t}{\partial W_t} \right\|. \quad (55)$$

966 Which then yields:

$$\|\nabla_{W_t} \mathcal{L}(\hat{y}, y)\| \leq L(SU)^{T+1-t} \cdot \left\| \frac{\partial z_t}{\partial W_t} \right\|. \quad (56)$$

967 Now, for $T + 1 \geq t \geq 1$, according to Lemma 1 we either have:

$$\left\| \frac{\partial z_t}{\partial W_t} \right\| \leq \|h_{t-1}\| \leq S\|z_{t-1}\| = (SU)^{t-1} \left(X - \frac{SB}{1-SU} \right) + \frac{SB}{1-SU}, \quad (57)$$

968 or, when $US = 1$:

$$\left\| \frac{\partial z_t}{\partial W_t} \right\| \leq \|h_{t-1}\| = S\|z_{t-1}\| = SX + (t-1)SB \text{ if } t \geq 2, \quad (58)$$

$$\left\| \frac{\partial z_t}{\partial W_t} \right\| \leq X \text{ otherwise.} \quad (59)$$

969

■

970 Now, the derivatives of the loss with respect to each type of parameter (i.e W_t or b_t) are know, and
971 they can be combined to retrieve the overall gradient vector.

$$\theta = \{(W_1, b_1), (W_2, b_2), \dots, (W_{T+1}, b_{T+1})\}. \quad (60)$$

972 We introduce $\alpha = SU$.

973 **Case $\alpha = 1$.** The resulting norm is given by the series:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2^2 = \sum_{t=1}^{T+1} \|\nabla_{b_t} \mathcal{L}(\hat{y}, y)\|_2^2 + \|\nabla_{W_t} \mathcal{L}(\hat{y}, y)\|_2^2 \quad (61)$$

$$\leq L^2 \left((1 + X^2) + \sum_{t=2}^{T+1} (1 + (SX + (t-1)SB)^2) \right) \quad (62)$$

$$\leq L^2 \left(1 + X^2 + \sum_{u=1}^T (1 + (SX + uSB)^2) \right) \quad (63)$$

$$\leq L^2 \left(1 + X^2 + \sum_{u=1}^T (1 + S^2(X^2 + 2uBX + u^2B^2)) \right) \quad (64)$$

$$\leq L^2 \left(1 + X^2 + T(1 + S^2X^2) + S^2BXT(T+1) + S^2B^2 \frac{T(T+1)(2T+1)}{6} \right). \quad (65)$$

974 Finally:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O}(L\sqrt{X^2 + T + TS^2X^2 + BS^2XT^2 + B^2S^2T^3}) \quad (66)$$

$$= \mathcal{O}\left(LX + L\sqrt{T} + LSX\sqrt{T} + L\sqrt{BXST} + LBST^{3/2}\right). \quad (67)$$

975 This upper bound depends (asymptotically) linearly of $L, X, S, B, T^{3/2}$, when other factors are kept
976 fixed to non zero value.

977 **Case $\alpha \neq 1$.** We introduce $\beta = \frac{SB}{1-\alpha}$.

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2^2 = \sum_{t=1}^{T+1} \|\nabla_{b_t} \mathcal{L}(\hat{y}, y)\|_2^2 + \|\nabla_{W_t} \mathcal{L}(\hat{y}, y)\|_2^2 \quad (68)$$

$$\leq L^2 \left(\alpha^{2T} \sum_{t=1}^{T+1} (((X - \beta) + \alpha^{1-t} \beta)^2 + \alpha^{2-2t}) \right) \quad (69)$$

$$\leq L^2 \alpha^{2T} \left(\sum_{u=0}^T (((X - \beta)^2 + 2(X - \beta) \alpha^{-u} \beta + \alpha^{-2u} \beta^2) + \alpha^{-2u}) \right) \quad (70)$$

$$\leq L^2 \alpha^{2T} \left((T+1)(X - \beta)^2 + 2(X - \beta) \beta \sum_{u=0}^T \alpha^{-u} + (\beta^2 + 1) \sum_{u=0}^T \alpha^{-2u} \right) \quad (71)$$

$$\leq L^2 \alpha^{2T} \left((T+1)(X - \beta)^2 + 2(X - \beta) \beta \frac{\alpha - (\frac{1}{\alpha})^T}{\alpha - 1} + (\beta^2 + 1) \frac{\alpha^2 - (\frac{1}{\alpha^2})^T}{\alpha^2 - 1} \right). \quad (72)$$

978 Finally:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 \leq L \alpha^T \sqrt{(T+1)(X - \beta)^2 + 2(X - \beta) \beta \frac{\alpha - (\frac{1}{\alpha})^T}{\alpha - 1} + (\beta^2 + 1) \frac{\alpha^2 - (\frac{1}{\alpha^2})^T}{\alpha^2 - 1}}. \quad (73)$$

979 Now, the situation is a bit different for $\alpha < 1$ and $\alpha > 1$. One case corresponds to exploding gradient,
980 and the other to vanishing gradient.

981 When $\alpha < 1$ we necessarily have $\beta > 0$, hence we obtain a crude upper-bound:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L \alpha^T \left(X \sqrt{T} + \frac{1}{(1 - \alpha^2)} \left(\sqrt{\frac{XSB}{\alpha^T}} + \frac{SB}{\sqrt{(1 - \alpha)}} \right) \right) + \frac{L}{(1 - \alpha) \sqrt{1 - \alpha}} \right). \quad (74)$$

982 Once again $B = 0$ (network with no bias) leads to useful simplifications:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L \alpha^T X \sqrt{T} + \frac{L}{\sqrt{(1 - \alpha)^3}} \right). \quad (75)$$

983 This is a typical case of vanishing gradient since when $T \gg 1$ the upper bound does not depend on
984 the input scale X anymore.

985 Similarly, we can perform the analysis for $\alpha > 1$, which implies $\beta < 0$, yielding another bound:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L \frac{\alpha^T}{\alpha - 1} \left(\sqrt{T} (\alpha X + SB) + \frac{\alpha(SB + \alpha)}{\sqrt{\alpha^2 - 1}} \right) \right). \quad (76)$$

986 Without biases we get:

$$\|\nabla_{\theta} \mathcal{L}(\hat{y}, y)\|_2 = \mathcal{O} \left(L \frac{\alpha^{T+1}}{\alpha - 1} \left(\sqrt{T} X + \frac{\alpha}{\sqrt{\alpha^2 - 1}} \right) \right). \quad (77)$$

987 We recognize an exploding gradient phenomenon due to the α^T term. \square

988 E.3 Variance of the gradient

989 The result mentioned in section 3.1 is based on the following result, directly adapted from a classical
990 result on concentration inequalities.

991 **Corollary 2. Concentration of stochastic gradient around its mean.** Assume the samples (x, y)
992 are i.i.d and sampled from an arbitrary distribution \mathcal{D} . We introduce the R.V $g = \nabla_{\theta} \mathcal{L}(x, y)$ which is
993 a function of the sample (x, y) , and its expectation $\bar{g} = \mathbb{E}_{(x,y) \sim \mathcal{D}}[\nabla_{\theta} \mathcal{L}(x, y)]$. Then for all $u \geq \frac{2}{\sqrt{b}}$
994 the following inequality hold:

$$\mathbb{P}(\|\frac{1}{b} \sum_{i=1}^b g_i - \bar{g}\| > uK) \leq \exp \left(-\frac{\sqrt{b}}{8} (u - \frac{2}{\sqrt{b}})^2 \right). \quad (78)$$

995 *Proof.* The result is an immediate consequence of Example 6.3 p167 in [65]. We apply the theorem
 996 with the centered variable $X_i = \frac{1}{b}(g_i - \bar{g})$ that fulfills condition $\|X_i\| \leq \frac{c_i}{2}$ with $c_i = \frac{4K}{b}$ since
 997 $\|g_i\| \leq K$. Then for every $t \geq \frac{2K}{\sqrt{b}}$ we have:

$$\mathbb{P}(\|\frac{1}{b} \sum_{i=1}^b g_i - \bar{g}\| > t) \leq \exp\left(-\frac{\sqrt{b}}{8K^2}(t - \frac{2K}{\sqrt{b}})^2\right). \quad (79)$$

998 We conclude with the change of variables $u = \frac{t}{K}$. □

References

- [1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.
- [2] Natalia Ponomareva, Hussein Hazimeh, Alex Kurakin, Zheng Xu, Carson Denison, H Brendan McMahan, Sergei Vassilvitskii, Steve Chien, and Abhradeep Thakurta. How to dp-fy ml: A practical guide to machine learning with differential privacy. *arXiv preprint arXiv:2303.00654*, 2023.
- [3] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*, pages 265–284. Springer, 2006.
- [4] Mário S Alvim, Miguel E Andrés, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. Differential privacy: on the trade-off between utility and information leakage. In *Formal Aspects of Security and Trust: 8th International Workshop, FAST 2011, Leuven, Belgium, September 12-14, 2011. Revised Selected Papers 8*, pages 39–54. Springer, 2012.
- [5] Nicolas Papernot and Thomas Steinke. Hyperparameter tuning with renyi differential privacy. In *International Conference on Learning Representations*, 2022.
- [6] Jaewoo Lee and Daniel Kifer. Scaling up differentially private deep learning with fast per-example gradient clipping. *Proceedings on Privacy Enhancing Technologies*, 2021(1), 2021.
- [7] Xiangyi Chen, Steven Z Wu, and Mingyi Hong. Understanding gradient clipping in private sgd: A geometric perspective. *Advances in Neural Information Processing Systems*, 33:13773–13782, 2020.
- [8] Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. In *International Conference on Machine Learning*, pages 291–301. PMLR, 2019.
- [9] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.
- [10] Leon Simon et al. *Lectures on geometric measure theory*. The Australian National University, Mathematical Sciences Institute, Centre . . . , 1983.
- [11] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.
- [12] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.
- [13] Peter L Bartlett, Dylan J Foster, and Matus Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6241–6250, 2017.
- [14] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, volume 30, pages 5767–5777. Curran Associates, Inc., 2017.
- [15] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pages 854–863. PMLR, 2017.
- [16] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110:393–416, 2021.

- [17] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B Grosse, and Jörn-Henrik Jacobsen. Preventing gradient attenuation in lipschitz constrained convolutional networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, Cambridge, MA, 2019. MIT Press.
- [18] Louis Béthune, Thibaut Boissin, Mathieu Serrurier, Franck Mamalet, Corentin Friedrich, and Alberto Gonzalez Sanz. Pay attention to your loss : understanding misconceptions about lipschitz neural networks. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- [19] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.
- [20] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2008.
- [21] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.
- [22] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2015.
- [23] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [24] Lloyd N Trefethen and David Bau. *Numerical linear algebra*, volume 181. Siam, 2022.
- [25] S Singla and S Feizi. Fantastic four: Differentiable bounds on singular values of convolution layers. In *International Conference on Learning Representations (ICLR)*, 2021.
- [26] Airbus. Decomon. <https://github.com/airbus/decomon>, 2023.
- [27] Kaidi Xu, Zhouxing Shi, Huan Zhang, Yihan Wang, Kai-Wei Chang, Minlie Huang, Bhavya Kaillkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.
- [28] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.
- [29] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.
- [30] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.
- [31] Borja Balle, Gilles Barthe, and Marco Gaboardi. Privacy amplification by subsampling: Tight analyses via couplings and divergences. *Advances in Neural Information Processing Systems*, 31, 2018.
- [32] Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. Subsampled rényi differential privacy and analytical moments accountant. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1226–1235. PMLR, 2019.

- 1094 [33] Yuqing Zhu and Yu-Xiang Wang. Poission subsampled rényi differential privacy. In *International*
1095 *Conference on Machine Learning*, pages 7634–7642. PMLR, 2019.
- 1096 [34] Yuqing Zhu and Yu-Xiang Wang. Improving sparse vector technique with renyi differential
1097 privacy. *Advances in Neural Information Processing Systems*, 33:20249–20258, 2020.
- 1098 [35] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations*
1099 *symposium (CSF)*, pages 263–275. IEEE, 2017.
- 1100 [36] Ilya Mironov, Kunal Talwar, and Li Zhang. R\`enyi differential privacy of the sampled gaussian
1101 mechanism. *arXiv preprint arXiv:1908.10530*, 2019.
- 1102 [37] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent
1103 neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr,
1104 2013.
- 1105 [38] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with
1106 gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
- 1107 [39] Ugo Tanielian and Gerard Biau. Approximating lipschitz continuous functions with groupsort
1108 neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages
1109 442–450. PMLR, 2021.
- 1110 [40] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal
1111 parametrisation of recurrent neural networks using householder reflections. In *International*
1112 *Conference on Machine Learning*, pages 2401–2409. PMLR, 2017.
- 1113 [41] Shuai Li, Kui Jia, Yuxin Wen, Tongliang Liu, and Dacheng Tao. Orthogonal deep neural
1114 networks. *IEEE transactions on pattern analysis and machine intelligence*, 43(4):1352–1368,
1115 2019.
- 1116 [42] Asher Trockman and J Zico Kolter. Orthogonalizing convolutional layers with the cayley
1117 transform. In *International Conference on Learning Representations*, 2021.
- 1118 [43] Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In *International Conference on*
1119 *Machine Learning*, pages 9756–9766. PMLR, 2021.
- 1120 [44] El Mehdi Achour, François Malgouyres, and Franck Mamalet. Existence, stability and scalability
1121 of orthogonal convolutional neural networks. *The Journal of Machine Learning Research*,
1122 23(1):15743–15798, 2022.
- 1123 [45] Sahil Singla and Soheil Feizi. Improved techniques for deterministic l2 robustness. *arXiv*
1124 *preprint arXiv:2211.08453*, 2022.
- 1125 [46] Xiaojun Xu, Linyi Li, and Bo Li. Lot: Layer-wise orthogonal training on improving l2 certified
1126 robustness. *arXiv preprint arXiv:2210.11620*, 2022.
- 1127 [47] Stephen P Boyd and Lieven Vandenbergh. *Convex optimization*. Cambridge university press,
1128 2004.
- 1129 [48] Mathieu Serrurier, Franck Mamalet, Alberto González-Sanz, Thibaut Boissin, Jean-Michel
1130 Loubes, and Eustasio Del Barrio. Achieving robustness in classification using optimal transport
1131 with hinge regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision*
1132 *and Pattern Recognition*, pages 505–514, 2021.
- 1133 [49] Nicolas Papernot, Abhradeep Thakurta, Shuang Song, Steve Chien, and Úlfar Erlingsson.
1134 Tempered sigmoid activations for deep learning with differential privacy. In *Proceedings of the*
1135 *AAAI Conference on Artificial Intelligence*, volume 35, pages 9312–9321, 2021.
- 1136 [50] Florian Tramèr and Dan Boneh. Differentially private learning needs better features (or much
1137 more data), 2021.
- 1138 [51] Soham De, Leonard Berrada, Jamie Hayes, Samuel L Smith, and Borja Balle. Unlock-
1139 ing high-accuracy differentially private image classification through scale. *arXiv preprint*
1140 *arXiv:2204.13650*, 2022.

- 1141 [52] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale
1142 image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- 1143 [53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image
1144 recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*,
1145 pages 770–778, 2016.
- 1146 [54] Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas
1147 Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-
1148 mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*,
1149 34:24261–24272, 2021.
- 1150 [55] David Goldberg. What every computer scientist should know about floating-point arithmetic.
1151 *ACM computing surveys (CSUR)*, 23(1):5–48, 1991.
- 1152 [56] Hadi Jooybar, Wilson WL Fung, Mike O’Connor, Joseph Devietti, and Tor M Aamodt. Gpudet:
1153 a deterministic gpu architecture. In *Proceedings of the eighteenth international conference on*
1154 *Architectural support for programming languages and operating systems*, pages 1–12, 2013.
- 1155 [57] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas.
1156 Efficient and accurate estimation of lipschitz constants for deep neural networks. *Advances in*
1157 *Neural Information Processing Systems*, 32, 2019.
- 1158 [58] Sébastien Bubeck et al. Convex optimization: Algorithms and complexity. *Foundations and*
1159 *Trends® in Machine Learning*, 8(3-4):231–357, 2015.
- 1160 [59] Olivier Bousquet and André Elisseeff. Stability and generalization. *The Journal of Machine*
1161 *Learning Research*, 2:499–526, 2002.
- 1162 [60] Colin McDiarmid et al. On the method of bounded differences. *Surveys in combinatorics*,
1163 141(1):148–188, 1989.
- 1164 [61] Galen Andrew, Om Thakkar, Brendan McMahan, and Swaroop Ramaswamy. Differentially
1165 private learning with adaptive clipping. *Advances in Neural Information Processing Systems*,
1166 34:17455–17466, 2021.
- 1167 [62] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine
1168 learning algorithms. *Advances in neural information processing systems*, 25, 2012.
- 1169 [63] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyper-
1170 band: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine*
1171 *Learning Research*, 18(1):6765–6816, 2017.
- 1172 [64] Tom Sander, Pierre Stock, and Alexandre Sablayrolles. Tan without a burn: Scaling laws of
1173 dp-sgd. *arXiv preprint arXiv:2210.03403*, 2022.
- 1174 [65] Stéphane Boucheron, Gábor Lugosi, and Pascal Massart. *Concentration inequalities: A*
1175 *nonasymptotic theory of independence*. Oxford university press, 2013.