A Linear Algebra Facts

Fact A.1. The product $A = M_1 M_2$, where M_1 is a symmetric and positive matrix and M_2 a positive diagonal matrix, is positive definite in eigenvalues but is non-symmetric in general (unless the diagonal matrix is constant) and non-positive in quadratic forms.

Proof of Fact A.1. To see the non-symmetry of A, suppose there exists i, j such that $(M_2)_{jj} \neq (M_2)_{ii}$, then

$$(M_1 M_2)_{ij} = \sum_k (M_1)_{ik}(M_2)_{kj} = (M_1)_{ij}(M_2)_{jj} = (M_1)_{ji}(M_2)_{jj},$$
$$(M_1 M_2)_{ji} = (M_1)_{ji}(M_2)_{ii} \neq (M_1)_{ji}(M_2)_{jj}.$$

Hence A is not symmetric and positive definite. To see that A may be non-positive in the quadratic form, we give a counter-example.

$$M_1 = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix}, M_2 = \begin{pmatrix} 1 & 0 \\ 0 & 0.1 \end{pmatrix}, A = M_1 M_2 = \begin{pmatrix} 1 & 0.1 \\ 1 & 0.2 \end{pmatrix}, (1, -2) A \begin{pmatrix} 1 \\ -2 \end{pmatrix} = -0.4.$$

To see that A is positive in eigenvalues, we claim that an invertible square root $M_1^{1/2}$ exists as M_1 is symmetric and positive definite. Now A is similar to $(M_1^{1/2})^{-1} A M_1^{1/2} = M_1^{1/2} M_2 M_1^{1/2}$, hence the non-symmetric A has the same eigenvalues as the symmetric and positive definite $M_1^{1/2} M_2 M_1^{1/2}$. □

Fact A.2. Matrix with all eigenvalues positive may be non-positive in quadratic form.

Proof of Fact A.2.

$$A = \begin{pmatrix} -1 & 3 \\ -3 & 8 \end{pmatrix}, (1, 0) A \begin{pmatrix} 1 \\ 0 \end{pmatrix} = -1,$$

though eigenvalues of A are $\frac{1}{2}(7 \pm 3\sqrt{5}) > 0$. □

Fact A.3. Matrix with positive quadratic forms may have non-positive eigenvalues.

Proof of Fact A.3.

$$A = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, (x, y) A \begin{pmatrix} x \\ y \end{pmatrix} = x^2 + y^2 > 0,$$

but eigenvalues of A are $1 \pm i$, not positive nor real. Actually, all eigenvalues of A always have positive real part. □

Fact A.4. Sum of products of positive definite (symmetric) matrix and positive diagonal matrix may have zero or negative eigenvalues.

Proof of Fact A.4.

$$\mathbf{H}_1 = \begin{pmatrix} 8/9 & 2 \\ 2 & 7 \end{pmatrix}, \quad \mathbf{C}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.4 \end{pmatrix}, \quad \mathbf{H}_2 = \begin{pmatrix} 3 & 2 \\ 2 & 2 \end{pmatrix}, \quad \mathbf{C}_2 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.6 \end{pmatrix}.$$

Although $\mathbf{H}_j$ are positive definite, $\mathbf{H}_1 \mathbf{C}_1 + \mathbf{H}_2 \mathbf{C}_2$ has a zero eigenvalue. Further, if $\mathbf{H}_1[1, 1] = 0.7$, then $\mathbf{H}_1 \mathbf{C}_1 + \mathbf{H}_2 \mathbf{C}_2$ has a negative eigenvalue. □

B Details of Main Results

B.1 Proofs of main results

Derivation of the loss dynamic of layer-wise clipping method. Recall the layer-wise clipping defined in Section 4.2. The gradient flow corresponds to the DP-GD with layerwise clipping is:

$$\mathbf{w}_r(k+1) = \mathbf{w}_r(k) - \frac{\eta}{n}\left(\sum\nolimits_i \nabla_{\mathbf{w}_r}\ell_i C_{i,r} + \sigma R_r \cdot \mathcal{N}(0,1)\right) \quad \text{and } \dot{\mathbf{w}}_r(t) = -\frac{1}{n}\sum_i \nabla_{\mathbf{w}_r}\ell_i C_{i,r}.$$

Then, we have

$$\dot{L} = \sum_r \frac{\partial L}{\partial \mathbf{w}_r}\dot{\mathbf{w}}_r = -\sum_r \frac{\partial L}{\partial \boldsymbol{f}}\mathbf{H}_r\mathbf{C}_r\frac{\partial L}{\partial \boldsymbol{f}}^\top. \tag{B.1}$$

As we can see from (4.3) and (B.1), the per-sample clipping precisely changes the NTK matrix from $\mathbf{H} \equiv \sum_r \mathbf{H}_r$, in standard non-DP deep learning, to $\mathbf{HC}$ in DP training with flat clipping, and to $\sum_r \mathbf{H}_r\mathbf{C}_r$ in DP training with layerwise clipping. Subsequently, we will show that this may break the NTK's positivity and worsen the convergence of DP training than the non-DP one.

Proof of Fact 4.1. Expanding the discrete dynamic in (4.1) as $\mathbf{w}(k+1) = \mathbf{w}(k) - \frac{\eta}{n}\sum_i \nabla_{\mathbf{w}}\ell_i C_i - \frac{\eta\sigma R}{n}\mathcal{N}(0,1)$, and chaining it for $r \geq 1$ times, we obtain

$$\mathbf{w}(k+r) - \mathbf{w}(k) = -\sum_{j=0}^{r-1}\frac{\eta}{n}\sum_i \nabla_{\mathbf{w}}\ell_i(\mathbf{w}(k+j))C_i - \sum_{j=0}^{r-1}\frac{\eta\sigma R}{n}\mathcal{N}(0,1).$$

In the limit of $\eta \to 0$, we re-index the weights $\mathbf{w}$ by time, with $t = k\eta$ and $s = r\eta$. Then the left hand side becomes $\mathbf{w}(t+s) - \mathbf{w}(t)$; the first summation on the right hand side converges to $-\frac{1}{n}\int_t^{t+s}\sum_i \nabla_{\mathbf{w}}\ell_i(\tau)C_i(\tau)d\tau$, as long as the integral exists, and the second summation $J(\eta) = \sum_{j=0}^{r-1}\frac{\eta\sigma R}{n}\mathcal{N}(0,1)$ has

$$\mathbb{E}[J(\eta)] = 0 \quad \text{and} \quad \mathrm{Var}(J(\eta)) = \frac{\sigma^2R^2\eta^2}{n^2}r = \eta s\frac{\sigma^2R^2}{n^2} \to 0, \text{ as } \eta \to 0.$$

Therefore, as $\eta \to 0$, the discrete stochastic dynamic (4.1) converges to a deterministic gradient flow given by the integral

$$\mathbf{w}(t) - \mathbf{w}(0) = -\frac{1}{n}\int_0^t \sum\nolimits_i \nabla_{\mathbf{w}}\ell_i(\tau)C_i(\tau)d\tau,$$

which corresponds to the ordinary differential equations (4.2). □

Proof of Theorem 1. We prove the statements using the derived gradient flow dynamics (4.2).

For Statement 1, from our narrative in Section 4.2 and Table 2, we know that the local flat clipping algorithm has $\mathbf{H}(t)\mathbf{C}(t)$ as its NTK. Since $\mathbf{H}(t)$ is positive definite and $\mathbf{C}(t)$ is a positive diagonal matrix, by Fact A.1, the product $\mathbf{H}(t)\mathbf{C}(t)$ is positive in eigenvalues, yet may be asymmetric and not positive in quadratic form in general.

Similarly, for Statement 2, we know the NTK of local layerwise clipping has the form $\sum_r \mathbf{H}_r(t)\mathbf{C}_r(t)$, which by Fact A.4 is asymmetric in general, and may be not positive in quadratic form nor positive in eigenvalues.

For Statement 3, by the training dynamics (4.3) for the local flat clipping algorithm and (B.1) for the local layerwise clipping, we see that $\dot{L}$ equal the negation of a quadratic form of the corresponding NTK. By statement 1 & 2 of this theorem, such quadratic form may not be positive at all t, and hence the loss $L(t)$ is not guaranteed to decrease monotonically.

Lastly, for Statement 4, suppose $L(t)$ converges, i.e. $\dot{L} = 0 = \frac{\partial L}{\partial \boldsymbol{f}} \dot{\boldsymbol{f}}$. Suppose we have $L > 0$, then $\frac{\partial L}{\partial \boldsymbol{f}} \neq 0$ since L is convex in the prediction $\boldsymbol{f}$. In this case, we know $\dot{\boldsymbol{f}} = 0$. Observe that

$$0 = \dot{\boldsymbol{f}} = \frac{\partial \boldsymbol{f}}{\partial \mathbf{w}} \frac{\partial \mathbf{w}}{\partial t} = -\frac{\partial \boldsymbol{f}}{\partial \mathbf{w}} \frac{\partial \boldsymbol{f}}{\partial \mathbf{w}}^\top \frac{\partial L}{\partial \boldsymbol{f}}^\top .$$

For the local flat clipping, the NTK matrix, $\frac{\partial \boldsymbol{f}}{\partial \mathbf{w}} \frac{\partial \boldsymbol{f}}{\partial \mathbf{w}}^\top = \mathbf{HC}$ is positive in eigenvalues (by Statement 1), so it could only be the case that $\frac{\partial L}{\partial \boldsymbol{f}} = \mathbf{0}$, contradicting to our premise that $L > 0$. Therefore we know $L = 0$ as long as it converges for the local flat clipping. On the other hand, for the local layerwise clipping, the NTK may be not positive in eigenvalues. Hence it is possible that $L \neq 0$ when $\dot{L} = 0$.

□

Proof of Theorem 2. The proof is similar to the previous proof, we consider the gradient flow dynamic for global clipping as follows:

For the first statement, we note that the NTKs for global clipping are obviously symmetric: since C and C_r are scalars, we have $(\mathbf{H}(t)C(t))^\top = \mathbf{H}(t)C(t)$, and $(\sum_r \mathbf{H}_r(t)C_r(t))^\top = \sum_r \mathbf{H}_r(t)C_r(t)$, are all symmetric matrices. Also, for all $x \neq 0$, $x^\top x \mathbf{H} x = C \cdot x^\top \mathbf{H} x > 0$, so the symmetric matrix $\mathbf{H}C$ is positive definite. Similar argument can easily show that the summation of symmetric positive matrices is also a positive definite matrix.

Now, to prove the second statement, we note that for both global flat clipping and global layerwise clipping, (4.3) and (B.1) give $\dot{L}(t) < 0$ since the NTKs are positive in quadratic form. That means L decreases monotonically. Additionally, L is bounded below by zero. Therefore L must converge and thus $\dot{L} = 0$. Note that when we have $\frac{\partial L}{\partial \boldsymbol{f}} = \mathbf{0}$, it implies all $\ell_i = 0$, and thus $L = 0$. □

To prove Theorem 3, we citep the known result of Gaussian Mechanism as follows.

Lemma B.1 (Theorem A.1 (Dwork et al., 2014); Theorem 2.7 (Dong et al., 2019))**.** *Define the ℓ_2 **sensitivity** of any function g to be $\Delta g = \sup_{S,S'} \|g(S) - g(S')\|_2$ where the supreme is over all neighboring (S, S'). Then the **Gaussian mechanism** $\hat{g}(S) = g(S) + \sigma \Delta g \cdot \mathcal{N}(0, \mathbf{I})$ is (ϵ, δ)-DP for some ϵ depending on (σ, n, p, δ).*

Proof of Theorem 3. Under local or global clipping in Algorithm 1, each clipped gradient $\bar{v}_t^{(i)}$ has a norm bounded by R. Therefore, both clippings have the same sensitivity of $\bar{V}_t = \sum_{i \in I_t} \bar{v}_t^{(i)}$ and hence the same privacy risk, regardless which privacy accountant is adopted. □

C Layerwise Per-Sample Clipping

We elaborate the details of layerwise clipping in this section. We describe the layerwise clipping algorithm for DP-SGD, in complement to Algorithm 1 (not an generalization, i.e. flat clipping is not a subset of layerwise clipping). For other optimizers the extension to layerwise clipping is similar. Assume the neural network has d layers, denote the weights of the r-th layer as $\mathbf{w}_r$, then the layerwise clipping can clip the per-sample gradient of each layer either locally or globally.

Algorithm 2 `DP-SGD` (with local or global layerwise per-sample clipping)

Input: Dataset $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, loss function $\ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$.
Parameters: initial weights $\mathbf{w}_0$, learning rate η_t, subsampling probability p, number of iterations T, noise scale σ, gradient norm bound R_r, maximum norm bound Z_r for each layer $1 \leq r \leq d$.

for $t = 0, \ldots, T-1$ **do**
  Take a subsample $I_t \subseteq \{1, \ldots, n\}$ from training set D with subsampling probability p
  for $r = 1, ..., d$ **do**
    for $i \in I_t$ **do**
      $v_{r,t}^{(i)} \leftarrow \nabla_{\mathbf{w}_r} \ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$
      Option 1: $C_{local,(i,r)} = \min\left\{1, R_r / \|v_{r,t}^{(i)}\|_2\right\}$ ▷ Local clipping factor
      Option 2: $C_{global,(i,r)} \equiv \begin{cases} R_r/Z_r & \text{if } \|v_{r,t}^{(i)}\|_2 \leq Z_r \\ 0 & \text{if } \|v_{r,t}^{(i)}\|_2 > Z_r \end{cases}$ ▷ Global clipping factor
      $\bar{v}_{r,t}^{(i)} \leftarrow C_{i,r} \cdot v_{r,t}^{(i)}$ ▷ Clip the gradient
    $\bar{V}_{r,t} \leftarrow \sum_{i \in I_t} \bar{v}_{r,t}^{(i)}$ ▷ Sum over batch
    $\widetilde{V}_{r,t} \leftarrow \bar{V}_{r,t} + \sigma R_r \cdot \mathcal{N}(0, I)$ ▷ Apply Gaussian mechanism
    $\mathbf{w}_{r,t+1} \leftarrow \mathbf{w}_{r,t} - \frac{\eta_t}{|I_t|} \widetilde{V}_{r,t}$ ▷ Descend
  Output $\mathbf{w}_{r,T}$

For implementation, one can set `max_grad_norm` as a list of scalars in the `Opacus` PrivacyEngine[11].

D Code Implementation

Building on top of the Pytorch `Opacus`[12] library, we only need to add one line of code into

`https://github.com/pytorch/opacus/blob/master/opacus/per_sample_gradient_clip.py`

To understand our implementation, we can equivalently view Option 2 in Algorithm 1 as

$$C_{global,i} = \begin{cases} R/Z & \text{if } C_{local,i} \geq R/Z \\ 0 & \text{if } C_{local,i} < R/Z \end{cases}$$

In this formulation, we can easily implement our global clipping by leveraging the `Opacus` library (which already computes $C_{local,i}$). This can be realized in multiple ways.

For example, we can add the following one line after line 179 (within the for loop),

```
import config;clip_factor=torch.where(clip_factor > self.norm_clipper.thresholds[0]/config.Z,
torch.ones_like(clip_factor)*self.norm_clipper.thresholds[0]/config.Z,
torch.zeros_like(clip_factor))
```

Here we use the package `config` to pass global variable, the maximum norm bound Z. An equivalent but easier-to-follow version is

```
import config;
R=self.norm_clipper.thresholds[0]
Z=config.Z
clip_factor=torch.where(clip_factor > R/Z,
torch.ones_like(clip_factor)*R/Z,
torch.zeros_like(clip_factor))
```

[11] See `https://github.com/pytorch/opacus/blob/e9983eced87619f683d84861c6503aca4e9287d1/opacus/privacy_engine.py`

[12] see `https://github.com/pytorch/opacus` as for 2021/09/09.

Comparing to the original PyTorch implementation, our code only computes an additional thresholding over B (batch size) values, the extra computational complexity is negligible.

E Experimental Details

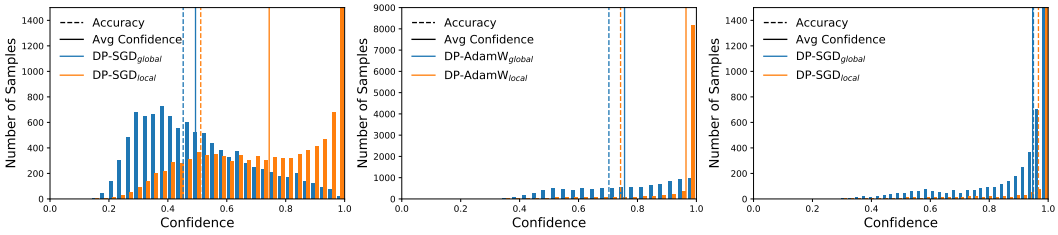

Figure 10: Confidence histograms on CIFAR 10 (left), SNLI (middle), and MNIST (right).

E.1 MNIST

For MNIST, we use the standard CNN in `Tensorflow Privacy` and `Opacus`, as listed below. For both global and local clippings, the training hyperparameters (e.g. batch size) in Section 6.1 are exactly the same as reported in `https://github.com/tensorflow/privacy/tree/master/tutorials`, which gives 96.6% accuracy for the local clipping in Tensorflow and similar accuracy in Pytorch, where our experiments are conducted. The non-DP network is about 99% accurate. Notice the tutorial uses a different privacy accountant than the GDP that we used.

```
class SampleConvNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 16, 8, 2, padding=3)
        self.conv2 = nn.Conv2d(16, 32, 4, 2)
        self.fc1 = nn.Linear(32 * 4 * 4, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        # x of shape [B, 1, 28, 28]
        x = F.relu(self.conv1(x))  # -> [B, 16, 14, 14]
        x = F.max_pool2d(x, 2, 1)  # -> [B, 16, 13, 13]
        x = F.relu(self.conv2(x))  # -> [B, 32, 5, 5]
        x = F.max_pool2d(x, 2, 1)  # -> [B, 32, 4, 4]
        x = x.view(-1, 32 * 4 * 4)  # -> [B, 512]
        x = F.relu(self.fc1(x))  # -> [B, 32]
        x = self.fc2(x)  # -> [B, 10]
        return x
```

E.2 CIFAR10 with 5-layer CNN

In Section 6.2, we adopt the model from Pytorch tutorial in `https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`, which is the following 5-layer CNN.

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
```

```
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

In addition to Figure 10 and Figure 6, we plot in Figure 11 the distribution of prediction probability on the true class, say $[\boldsymbol{\pi}_i]_{y_i}$ for the i-th sample (notice that Figure 10 plots $\max_k[\boldsymbol{\pi}_i]_k$). Clearly the local clipping gives overly confident prediction: almost half of the time the true class is assigned close to zero prediction probability. The global clipping has a much more balanced prediction probability.

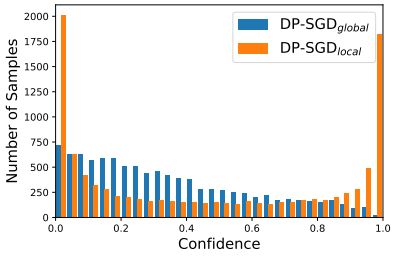

Figure 11: Prediction probability on the true class on CIFAR10 with 5-layer CNN.

E.3 NLP: SNLI with BERT model

In Section 6.3, we use the model from `Opacus` tutorial in `https://github.com/pytorch/opacus/blob/master/tutorials/building_text_classifier.ipynb`. The BERT architecture can be found in `https://github.com/pytorch/opacus/blob/master/tutorials/img/BERT.png`.

To train the BERT model, we do the standard pre-processing on the corpus (tokenize the input, cut or pad each sequence to MAX_LENGTH = 128, and convert tokens into unique IDs). We train the BERT model for 3 epochs. Similar to Appendix E.2, in addition to Figure 7 and Figure 8, we plot the distribution of prediction probability on the true class in Figure 12. Again, the local clipping is overly confident, with probability masses concentrating on the two extremes, yet the global clipping is more balanced in assigning the prediction probability.

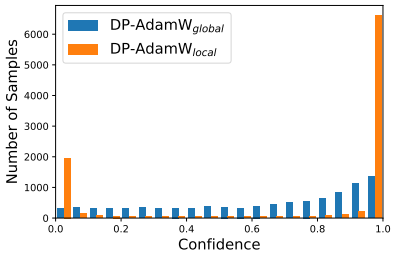

Figure 12: Prediction probability on the true class on SNLI with BERT.

E.4 Regression Experiments

We experiment on the Wine Quality[13] (1279 training samples, 320 test samples, 11 features) and California Housing[14] (18576 training samples, 2064 test samples, 8 features) datasets in Section 6. For the California Housing, we use DP-Adam with batch size 256. Since other datasets are not large, we use the full-batch DP-GD.

Across all the two experiments, we set $\delta = \frac{1}{1.1 \times \text{training sample size}}$ and use the four-layer neural network with the following structure, where `input_width` is the input dimension for each dataset:

```
class Net(nn.Module):
    def __init__(self, input_width):
        super(StandardNet, self).__init__()
        self.fc1 = nn.Linear(input_width, 64, bias = True)
        self.fc2 = nn.Linear(64, 64, bias = True)
        self.fc3 = nn.Linear(64, 32, bias = True)
        self.fc4 = nn.Linear(32, 1, bias = True)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        return self.fc4(x)
```

The California Housing dataset is used to predict the mean price value of owner-occupied home in California. We train both global flat and local flat clipping with DP-Adam, both with noise $\sigma = 1$, clipping norm 1, and learning rate 0.0002. We also trained a non-DP GD with the same learning rate. The GDP accountant gives $\epsilon = 4.41$ after 50 epochs / 3650 iterations.

The UCI Wine Quality (red wine) dataset is used to predict the wine quality (an integer score between 0 and 10). We train both global flat and local flat clipping with DP-GD, both with noise $\sigma = 35$, clipping norm 2, and learning rate 0.03. We also trained a non-DP GD with learning rate 0.001. The GDP accountant gives $\epsilon = 4.40$ after 2000 iterations.

The California Housing and Wine Quality experiments are conducted in 30 independent runs. In Figure 9, the lines are the average losses and the shaded regions are the standard deviations.

F Optimizers with Clipping beyond Gradient Descent

We can extend Theorem 1 and Theorem 2 to a wide class of full-batch optimizers besides DP-GD (with $\sigma = 0$ and $\sigma \neq 0$). We show that the NTK matrices in these optimizers determine whether the loss is zero if the model converges.

Theorem 4. *For an arbitrary neural network and a loss convex in f, suppose we clip the per-sample gradients in the gradient flow of Heavy Ball (HB), Nesterov Accelerated Gradient (NAG), Adam, AdaGrad, RMSprop or their DP variants and that $\|v_t^{(i)}\|_2 \leq Z$, assuming $\mathbf{H}(t) \succ 0$, then*

1. *if the loss $L(t)$ converges, it must converge to 0 for local flat, global flat and global layerwise clipping;*
2. *even if the loss $L(t)$ converges, it may converge to non-zero for local layerwise clipping.*

The proof can be easily extracted from that of Theorem 1 and Theorem 2 and hence is omitted. We highlight that DP optimizers in general correspond to deterministic gradient

[13] `http://archive.672ics.uci.edu/ml/datasets/Wine+Quality`

[14] `http://lib.stat.cmu.edu/datasets/houses.zip`

flow (for DP-GD, see Fact 4.1) – as long as the noise injected in each step is linear in step size. Therefore, the gradient flow is the same whether $\sigma > 0$ (the noisy case) or $\sigma = 0$ (the noiseless case).

We also note that the only difference between layerwise clipping and flat clipping is the form of NTK kernel, as we showed in Theorem 1 and Theorem 2. In this section, we will only present the result for flat clipping since its generalization to layerwise clipping is straightforward. In fact, part of the results for the global clipping has been implied by (Bu et al., 2021b), which establishes the error dynamics for HB and NAG, but only on MSE loss and on specific network architecture. To analyze a broader class of optimizers and on the general loss and architecture, we turn to (da Silva & Gazeau, 2020) which gives the dynamical systems of all optimizers aforementioned.

F.1 Gradient Methods with Momentum

We study two commonly used momentums, the Heavy Ball (Polyak, 1964) and the Nesterov's one (Nesterov, 1983). These gradient methods correspond to the gradient flow system (da Silva & Gazeau, 2020, Equation (2.1))

$$\dot{\mathbf{w}}(t) = -\mathbf{m}(t), \tag{F.1}$$

$$\dot{\mathbf{m}}(t) = \sum_i \nabla_{\mathbf{w}} \ell_i C_i - r(t)\mathbf{m}(t). \tag{F.2}$$

We note that HB corresponds to time-independent $r(t) = r$ for some r and NAG corresponds to $r(t) = 3/t$. At the stationary point, we have $\dot{L} = \dot{\mathbf{w}} = \dot{\mathbf{m}} = 0$. Consequently (F.1) gives $\mathbf{m} = \mathbf{0}$ and (F.2) gives

$$\sum_i \nabla_{\mathbf{w}} \ell_i C_i = r\mathbf{m} = \mathbf{0}. \tag{F.3}$$

Multiplying both sides with $\frac{\partial \boldsymbol{f}}{\partial \mathbf{w}}$, we get

$$\mathbf{H}\mathbf{C}\frac{\partial L}{\partial \boldsymbol{f}} = \mathbf{0},$$

where $\frac{\partial L}{\partial \boldsymbol{f}}$ is defined in (4.3). If the NTK is positive in eigenvalues, as is the case for local flat and global clipping, we get $\frac{\partial L}{\partial \boldsymbol{f}} = \mathbf{0}$ and $\ell_i = 0$ for all i since the loss is convex (thus the only stationary point is the global minimum 0). Hence $L = 0$. Otherwise, e.g. for local layerwise clipping, it is possible that $\frac{\partial L}{\partial \boldsymbol{f}}^\top \neq \mathbf{0}$ and $L \neq 0$.

F.2 Adaptive Gradient Methods with Momentum

We consider Adam which corresponds to the dynamical system in (da Silva & Gazeau, 2020, Equation (2.1))

$$\dot{\mathbf{w}}(t) = -\mathbf{m}(t)/\sqrt{\mathbf{v}(t) + \xi}, \tag{F.4}$$

$$\dot{\mathbf{m}}(t) = \sum_i \nabla_{\mathbf{w}} \ell_i C_i - \frac{1}{\alpha_1}\mathbf{m}(t), \tag{F.5}$$

$$\dot{\mathbf{v}}(t) = \frac{1}{\alpha_2}\left[\sum_i \nabla_{\mathbf{w}} \ell_i C_i\right]^2 - \frac{1}{\alpha_2}\mathbf{v}(t). \tag{F.6}$$

Here $\xi \geq 0$ and the square is taken elementwise. At the stationary point, we have $\dot{L} = \dot{\mathbf{w}} = \dot{\mathbf{m}} = \dot{\mathbf{v}} = 0$. Consequently (F.4) gives $\mathbf{m} = \mathbf{0}$ and (F.5) gives $\sum_i \nabla_{\mathbf{w}} \ell_i C_i = \mathbf{m}/\alpha_1 = \mathbf{0}$. Multiplying both sides with $\frac{\partial \boldsymbol{f}}{\partial \mathbf{w}}$, we get again $\mathbf{H}\mathbf{C}\frac{\partial L}{\partial \boldsymbol{f}} = \mathbf{0}$, and hence the results follow.

F.3 Adaptive Gradient Methods without Momentum

We consider ADAGRAD and RMSprop which correspond to the dynamical system in (da Silva & Gazeau, 2020, Remark 1)

$$\dot{\mathbf{w}}(t) = -\sum_i \nabla_{\mathbf{w}} \ell_i C_i / \sqrt{\mathbf{v}(t) + \xi}, \tag{F.7}$$

$$\dot{\mathbf{v}}(t) = p(t) \left[\sum_i \nabla_{\mathbf{w}} \ell_i C_i \right]^2 - q(t)\mathbf{v}(t), \tag{F.8}$$

for some $p(t), q(t)$. At the stationary point, we have $\dot{L} = \dot{\mathbf{w}} = \dot{\mathbf{m}} = \dot{\mathbf{v}} = 0$. Consequently (F.7) gives $\sum_i \nabla_{\mathbf{w}} \ell_i C_i = \mathbf{0}$. Multiplying both sides with $\frac{\partial \boldsymbol{f}}{\partial \mathbf{w}}$, we get again $\mathbf{H}\mathbf{C}\frac{\partial L}{\partial \boldsymbol{f}}^\top = 0$, and hence the results follow.

F.4 Applying Global Clipping to DP Optimization Algorithms

Here we give some concrete algorithms where we can apply the global clipping method.

Many DP optimizers, non-adaptive (like HeavyBall and Nesterov Accelerated Gradient) and adaptive (like Adam, ADAGRAD), can use the global clipping easily. These optimizers are supported in `Opacus` and `Tensorflow Privacy` libraries. The original form of DP-Adam can be found in (Bu et al., 2019).

Algorithm 3 `DP-Adam` (with local or global per-sample clipping)

Input: Dataset $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, loss function $\ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$.
Parameters: initial weights $\mathbf{w}_0$, learning rate η_t, subsampling probability p, number of iterations T, noise scale σ, gradient norm bound R, maximum norm bound Z, momentum parameters (β_1, β_2), initial momentum m_0, initial past squared gradient u_0, and a small constant $\xi > 0$.

for $t = 0, \ldots, T-1$ **do**
  Take a subsample $I_t \subseteq \{1, \ldots, n\}$ from training set D with subsampling probability p
  for $i \in I_t$ **do**
    $v_t^{(i)} \leftarrow \nabla_{\mathbf{w}} \ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$
    Option 1: $C_{local,i} = \min\left\{1, R/\|v_t^{(i)}\|_2\right\}$ ▷ Local clipping factor
    Option 2: $C_{global,i} \equiv \begin{cases} R/Z & \text{if } \|v_t^{(i)}\|_2 \le Z \\ 0 & \text{if } \|v_t^{(i)}\|_2 > Z \end{cases}$ ▷ Global clipping factor
    $\bar{v}_t^{(i)} \leftarrow C_i \cdot v_t^{(i)}$ ▷ Clip the gradient
  $\widetilde{V}_t \leftarrow \frac{1}{|I_t|}\left(\sum_{i \in I_t} \bar{v}_t^{(i)} + \sigma R \cdot \mathcal{N}(0, I)\right)$ ▷ Apply Gaussian mechanism
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1)\widetilde{V}_t$
  $u_t \leftarrow \beta_2 u_{t-1} + (1-\beta_2)(\widetilde{V}_t \odot \widetilde{V}_t)$ ▷ $\odot$ is the Hadamard product
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t m_t / (\sqrt{u_t} + \xi)$ ▷ Descend
Output $\mathbf{w}_T$

Recently, (Bu et al., 2021a) proposes to accelerate many DP optimizers with the JL projections in a memory efficient manner. Examples include DP-SGD-JL and DP-Adam-JL. The acceleration is achieved by only approximately instead of exactly computing the per-sample gradient norms. This does not affect the clipping operation afterwards and hence we can replace the local clipping currently used by our global clipping.

Algorithm 4 `DP-SGD-JL` (with local or global per-sample clipping)

Input: Dataset $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, loss function $\ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$.
Parameters: initial weights $\mathbf{w}_0$, learning rate η_t, subsampling probability p, number of iterations T, noise scale σ, gradient norm bound R, maximum norm bound Z, number of JL projections r.

for $t = 0, \ldots, T-1$ **do**
  Take a subsample $I_t \subseteq \{1, \ldots, n\}$ from training set D with subsampling probability p
  Sample $u_1, ..., u_r \sim \mathcal{N}(0, I)$
  for $i \in I_t$ **do**
    $v_t^{(i)} \leftarrow \nabla_{\mathbf{w}} \ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$
    for $j = 1$ to r **do**
      $P_{ij} \leftarrow v_t^{(i)} \cdot u_j$ (using `jvp`)
    $M_i = \sqrt{\frac{1}{r} \sum_{j=1}^{r} P_{ij}^2}$ ▷ M_i is an estimate for $\|v_t^{(i)}\|_2$.
    Option 1: $C_{local,i} = \min\left\{1, R/M_i\right\}$ ▷ Local clipping factor
    Option 2: $C_{global,i} \equiv \begin{cases} R/Z & \text{if } M_i \leq Z \\ 0 & \text{if } M_i > Z \end{cases}$
    $\bar{v}_t^{(i)} \leftarrow C_i \cdot v_t^{(i)}$ ▷ Clip the gradient
  $\bar{V} \leftarrow \sum_{i \in I_t} \bar{v}_t^{(i)}$ ▷ Sum over batch
  $\widetilde{V}_t \leftarrow \bar{V}_t + \sigma R \cdot \mathcal{N}(0, I)$ ▷ Apply Gaussian mechanism
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{|I_t|} \widetilde{V}_t$ ▷ Descend

Output $\mathbf{w}_T$

In another line of research on the Bayesian neural networks, where the reliability of networks are emphasized, stochastic gradient Markov chain Monte Carlo (SG-MCMC) methods are applied to quantify the uncertainty of the weights. When DP is within the scope, one popular method is the DP stochastic gradient Langevin dynamics (DP-SGLD), where we can apply the global clipping.

Algorithm 5 `DP-SGLD` (with local or global per-sample clipping)

Input: Dataset $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, loss function $\ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$.
Parameters: initial weights $\mathbf{w}_0$, learning rate η_t, subsampling probability p, number of iterations T, gradient norm bound R, maximum norm bound Z, and a prior $p(\mathbf{w})$.

for $t = 0, \ldots, T-1$ **do**
  Take a subsample $I_t \subseteq \{1, \ldots, n\}$ from training set D with subsampling probability p
  for $i \in I_t$ **do**
    $v_t^{(i)} \leftarrow \nabla_{\mathbf{w}} \ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$
    Option 1: $C_{local,i} = \min\left\{1, R/\|v_t^{(i)}\|_2\right\}$ ▷ Local clipping factor
    Option 2: $C_{global,i} \equiv \begin{cases} R/Z & \text{if } \|v_t^{(i)}\|_2 \leq Z \\ 0 & \text{if } \|v_t^{(i)}\|_2 > Z \end{cases}$ ▷ Global clipping factor
    $\bar{v}_t^{(i)} \leftarrow C_i \cdot v_t^{(i)}$ ▷ Clip the gradient
  $\bar{V} \leftarrow \sum_{i \in I_t} \bar{v}_t^{(i)}$ ▷ Sum over batch
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \left(\frac{\bar{V}_t}{|I_t|} - \frac{\nabla_{\mathbf{w}} \log p(\mathbf{w})}{n} \right) + \mathcal{N}(0, \eta_t I)$ ▷ Descend with Gaussian noise

Output $\mathbf{w}_T$

Here we treat $\mathbf{w}_{t+1}$ as a posterior sample, instead of as a point estimate. Notice that other SG-MCMC methods such as SGNHT (Ding et al., 2014) can also be DP with the global per-sample clipping.

We emphasize that our global clipping applies whenever an optimization algorithm uses per-sample clipping. Therefore this appendix only gives a few example of the full capacity of global clipping.

F.5 Applying Global Clipping to DP Federated Learning

Here we present two federated learning methods in (McMahan et al., 2017): `DP-FedSGD` and `DP-FedAvg` with the global or local clipping. Notice that we only demonstrate the flat clippings and SGD. Layerwise clippings can be easily implemented by changing the ClipFn, and the optimizer can be replaced by other ones.

Main training loop:

parameters
  user selection probability $q \in (0, 1]$
  number of examples per-user w_k
  gradient norm bound R
  maximum norm bound Z
  noise scale σ
  UserUpdate (for FedAvg or FedSGD)
  ClipFn (LocalClip or GlobalClip)

Initialize model θ^0, $W = \sum_k w_k$
for each round $t = 0, 1, 2, \dots$ **do**
  $\mathcal{C}^t \leftarrow$ (sample users with probability q)
  for each user $k \in \mathcal{C}^t$ **in parallel do**
    $\Delta_k^{t+1} \leftarrow \text{UserUpdate}(k, \theta^t, \text{ClipFn})$
  $\Delta^{t+1} = \frac{\sum_{k \in \mathcal{C}^t} w_k \Delta_k}{qW}$
  $\theta^{t+1} \leftarrow \theta^t + \Delta^{t+1} + \frac{\sigma}{qW} R\mathcal{N}(0, I)$

FlatClip(Δ):
  return $\Delta \cdot \min\{1, R/\|\Delta\|_2\}$
GlobalClip(Δ):
  if $\Delta > Z$:
    return 0
  else:
    return $\Delta \cdot R/Z$
UserUpdateFedAvg(k, θ^0, ClipFn):
  parameters B, E, η
  $\theta \leftarrow \theta^0$
  for each local epoch i from 1 to E **do**
    $\mathcal{B} \leftarrow$ (k's data split into size B batches)
    for batch $b \in \mathcal{B}$ **do**
      $\theta \leftarrow \theta - \eta \nabla \ell(\theta; b)$
      $\theta \leftarrow \theta^0 + \text{ClipFn}(\theta - \theta^0)$
  return update $\Delta_k = \theta - \theta^0$
UserUpdateFedSGD(k, θ^0, ClipFn):
  parameters B, η
  select a batch b of size B from k's examples
  return update $\Delta_k = \text{ClipFn}(-\eta \nabla \ell(\theta; b))$

Algorithm 6: `DP-FedAvg` and `DP-FedSGD` with global or local clipping.

F.6 Comparison between Different Clippings

Here we give a brief comparison between different clippings: the local per-sample clipping, the global per-sample clipping and the non-DP batch clipping (see Algorithm 7). In the

Algorithm 7 `Non-DP SGD (with batch clipping)`

for $t = 0, \dots, T-1$ **do**
  Take a subsample $I_t \subseteq \{1, \dots, n\}$ from training set D with subsampling probability p
  for $i \in I_t$ **do**
    $v_t^{(i)} \leftarrow \nabla_{\mathbf{w}} \ell(f(\boldsymbol{x}_i, \mathbf{w}_t), y_i)$
  $V_t \leftarrow \frac{1}{|I_t|} \sum_{i \in I_t} v_t^{(i)}$ ▷ Sum over batch
  $\bar{V}_t \leftarrow V_t \cdot \min\left\{1, R'/\|V_t\|_2\right\}$ ▷ Clip the gradient
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \bar{V}_t$ ▷ Descend
Output $\mathbf{w}_T$

example of SGD, the key difference between the non-DP clipping and the DP clippings is the order of operations: the non-DP clipping first average the per-sample gradients, then clips in a batch manner. However, for DP clippings in Algorithm 1, we first clip the per-sample gradients (the local clipping works in a per-sample manner but the global clipping works in a batch manner) then take the average of the clipped gradients. Informally, we distinguish the gradients after different clippins as follows:

- **non-DP batch clipping:** $\bar{V}_t = \text{ave}(v_t^{(i)}) \cdot c' = \text{ave}(v_t^{(i)} \cdot c')$;

- **DP local per-sample clipping:** $\bar{V}_t = \text{ave}(v_t^{(i)} \cdot C_i)$;
- **DP global per-sample clipping:** $\bar{V}_t = \text{ave}(v_t^{(i)} \cdot C)$;

where $0 < c', C, C_i \leq 1$ are clipping factors and 'ave' is the average. Notice that $C = R/Z$ and hence even though the global clipping is performing a batch clipping, it still requires the $|I_t|$ per-sample clipping factors (which needs to compute the per-sample gradients and their norms) and hence is different from non-DP batch clipping. See Table 4.

	non-DP clipping	DP local clipping	DP global clipping
Need per-sample gradient	No	Yes	Yes
Batch clipping	Yes	No	Yes

Table 4: Comparison of different clippings with respect to whether per-sample gradient information is needed and whether the operation can be applied on the batch as a whole.

G The effect of noise on DP training

We demonstrate the following toy experiment that shares spirit of Fact 4.1. We would like to compare the performance of DP-SGD algorithms with various level of noise. Notably as in Figure 13, within a moderate range of noise addition, the performance changes little; but the removal of the clipping changes the performance dramatically.

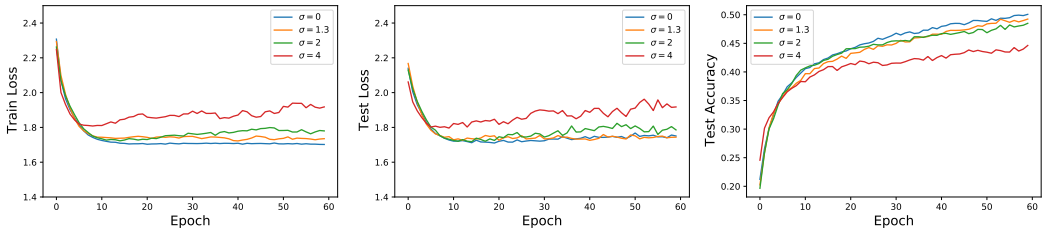

Figure 13: Performance of DP-SGD$_{local}$ on CIFAR10 with various σ. All other parameters are as in Section 6.2. We note that with $\sigma = 0$ and no clipping, test accuracy is about 62%.