

## 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 may be 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

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

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  $M_1^{1/2} M_2 M_1^{1/2}$ .  $\square$

**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  $\frac{1}{2}(7 \pm 3\sqrt{5}) > 0$ .  $\square$

**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.  $\square$

**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  $\mathbf{H}_1[1, 1] = 0.7$ ,  $\mathbf{H}_1 \mathbf{C}_1 + \mathbf{H}_2 \mathbf{C}_2$  has a negative eigenvalue.  $\square$

## B DETAILS OF MAIN RESULTS

*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  $\eta \rightarrow 0$ , 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

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

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

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

which corresponds to the ordinary differential equations (4.2).  $\square$

*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, we know that the 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 asymmetric and maybe not positive in quadratic form in general.

Similarly, for Statement 2, we know the NTK of 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 flat clipping algorithm and (4.4) for the 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.  $\dot{L} = 0 = \frac{\partial L}{\partial \mathbf{f}} \dot{\mathbf{f}}$ . Suppose we have  $L > 0$ , then  $\frac{\partial L}{\partial \mathbf{f}} \neq 0$  since  $L$  is convex in the prediction  $\mathbf{f}$ . In this case, we know  $\dot{\mathbf{f}} = 0$ . Observe that

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

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

*Proof of Theorem 2.* The proof is similar to the previous proof. The first statement is obvious.

Now, to prove the second statement, we note that for both flat clipping and layerwise clipping, (4.3) and (4.4) give  $\dot{L}(t) < 0$  since the NTK is 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  $\frac{\partial L}{\partial \mathbf{f}} = \mathbf{0}$ , it implies all  $\ell_i = 0$ , and thus  $L = 0$ .  $\square$

*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 by any privacy accountant.  $\square$

## 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  $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ , loss function  $\ell(f(\mathbf{x}_i, \mathbf{w}_t), y_i)$ .

**Parameters:** initial weights  $\mathbf{w}_0$ , learning rate  $\eta_t$ , subsampling probability  $p$ , number of iterations  $T$ , noise scale  $\sigma$ , clipping norm  $R_r$ .

**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**  $r = 1, \dots, d$  **do**

**for**  $i \in I_t$  **do**

$v_{r,t}^{(i)} \leftarrow \nabla_{\mathbf{w}_r} \ell(f(\mathbf{x}_i, \mathbf{w}_t), y_i)$

            Option 1:  $C_{local,(i,r)} = \min \{1, R_r / \|v_{r,t}^{(i)}\|_2\}$  ▷ Local clipping factor

            Option 2:  $C_{global,(i,r)} \equiv \mathbb{I}\{\|v_{r,t}^{(i)}\|_2 \leq R_r\}$  ▷ 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

$\tilde{V}_{r,t} \leftarrow \bar{V}_{r,t} + \sigma R_r \cdot \mathcal{N}(0, I)$  ▷ Apply Gaussian mechanism

$\mathbf{w}_{r,t+1} \leftarrow \mathbf{w}_{r,t} - \frac{\eta_t}{|I_t|} \tilde{V}_{r,t}$  ▷ Descend

**Output**  $\mathbf{w}_{r,T}$

---

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

## D CODE IMPLEMENTATION

Building on of the Opacus library v0.15.0, 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

$$C_{global,i} = \begin{cases} 1 & \text{if } C_{local,i} = 1 \\ 0 & \text{if } C_{local,i} < 1 \end{cases}$$

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),

```
if hasattr(config, 'clipping_fn') and config.clipping_fn != 'local':
    clip_factor = (clip_factor >= 1).float()
```

Here we use the package `config` to pass global variable. Comparing to the original PyTorch implementation, our code only computes an additional boolean operation, thus the extra computational complexity is negligible.

## E EXPERIMENTAL DETAILS

### 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](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 6 and Figure 14, we plot in Figure 13 the distribution of prediction probability on the true class, say  $[\pi_i]_{y_i}$  for the  $i$ -th sample (notice that Figure 6 plots  $\max_k [\pi_i]_k$ ). 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.

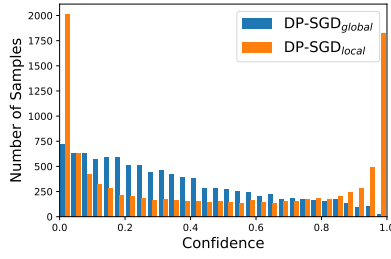


Figure 13: Prediction probability on the true class on CIFAR10 with 5-layer CNN.

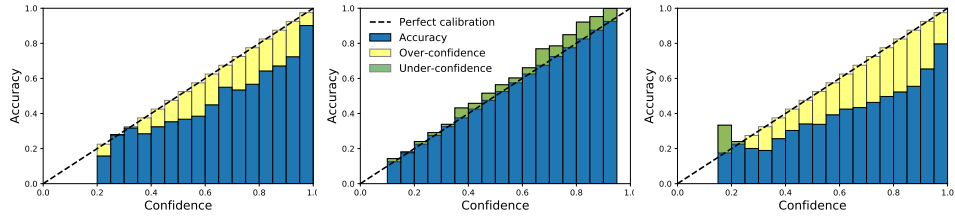


Figure 14: Reliability diagrams (left for non-DP; middle for global clipping; right for local clipping) on CIFAR10 with 5-layer CNN.

### 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](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 10 and Figure 16, we plot the distribution of prediction probability on the true class in Figure 15. 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.

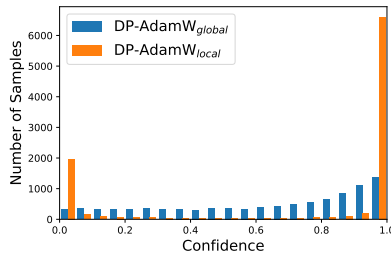


Figure 15: Prediction probability on the true class on SNLI with BERT.

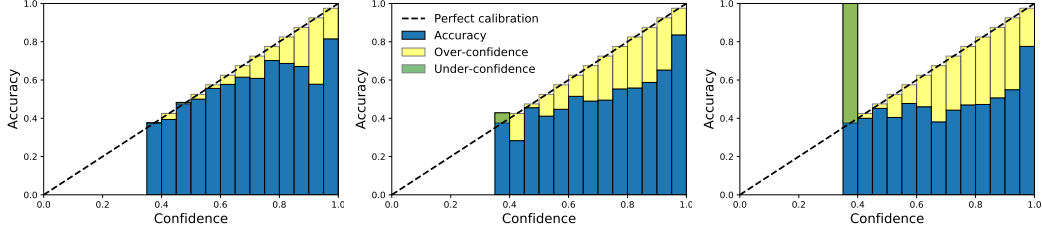


Figure 16: Reliability diagrams (left for non-DP; middle for global clipping; right for local clipping) on SNLI with BERT. Note that global clipping is only used for the last 2500 iterations out of the entire 54000 iterations.

#### E.4 REGRESSION EXPERIMENTS

We experiment on the Wine Quality<sup>9</sup> (1279 training samples, 320 test samples, 11 features) and California Housing<sup>10</sup> (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  $\sigma = 1$ ,  $R_{local} = 1$ ,  $\eta_{local} = 0.0002$ ,  $R_{global} = 2000$ ,  $\eta_{global} = 1/2000 * 0.0002$ . 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  $\sigma = 35$ ,  $R_{local} = 2$ ,  $\eta_{local} = 2$ ,  $R_{global} = 400$ ,  $\eta_{global} = 2/400 * 0.03$ . 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 11 and Figure 12, 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.

<sup>9</sup><http://archive.672ics.uci.edu/ml/datasets/Wine+Quality>

<sup>10</sup><http://lib.stat.cmu.edu/datasets/houses.zip>

**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  $\|v_t^{(i)}\|_2 \leq R$ , 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 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))

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

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

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

$$\sum_i \nabla_{\mathbf{w}} \ell_i C_i = r \mathbf{m} = \mathbf{0}. \quad (\text{F.3})$$

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

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

where  $\frac{\partial L}{\partial \mathbf{f}}$  is defined in (4.3). If the NTK is positive in eigenvalues, as is the case for local flat and global clipping, we get  $\frac{\partial L}{\partial \mathbf{f}} = \mathbf{0}$  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  $\frac{\partial L}{\partial \mathbf{f}}^\top \neq \mathbf{0}$  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))

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

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

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

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  $\frac{\partial \mathbf{f}}{\partial \mathbf{w}}$ , we get again  $\mathbf{H}\mathbf{C} \frac{\partial L}{\partial \mathbf{f}} = \mathbf{0}$ , 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)

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

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

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  $\frac{\partial \mathbf{f}}{\partial \mathbf{w}}$ , we get again  $\mathbf{H}\mathbf{C} \frac{\partial L}{\partial \mathbf{f}}^\top = 0$ , 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  $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ , loss function  $\ell(f(\mathbf{x}_i, \mathbf{w}_t), y_i)$ .

**Parameters:** initial weights  $\mathbf{w}_0$ , learning rate  $\eta_t$ , subsampling probability  $p$ , number of iterations  $T$ , noise scale  $\sigma$ , clipping norm  $R$ , momentum parameters  $(\beta_1, \beta_2)$ , initial momentum  $m_0$ , initial past squared gradient  $u_0$ , and a small constant  $\xi > 0$ .

**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**

$v_t^{(i)} \leftarrow \nabla_{\mathbf{w}} \ell(f(\mathbf{x}_i, \mathbf{w}_t), y_i)$

        Option 1:  $C_{local,i} = \min \{1, R/\|v_t^{(i)}\|_2\}$

        ▷ Local clipping factor

        Option 2:  $C_{global,i} \equiv \mathbb{I}\{\|v_t^{(i)}\|_2 \leq R\}$

        ▷ Global clipping factor

$\bar{v}_t^{(i)} \leftarrow C_i \cdot v_t^{(i)}$

        ▷ Clip the gradient

$\tilde{V}_t \leftarrow \frac{1}{|I_t|} \left( \sum_{i \in I_t} \bar{v}_t^{(i)} + \sigma R \cdot \mathcal{N}(0, I) \right)$

    ▷ Apply Gaussian mechanism

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) \tilde{V}_t$

$u_t \leftarrow \beta_2 u_{t-1} + (1 - \beta_2) (\tilde{V}_t \odot \tilde{V}_t)$

    ▷  $\odot$  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  $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ , loss function  $\ell(f(\mathbf{x}_i, \mathbf{w}_t), y_i)$ .**Parameters:** initial weights  $\mathbf{w}_0$ , learning rate  $\eta_t$ , subsampling probability  $p$ , number of iterations  $T$ , noise scale  $\sigma$ , clipping norm  $R$ , number of JL projections  $r$ .

---

```

for  $t = 0, \dots, T-1$  do
  Take a subsample  $I_t \subseteq \{1, \dots, n\}$  from training set  $D$  with subsampling probability  $p$ 
  Sample  $u_1, \dots, u_r \sim \mathcal{N}(0, I)$ 
  for  $i \in I_t$  do
     $v_t^{(i)} \leftarrow \nabla_{\mathbf{w}} \ell(f(\mathbf{x}_i, \mathbf{w}_t), y_i)$ 
    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:  $C_{local,i} = \min\{1, R/M_i\}$  ▷ Local clipping factor
    Option 2:  $C_{global,i} \equiv \mathbb{I}\{\|v_t^{(i)}\|_2 \leq R\}$ 
     $\bar{v}_t^{(i)} \leftarrow C_i \cdot v_t^{(i)}$  ▷ Clip the gradient
   $\bar{V} \leftarrow \sum_{i \in I_t} \bar{v}_t^{(i)}$  ▷ Sum over batch
   $\tilde{V}_t \leftarrow \bar{V} + \sigma R \cdot \mathcal{N}(0, I)$  ▷ Apply Gaussian mechanism
   $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{|I_t|} \tilde{V}_t$  ▷ 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  $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ , loss function  $\ell(f(\mathbf{x}_i, \mathbf{w}_t), y_i)$ .**Parameters:** initial weights  $\mathbf{w}_0$ , learning rate  $\eta_t$ , subsampling probability  $p$ , number of iterations  $T$ , clipping norm  $R$ , and a prior  $p(\mathbf{w})$ .

---

```

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
     $v_t^{(i)} \leftarrow \nabla_{\mathbf{w}} \ell(f(\mathbf{x}_i, \mathbf{w}_t), y_i)$ 
    Option 1:  $C_{local,i} = \min\{1, R/\|v_t^{(i)}\|_2\}$  ▷ Local clipping factor
    Option 2:  $C_{global,i} \equiv \mathbb{I}\{\|v_t^{(i)}\|_2 \leq R\}$  ▷ 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.