Anonymization, Privacy : Deep Learning with Differential Privacy

Kaoutar Boulif and Charlotte Durand

April 2021

Contents

1 Introduction

As more and more problems are solved through machine learning algorithms, they often include large datasets usually involving sensitive data. Thus, the model should not reveal private information about a particular individual and should keep it anonymous.
The paper "Deep Learning with Differential Privacy" [1] deals with problems related to privacy in data used to train machine learning algorithms. Indeed, authors aim to combine machine learning algorithms with advanced privacy preserving mechanisms by training neural networks within a modest privacy budget. The implementation is done on the machine learning framework TensorFlow for training models with differential privacy. The two datasets that were used are 'MNIST' and

'CIFAR-10'. In this short report, we will briefly summarize the paper [1] and then present our contributions and results.

2 Article Presentation

2.1 Basic concepts

2.1.1 Differential privacy

An algorithm is said to be differentially private if by looking at the output, one cannot tell whether any individual's data was included in the original dataset or not [3]. Formally, DP is defined as follows:

Definition A randomized mechanism $M : D \to R$ with domain D and range R satisfies $(\epsilon, \delta)-$differential privacy if for any two adjacent inputs d, $d' \in D$ and for any subset of outputs $S \subseteq R$ it holds that:

$$Pr[M(d) \in S] \leq e^{\epsilon} Pr[M(d') \in S] + \delta \tag{1}$$

If $\delta = 0$, Mechanism is ϵ-differentially private. In case of (ϵ, δ) differentially private mechanisms, there is a little possibility that an attacker can find out information about a particular individual.

2.1.2 Sensitivity

Sensitivity is said to be maximum change in the output of query of a function when we remove one of the instances from the database [2]. Mathematically, it is defined as:

$$\Delta f = max||f(D_1) - f(D_2)||_1 \tag{2}$$

2.2 Differentially Private SGD Algorithm

The algorithm used in the paper (ie. Figure 1) aims to control the influence of the training data during the training process. Given inputs, a loss function which is taken averaged on all the examples, learning rate ν_t, noise scale σ, group size L and gradient norm bound C, the algorithm starts by initializing the parameters weights and biases, then it takes a random sample L_t with a uniform probability L/N. After that, it computes the gradient, clip it so that the gradient values stays in a specific interval if it exceeds an expected range. Then it adds the noise to the gradient calculated over the lots and finally take the descent step. The output are final model parameters and the overall privacy cost (ϵ, δ) using a method known as privacy accounting method, which computes the overall privacy cost of the training according to the comparability of differential privacy.

Algorithm 1 Differentially private SGD (Outline)

```
Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate $\eta_t$, noise scale $\sigma$, group size $L$, gradient norm bound $C$.
  Initialize $\theta_0$ randomly
  for $t \in [T]$ do
    Take a random sample $L_t$ with sampling probability $L/N$
    Compute gradient
    For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$
    Clip gradient
    $\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i) / \max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$
    Add noise
    $\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$
    Descent
    $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$
  Output $\theta_T$ and compute the overall privacy cost $(\varepsilon, \delta)$ using a privacy accounting method.
```

Figure 1: DP-SGD algorithm

3 Experimental results

3.1 Python implementation

The following results are available on the notebook joined to this report. We implemented our code on Python using classical neural network library Tensorflow. The authors of the article presented above created a library Tensorflowprivacy where the DP-SGD algorithm is implemented.

3.1.1 Neural Network Model

First of all, we needed to select a neural network model to be trained on our different datasets. We chose a Convolutional Neural Network (CNN) different from the one proposed in the article. Its structure is presented in the Fig. 2. There are a 2D convolutional layer followed by a MaxPooling layer, two times. Then, after flattening the output, we have two dense layers, the first one activated by a ReLU and the last one by a softmax function to obtain a multiclass classification. As in the article, we decided to use the Categorical Cross-Entropy loss to train the neural network. Several optimizers are proposed to run the experiments :

- DP − SGD : Stochastic Gradient Descent with Differential

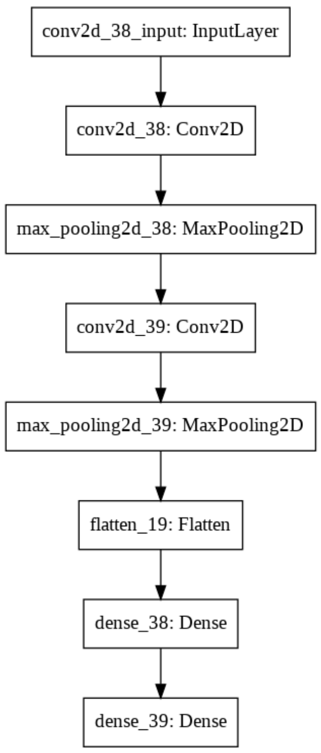
conv2d_38_input: InputLayer
conv2d_38: Conv2D
max_pooling2d_38: MaxPooling2D
conv2d_39: Conv2D
max_pooling2d_39: MaxPooling2D
flatten_19: Flatten
dense_38: Dense
dense_39: Dense

Figure 2: Model architecture

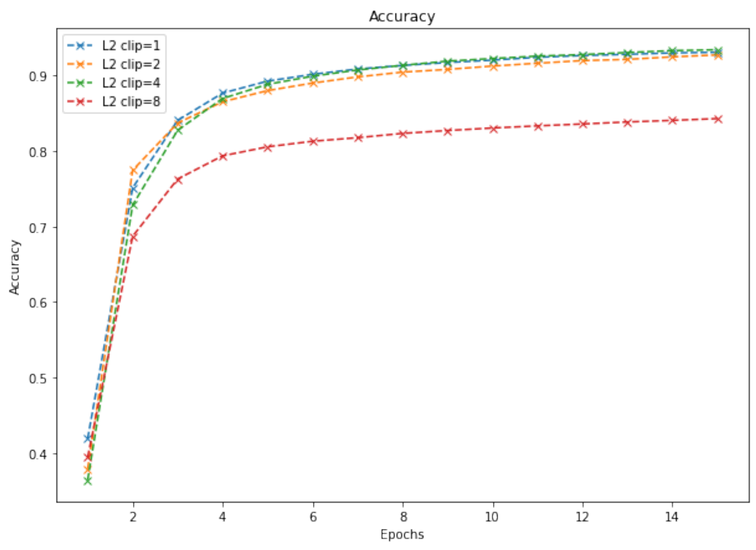

Figure 3: Influence of L2 norm clipping for a fixed learning rate of 0.1 and a fixed noise multiplier of 0.5

Privacy, the same optimizer used on the article, when not explicitly specified, it will be the optimizer used.

- DP − Adam : Adam with differential privacy
- DP − Adagrad : Adagrad with differential privacy

3.2 MNIST

In this section, we will discuss the results obtained on the classical dataset MNIST. It is a dataset containing hand-written images of single digits, from 0 to 9. The goal is to correctly classify the digits seen in the image with the right number. There are 60000 examples in the training set. The 10000 examples in the testing set are used as a validation set.

3.2.1 Influence of norm clipping

To begin with, we wanted to see what could be the influence of the L2 norm clipping presented in the DP-SGD algorithm. As it can be seen in Fig. 3, a small value of this norm clipping seems to offer better results : accuracy around 0.9% for norm clipping value of 1, 2 and 4 and the accuracy drop with a value of 8. The best accuracy is attained for a value of 4, this will be the value chosen for the following experiments.

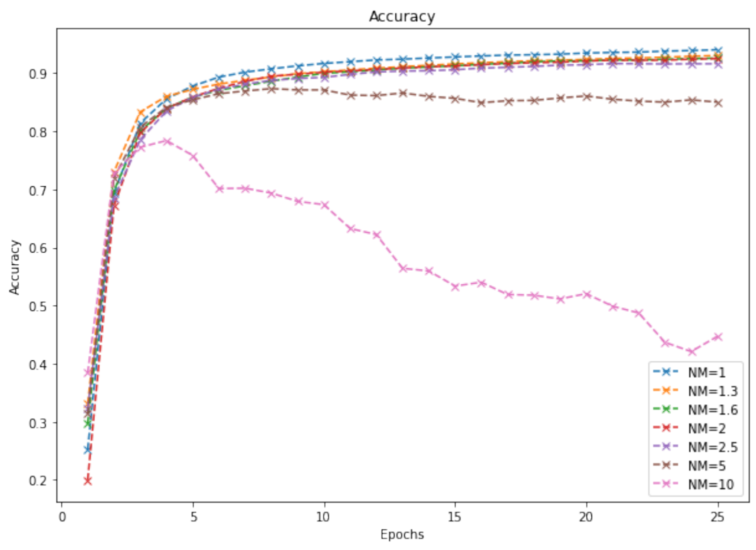

Figure 4: Influence of the noise multiplier on the accuracy for 15 epochs, with a learning rate of 0.08 and a L2 clip of 4

3.2.2 Influence of noise multiplier

In the Fig.4, we can see that if the noise multiplier parameter is too important, the system won't be stable at the end : it is quite intuitive : if we add too much noise on the gradient, the descent gradient will become more and more complicated. The issue is that if we want to insure a good differential privacy, we need to add enough noise.

A function is available in Tensorflow Privacy to compute the value of ϵ depending on different parameters such as the number of example in the dataset, the size of the microbatches, the number of epochs and the value of δ. In our case, its value is fixed to $1e^{-5}$ as it is inversely proportional to the number of examples in the dataset.
In Fig.5, we can see the value of the accuracy computed just before at the end of 15 epochs depending on the value of ϵ insured by the algorithm. As it was expected, if we want a good privacy, which means à low value of ϵ, it will means a decrease of the accuracy, which is obtained for the important value of noise multiplier. But others parameters are involved in the computation of epsilon such as the number of epochs. In Fig.6 we can see that for each value of noise multiplier, we have a constant increase of the value of ϵ when the number of epochs is increasing. This means that a long training do not benefit an interesting differential privacy. It is thus important to fine-tune the hyper parameters, like the learning rate and the number of epochs.

3.2.3 Comparison of optimizers

In Fig. 7, we can observe the differences on the training of our model for different optimizers. Each time, L2 norm clipping and noise multiplier are kept the same : 4 and 0.5. We obtain better results for Adagrad and Adam. the counterpart is that the computation takes a little longer. We can note that for Adam, it was necessary to significantly decrease the learning rate, as it is normally the case

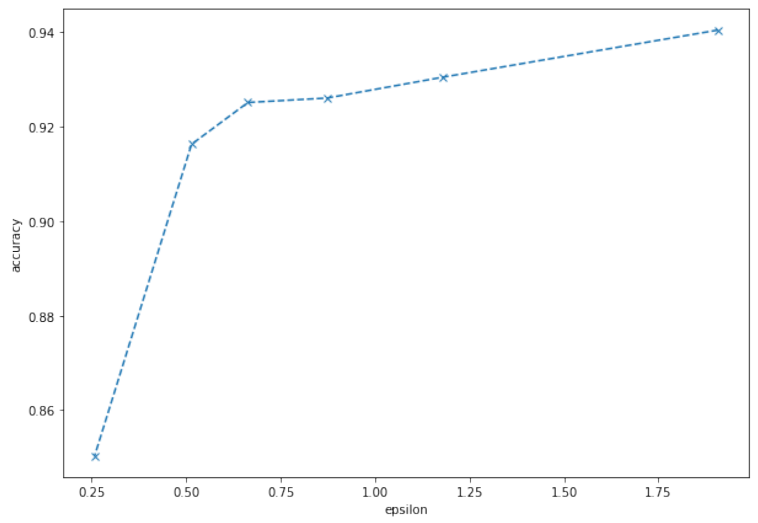

Figure 5: Influence of the value of epsilon on the accuracy of our model

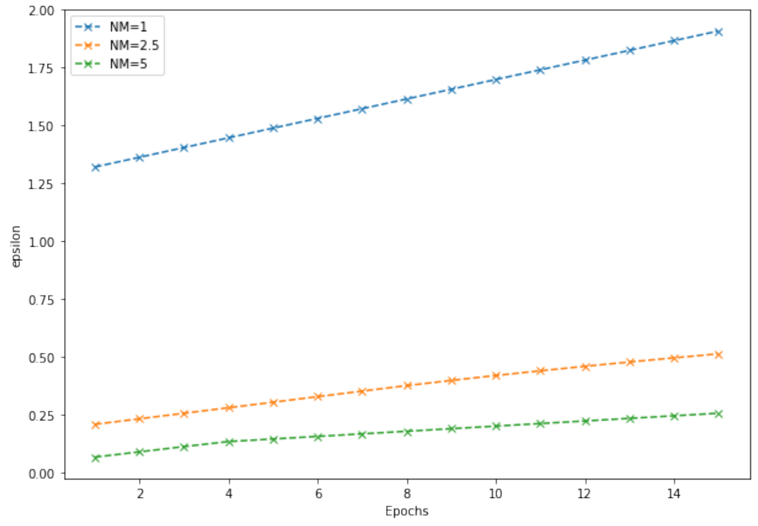

Figure 6: Influence of the number of epochs for different noise multiplier values on ϵ

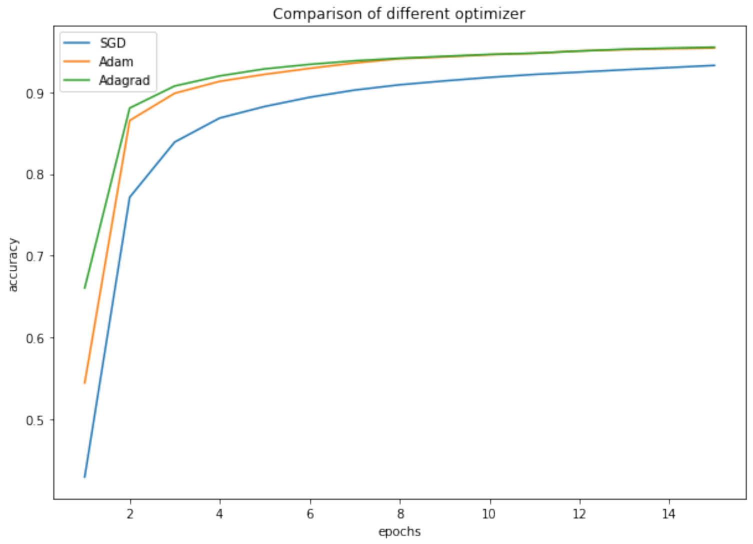

Figure 7: Comparison of different optimizers. For SGD and Adagrad, we have a 0.1 learning rate. For Adam, we have a 0.001 learning rate

in classical neural network.

3.3 Fashion MNIST

We also decided to apply DP-SGD algorithm to another dataset not presented in the article. To simplify the comparison with MNIST, we decided to experiment this algorithm on Fashion-MNIST. This dataset is similar to MNIST : ten classes of 70000 images representing various garments. Yet, the differences between garments are bigger than differences between numbers, thus the classification is a little bit more complicated. The non-DP accuracy is around 80% when it was around 95% for MNIST.

3.3.1 Influence of Noise Multiplier

We run the same experiments as before with a bigger learning rate and less epochs, as each epoch takes more time than for MNIST, we find similar results as well which is quite interesting. We may note than in Fig. 9 it seems that the accuracy is actually better for middle value of ϵ. This results might be linked with the fact that we did not made the training on an important number of epochs. It might be due to the non-finalization of the training. Yet, we can already see in Fig.8 that the model isn't stable for important value of noise multiplier.

3.3.2 Use of different optimizers

In the case of Fashion MNIST, we can see in Fig. 10 that the change of the optimizer has a bigger influence than for MNIST : SGD has lower performances than Adam and Adagrad.

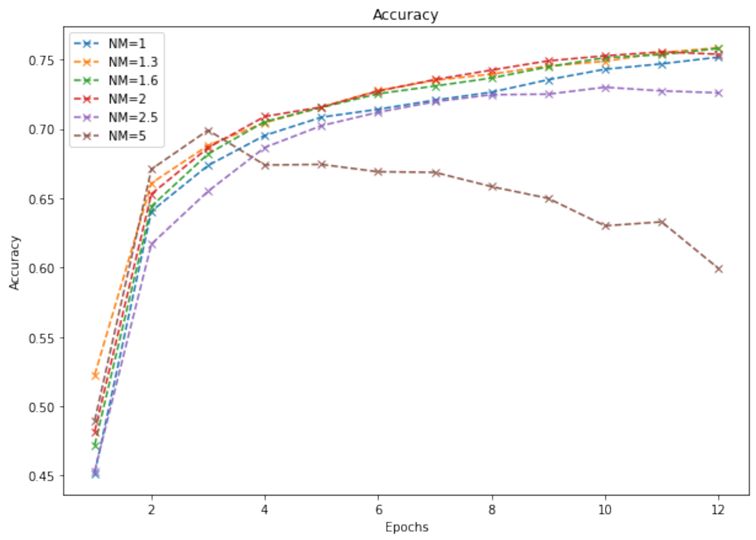

Figure 8: For Fashion MNIST, influence of the noise multiplier parameter on the accuracy. The experiments are made on 12 epochs with a L2 norm clipping value of 4 and a learning rate of 0.2

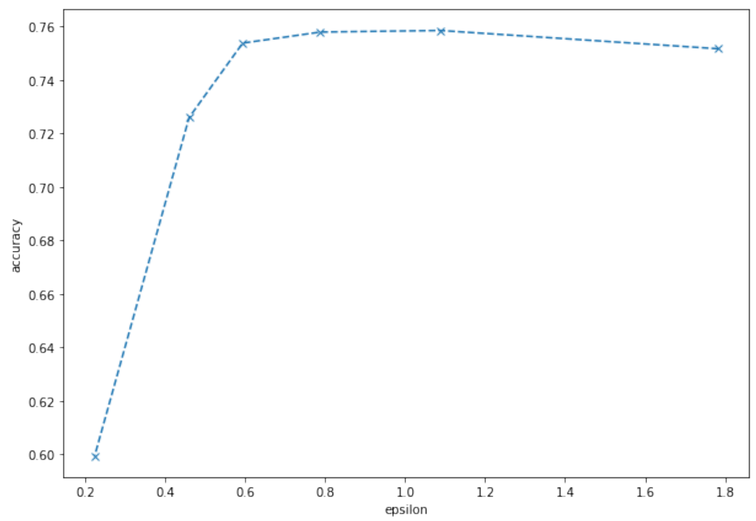

Figure 9: Influence of the value of epsilon on the accuracy of our model on fashion MNIST

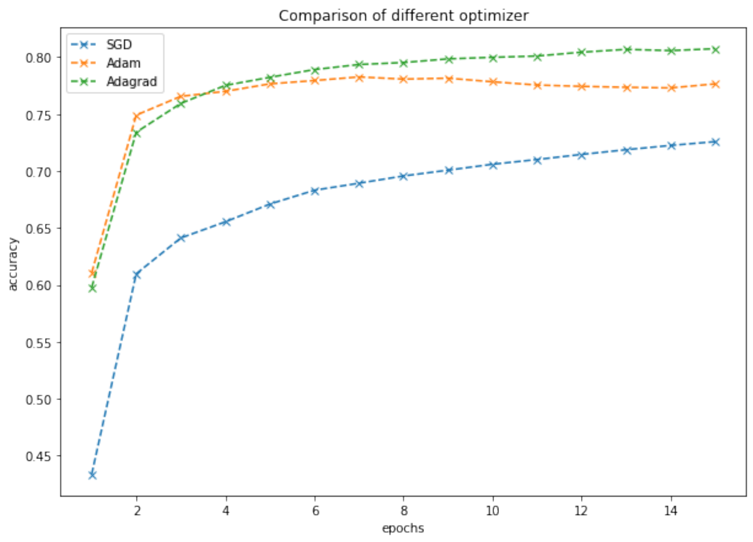

Figure 10: Comparison of different optimizers. For SGD and Adagrad, we have a 0.2 learning rate. For Adam, we have a 0.005 learning rate

4 Conclusion

We proposed in this short report some experimental results on the implementation of DP-SGD, a Differential Privacy algorithm which can be exploited on deep neural network. It offers some privacy, yet a better privacy come with a decrease of the accuracy of the model in comparison to a non differential private algorithm. Furthermore, the time of computation of each epoch is much more important than without differential privacy. Knowing that our dataset and our model is quite small compared to deep learning state-of the art results, we can wonder whether this algorithm is actually applicable to such model.

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] Sourav Kumar. Summary of deep learning with differential privacy, 2019.

[3] Harvard University. Differential privacy.