Feature-level privacy loss modelling in differentially private machine learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We introduce *Tritium*, an automatic differentiation-based sensitivity analysis framework for differentially private (DP) machine learning (ML). Optimal noise calibration in this setting requires efficient Jacobian matrix computations and tight bounds on the L2-sensitivity. Our framework achieves these objectives by relying on a functional analysis-based method for sensitivity tracking, which we briefly outline. This approach interoperates naturally and seamlessly with static graph-based automatic differentiation, which enables order-of-magnitude improvements in compilation times compared to previous work. Moreover, we demonstrate that optimising the sensitivity of the entire computational graph at once yields substantially tighter estimates of the true sensitivity compared to interval bound propagation techniques. Our work naturally befits recent developments in DP such as individual privacy accounting, aiming to offer improved privacy-utility trade-offs, and represents a step towards the integration of accessible machine learning tooling with advanced privacy accounting systems.

1 Introduction

Despite the growing availability of high-performance algorithmic tools for advanced statistical modelling and machine learning (ML), solutions to many of the world's most important problems require access to sensitive or confidential data. Technologies such as differential privacy (DP) can allow drawing insights from such data while objectively allocating and quantifying individual privacy expenditure. Although DP is the gold standard for data protection, its application to everyday ML workflows is –in practice –often constrained. For one, tightly introspecting the privacy attributes of complex models such as deep neural networks can be very challenging. Moreover, substantial expertise is required on the analyst's behalf to correctly apply DP mechanisms to such models. Software libraries [1, 2, 3, 4] are being developed to alleviate these issues in specific domains such as DP deep learning. They are, however, limited to a small number of programming languages and application programming interfaces (APIs). The democratisation of DP machine learning therefore awaits generic infrastructure, not only compatible with arbitrary workflows, but designed "from first principles" to facilitate the implementation of DP. At its core, contemporary ML is based around the manipulation of multidimensional arrays and the composition of differentiable functions, a programming paradigm referred to as *differentiable programming*. Besides deep learning, some of the most successful ML algorithms [5] and a large number of statistical queries, especially from the domain of *robust statistics*, can be expressed within this paradigm. *Automatic differentiation* (AD) systems are the core of differentiable programming frameworks and are able to track the flow of computation to return precise derivatives with respect to arbitrary computational quantities. Although this functionality may –at first –seem orthogonal to the goals described above, we contend that it is in fact not only highly compatible, but *synonymous* with automatic DP tracking.

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

In the current work, we present *Tritium*, a differentiable programming framework aiming to integrate the requirements of ML and privacy analysis through the use of AD. We recapitulate the link between the *sensitivity* of differentiable queries and the *Lipschitz* constant in section 2. We outline our system's implementation in section 3 and present the substantial improvements in computational efficiency and sensitivity bound tightness in section 4. A discussion of prior work can be found in the appendix.

2 Theoretical motivation

We begin by briefly introducing an interpretation of DP using the language of functional analysis, which forms the theoretical motivation behind our work. We concentrate on the Gaussian mechanism, which forms the basis for private data analysis in high dimensions. Differentially private ML can fundamentally be abstracted as the application of a higher-order function (or *functional*) to private data. This higher-order function (often termed a *DP mechanism*) receives as its input another function (termed a *query*) which has been applied to a private dataset, inspects the query to derive its privacy attributes and modifies it to preserve DP (Definitions 1 and 2).

Definition 1 (Query)**.** *A query is a function $q : \mathrm{R}^{m\times\mathcal{D}} \to \mathrm{R}^{n\times\mathcal{D}}$, where $\mathcal{D}$ represents arbitrary (possibly unused) dimensions and $n, m \geq 1$ which receives as input some private dataset $\mathbf{x}, |\mathbf{x}| \geq 1$ and outputs a result $\mathbf{y}$ representing the result of a computation over $\mathbf{x}$ (e.g. a mean calculation or the output of a neural network).*

Definition 2 (DP mechanism)**.** *A DP mechanism is a higher-order function M which receives as its input one or more query functions $q_1, q_2, \ldots, q_n$ and outputs $M(q_1 \circ q_2 \circ \cdots \circ q_n) = q_n(\ldots q_2(q_1(\mathbf{x}))) + \xi$, where $\xi \sim \mathcal{N}(0, C)$ and C is selected based on the privacy properties of $q_1 \circ q_2 \circ \cdots \circ q_n$.*

The tight characterisation of these privacy properties is central to enabling privacy expenditure tracking. The effect on inputs on the output of the query functions is reflected in query *sensitivity*. We use the *Lipschitz* constant to reason about sensitivity.

Definition 3 (Lipschitz constant and sensitivity)**.** *Let $q : X \to Y$ be a function between metric spaces X and Y with distance metrics d_X and d_Y, respectively. Then q is Lipschitz continuous with constant K_q (equivalently, "K-Lipschitz") if*

$$d_Y(f(x), f(x')) \leq K\, d_X(x, x')\ \forall x, x' \in X \tag{1}$$

The smallest value of K corresponds to the sensitivity of $q, \Delta_2(q)$ [6]:

$$\Delta_2(q) = \max_{x,x'} \|q(x) - q(x')\|_2 \tag{2}$$

where $\|\cdot\|_2$ is the L_2 distance. Recall that in this case, $d(x, x') = 1$ as x, x' are adjacent. Thus for differentiable query functions, $K_q \equiv \Delta_2(q)$ when X and Y are Euclidean spaces endowed with the L_2-norm. Then,

$$K_q = \sup \|\mathcal{J}(q)\|_2 \tag{3}$$

where $\mathcal{J}$ is the Jacobian matrix (the differential operator).

This equivalence between Lipschitz constant and query sensitivity allows, in principle, to reason over the privacy attributes of individual query functions and calibrate noise appropriately. Typical functions with globally bounded sensitivity are affine queries or linear functions of the form $q(x) = \alpha x(+\beta)$, with $K_q = \alpha$. However, queries exist for which the Lipschitz constant is not defined over the entire input domain. One example of such a function is $q(x) = x^2 = x \cdot x$ with $K_q = x$, whose sensitivity is *unbounded*, as it depends on the value of x. The sensitivity analysis of such queries sometimes therefore depends on (private) properties of the dataset. We term such a case as *data-dependent sensitivity*. Reasoning over sensitivity in such cases is complicated by a requirement to propagate this data dependency effect through function composition. Previous works on Lipschitz analysis of machine learning algorithms [7, 8] achieve this through techniques such as *interval bound propagation* [9], that carry the bounds on input variables (which, for DP, should be defined in a data-agnostic manner) through the computation flow. This technique can easily be made compatible with *tracing-type* AD systems which are widely used in contemporary machine learning. However, due to well-known limitations of interval arithmetic (such as interval dependency [10]) and due to

the fact that the Lipschitz constant is defined by inequality, the resulting sensitivity terms may be valid, but too loose to be of any practical utility (e.g. 10^5). The last challenge relates to the fact that the *actual* effect of an individual's data on the query's output may, in fact, be much smaller than the worst case assumed by the definition, resulting in more noise being added by the mechanism than would be required for the guarantee to hold. Consequently, although the worst-case sensitivity value has typically been used for privacy accounting, newer techniques perform accounting based not only the worst case but combine it with the actual output L_2-norm [11].

3 Implementation details

Our work presents *Tritium*, an automatic-differentiation-based machine learning and sensitivity analysis system engineered to address the above-mentioned challenges. It consists of the following components:

1. A user-facing front-end to specify a query $\mathbf{q} = q_1 \circ \cdots \circ q_n$ *abstractly*, i.e. without directly utilising private data during model creation. This is achieved through the utilisation of abstract tensors with pre-defined dimensions. The system creates an optimised computational graph $\mathcal{G}$ based on this specification.
2. During model specification, the user can impose *bounds* on the quantities (e.g. inputs, weights) used in the model.
3. The user selects the desired *privacy parameters*, e.g. ε and δ values or a maximum allowed sensitivity.
4. A *compiler* then emits a program which receives a private dataset and outputs an appropriately privatised result.

Internally, *Tritium* undertakes the following steps:

1. The computational graph $\mathcal{G}$ is compiled into a program which outputs $\mathcal{J}(\mathbf{q})$ with respect to the inputs.
2. $K_q = \sup \|\mathcal{J}(\mathbf{q})\|_2$ is computed given the input bounds.
3. Finally, $\mathcal{G}$ is compiled into the program described in step (4) above which receives a private dataset $\mathbf{x}$, computes $\mathbf{q}(\mathbf{x})$, potentially *clips* out-of-bound values to preserve the required $K_\mathbf{q}$, adds noise $\xi \sim \mathcal{N}(0, C)$ with C proportional to $K_\mathbf{q}$ to satisfy the required ε value for a given δ and outputs $M(\mathbf{q})$.

This system architecture has several benefits: It avoids utilising private data until the moment the final computation is executed (*data minimisation*). Moreover, it provides a tight sensitivity calculation by optimising the entire query function at once [12] instead of the above-mentioned forward-propagation, which can lead to vacuous sensitivity values. Furthermore, it utilises the pre-specified bounds on the input variables to not only enable the calculation of *data-dependent* sensitivity, but also greatly accelerate the process. Moreover, it is *agnostic* to the method used to actually *obtain* the desired sensitivity. For example, *Lipschitz neural network layers* [13, 14] or activation functions with bounded outputs and gradients [15] can be used for model building, but bounded sensitivity can also be enforced by clipping, as is common in DP-SGD [16]. In addition, the system is able to compute the full Jacobian matrix (which is required in DP-SGD) as well as arbitrary higher-order derivative matrices (which can be used to accelerate the sensitivity computation). Additionally, as the system outputs both the Lipschitz constant and the norm of the outputs, it can be leveraged to provide tighter privacy guarantees through the use of e.g. *individual privacy accounting*, as shown below. Finally, the system is designed to output privatised values by default instead of outputting non-private values and relying on the user to perform an appropriate privatisation step. This can reduce both user workload and the probability of failure due to incorrect application of DP mechanisms on the user's behalf.

4 Experimental evaluation

4.1 Exact sensitivity calculations through *a posteriori* optimisation

To assess the benefits of computing query sensitivity by assessing the entire computational graph at once instead of forward-propagating interval bounds, we constructed a small neural network

comprising 4 *linear* layers with *logistic sigmoid* activations and the *binary cross-entropy* cost function. We set the bounds for the neural network weights and the input bounds to the $[0, 1]$ interval. We then calculated the sensitivity using two techniques: *Interval Bound Propagation* (IBP [9]) and our proposed method optimising the entire computational graph at once. The estimate of the sensitivity was returned as $[0.0, 22175.37]$ by IBP (which is a valid, but vacuous bound) and as 0.99929 by *Tritium*. The IBP bounds are also similar to previous work (compare e.g. [17]). However, IBP was faster, requiring 103 ms to return a result, compared with 905 ms for our technique (excluding compilation time of ca. 3 s). These results are summarised in Table 1 in the appendix.

4.2 Compilation improvements

In this section, we compare the compilation and execution time improvements of *Tritium* to the previously described framework by [18] on neural network architectures with the architecture described above, but with increasing width of linear layers. We recall that the system proposed by the authors of this work relies on a scalar AD implementation and authors report compilation times quadratic in the number of model parameters (that is, around 60 hours for a 2.5 million parameter model). In contrast, the here-presented implementation utilises the *Aesara* library (a fork of the now-defunct *Theano* framework [19]) as a computational back-end. This allows both a memory-efficient vectorised execution of tensor operations and a more mature and faster compilation back-end. For all but the smallest architectures, this back-end achieved substantially faster compilation times which were independent of the number of parameters and able to leverage caching to accelerate re-compilation. A similar effect was observed in execution times, where our system achieved considerably higher performance. These results are visualised in Figure 1 in the appendix.

5 Discussion and conclusion

We propose *Tritium*, an automatic differentiation-based system for differentially private machine learning. Our framework relies on an interpretation of DP queries and mechanisms through the language of functional analysis, linking them by the definition of Lipschitz continuity. We found the combination of an efficient computational and compilation back-end with the consideration of the entire query function at once to yield both improved performance and tighter sensitivity bounds compared to previous work. Our proposed framework relies on *static graph-based* AD, which can apply specific compiler optimisations to the entire computational graph and is responsible for *Tritium*'s high performance. However, such systems have noteworthy limitations. For instance, the definition of control flow statements is cumbersome and such systems are not well-suited for utilisation with *just-in-time* compilers. Most currently used machine learning frameworks utilise *tracing/eager execution* AD back-ends, which, while more user-friendly, cannot always leverage the same optimisations. An alternative AD implementation, *source-to-source translation* can combine the benefits of dynamic graph specification with the high performance and optimisations of static compilation. It forms the basis of a recent paradigm in programming language design (e.g. [20]), attempting to merge a general-purpose programming language with differentiable programming primitives. A limitation of our method stems from the computational hardness of exactly calculating the Lipschitz constant [21]. Our system will output the true bound using *Simplicial Homology Global Optimisation* [22] if the bound exists and can be found, and the application of constraints can substantially accelerate this process. However, it will output a warning and switch to an approximate algorithm without a guaranteed bound otherwise. If such a bound is undesirable, an alternative technique is the utilisation of model components with known (or manually adjustable) Lipschitz constants, which can allow one to avoid the utility penalty imposed by clipping-based approaches, both enabling the design of algorithms with milder privacy-utility penalties and additionally reaping the benefits of well-defined model sensitivity, such as (certifiable) robustness to perturbations by adversarial samples. In conclusion, our work serves as a first proof-of-concept for the the design of generic infrastructure exposing familiar APIs to data scientists while automatically tracking privacy loss through the computation flow. We view the further development of such systems as an accelerator for the wide-spread adoption of privacy-preserving machine learning algorithms across data-driven research disciplines.

References

[1] Opacus PyTorch library. Available from opacus.ai, 2021.

[2] TensorFlow Privacy. Available from TensorFlow Privacy, 2021.

[3] Naoise Holohan, Stefano Braghin, Pól Mac Aonghusa, and Killian Levacher. Diffprivlib: The ibm differential privacy library, 2019.

[4] Differential Privacy. Available from google/differential-privacy, 2021.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[6] Sofya Raskhodnikova and Adam Smith. Lipschitz extensions for node-private graph statistics and the generalized exponential mechanism. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 495–504. IEEE, 2016.

[7] Aritra Bhowmick, Meenakshi D'Souza, and G Srinivasa Raghavan. Lipbab: Computing exact lipschitz constant of relu networks. *arXiv preprint arXiv:2105.05495*, 2021.

[8] Sungyoon Lee, Jaewook Lee, and Saerom Park. Lipschitz-certifiable training with a tight outer bound. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 16891–16902. Curran Associates, Inc., 2020.

[9] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models, 2019.

[10] Walter Krämer. Generalized intervals and the dependency problem. *PAMM*, 6(1):683–684, December 2006.

[11] Vitaly Feldman and Tijana Zrnic. Individual privacy accounting via a renyi filter. *arXiv preprint arXiv:2008.11193*, 2020.

[12] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

[13] Yonadav Shavit and Boriana Gjura. Exploring the use of lipschitz neural networks for automating the design of differentially private mechanisms. 2019.

[14] Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. In *International Conference on Machine Learning*, pages 291–301. PMLR, 2019.

[15] Nicolas Papernot, Abhradeep Thakurta, Shuang Song, Steve Chien, and Úlfar Erlingsson. Tempered sigmoid activations for deep learning with differential privacy. *arXiv preprint arXiv:2007.14191*, 2020.

[16] 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.

[17] Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5757–5764, 2019.

[18] Alexander Ziller, Dmitrii Usynin, Moritz Knolle, Kritika Prakash, Andrew Trask, Rickmer Braren, Marcus Makowski, Daniel Rueckert, and Georgios Kaissis. Sensitivity analysis in differentially private machine learning using hybrid automatic differentiation. 2021.

[19] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[20] Brennan Saeta, Denys Shabalin, Marc Rasi, Brad Larson, Xihui Wu, Parker Schuh, Michelle Casbon, Daniel Zheng, Saleem Abdulrasool, Aleksandr Efremov, Dave Abrahams, Chris Lattner, and Richard Wei. Swift for tensorflow: A portable, flexible platform for deep learning, 2021.

[21] Kevin Scaman and Aladin Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *arXiv preprint arXiv:1805.10965*, 2018.

[22] Stefan C Endres, Carl Sandrock, and Walter W Focke. A simplicial homology algorithm for lipschitz optimisation. *Journal of Global Optimization*, 72(2):181–217, 2018.

[23] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *arXiv preprint arXiv:1705.10941*, 2017.

[24] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

[25] Varun Gupta, Christopher Jung, Seth Neel, Aaron Roth, Saeed Sharifi-Malvajerdi, and Chris Waites. Adaptive machine unlearning. *arXiv preprint arXiv:2106.04378*, 2021.

[26] Chike Abuah, Alex Silence, David Darais, and Joe Near. Dduo: General-purpose dynamic analysis for differential privacy. *arXiv preprint arXiv:2103.08805*, 2021.

[27] Paolo Pistone. From identity to difference: A quantitative interpretation of the identity type. *arXiv preprint arXiv:2107.06150*, 2021.

A Tables and Figures

	Upper bound	Computation time (ms)
Ours	0.99929	905
IBP	22175.37	103

Table 1: Comparison of the sensitivity bounds and computation times for our proposed framework (*Ours*) vs. Interval Bound Propagation (*IBP*).

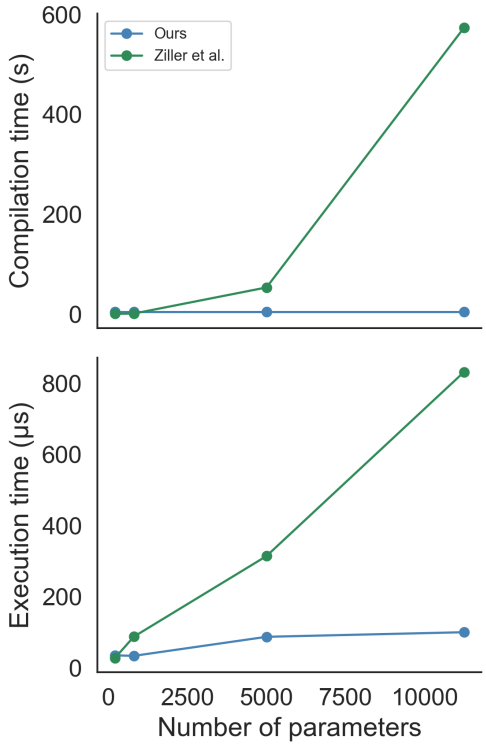

Figure 1: Performance comparison between our proposed framework (*Ours*, blue) vs. [18] (green). Compilation times in s and execution times in μs are shown for neural network architectures of increasing sizes.

B Related works

Our work can be seen as a natural evolution of the previous study by [18] in the context of AD-based sensitivity analysis for DP machine learning. In comparison to this work, *Tritium* relies on a vectorised, GPU-compatible execution engine and a mature graph compiler which drastically improves performance, as shown in the experimental section. The properties of Lipschitz continuous functions have been leveraged in several domains beyond DP. Works such as [13, 23, 14, 17] attempt to constrain the Lipschitz constant to reason over and control the properties of neural networks. The utilisation of this approach has been proposed for network certification against adversarial samples [24], whereby a network that is ϵ-certified is provably robust to input perturbations within a norm ball of radius ϵ. Additionally, constraining the Lipschitz constant has been proposed for DP model training, as this allows to calibrate the noise addition based on the bounded Lipschitz constant [13]. Moreover, certain works [25] have addressed the problem of *machine unlearning*, providing methods for a reliable removal of contributions associated with an individual in the context of neural network training. We note that the approach to sensitivity analysis employed in these studies is

orthogonal to our work, as our work is compatible with manual sensitivity constraints (such as directly adjusting the Lipschitz constant of neural network layers through appropriate layers as described above, which however may impair their expressivity) but also with sensitivity tracking for privacy loss calculation. Several recent works concentrate on the computation of accurate estimates of the Lipschitz constant mostly focused on *ReLU* networks such as [21, 14], but most of these obtain the upper bounds rather than the exact values of the Lipschitz constant, often resulting in valid, but extremely loose approximations that are not intended to be applied in DP training. A line of work centred on languages for differentially private programming also exists. Among these, the recently proposed *DDuo* framework [26] performs dynamic sensitivity analysis in the context of DP algorithm specification. As shown above however, this approach does not attempt to derive a tight bound on sensitivity in the setting of unbounded queries, declaring sensitivity as *infinite* and relying exclusively on clipping. More general approaches, including *category-theoretical* views on the intersection of differentiable programming and differential privacy such as [27] have also recently been proposed.