Differential Privacy and Neural Networks: A Preliminary Analysis

Giuseppe Manco, Giuseppe Pirrò
presented by **Ettore Ritacco**

ICAR-CNR, Italian National Research Council

PAP 2017

Outline

Outline

1. Differential Privacy

2. Differential Privacy and Neural Networks

Outline

1. Differential Privacy
2. Differential Privacy and Neural Networks
3. Concluding Remarks

Differential Privacy

- Releasing (Big) Data sets is fundamental toward analytics tasks (analysis and prediction)
- Releasing or letting analysts to use data raises privacy concerns

Differential Privacy

- Releasing (Big) Data sets is fundamental toward analytics tasks (analysis and prediction)
- Releasing or letting analysts to use data raises privacy concerns
- Differential privacy (DP) differs from data anonymization techniques in two main respects:
    - it does not require to modify the data
    - it does not assume any particular background knowledge
- DP ensures the outcomes of queries/calculations to be insensitive to any individual record in a database

Definition

Differential privacy provides information about a group while allowing to learn as little as possible about any individual in it.

Differential Privacy

Definition (Differential Privacy)

A randomized computation M provides ϵ-differential privacy if for any datasets A and B with symmetric difference $A \Delta B = 1$ (A and B differ in one record), and any set of possible outcomes $S \subseteq Range(M)$:

$$\Pr[M(A) \in S] \leq \Pr[M(B) \in S] \times e^{\epsilon} + \delta$$

- ϵ allows to control the level of privacy
- δ allows to introduce an error thus defining (ϵ, δ)-DP
  - lower values of ϵ mean stronger privacy
- The probability that an attacker guesses an individual record is (resp., is not) in the database in at most e^{ϵ}

Differential Privacy for real-valued functions

- DP is achieved by adding noise to the outcome of a query

Definition (Sensitivity of a real-valued function)

Given a function $f\colon D \rightarrow R^d$ the sensitivity of f is:

$$S(\text{f}) = \max_{A,B,A\Delta B=1} ||f(A) - f(B)||_1$$

Theorem (**DP and the Laplace mechanism**)

Given a function $f\colon D \rightarrow R^d$ the computation $M(X) = f(X) + (Laplace(S(f)/\epsilon))^d$ provides ϵ-differential privacy.

Theorem (**Approximate DP and the Gaussian Mechanism**)

Given a function $f\colon D \rightarrow R^d$, the computation $M(X) = f(X) + N(0, \sigma^2)^d + \delta$ provides (ϵ, δ)-differential privacy

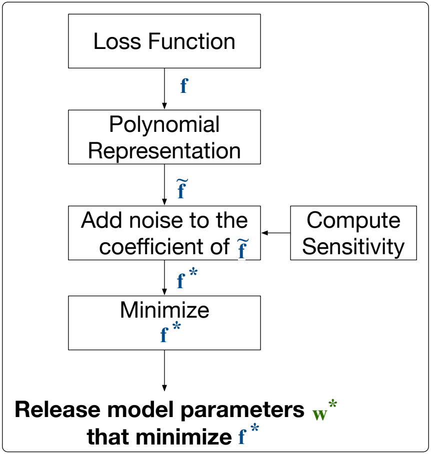
- $\sigma \geq \sqrt{2ln(2/\delta)}/\epsilon \times max_{A\Delta B}||f(A) - f(B)||_2$.

Context

- A dataset $\mathcal{D}$
- Multi Layer Perceptron (MLP) [2].
  - A set of input variables $\mathbf{x_i}$, $i \in [1, d]$
  - A set of output variable $\mathbf{t_l}$, $l \in [1, q]$
  - A a vector $\mathbf{w}$ of adjustable parameters
  - **Network training**: learn the model parameters (weights and biases) that minimize the error function $E(\mathcal{D}, \mathbf{w})$ (e.g., via the back-propagation algorithm [4]).

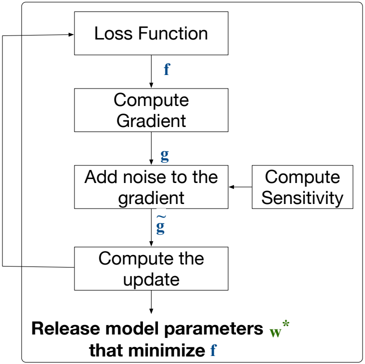
State of the Art

- **Functional Approaches** [6]: *Instead of perturbing the results, one can perturb the objective function and then optimize the perturbed objective function.*



State of the art

- **Non Functional Approaches** [1] are mainly based on *line search methods* such as the (Stochastic) Gradient Descent (SGD).



Our Approach

Our algorithm to learn via Neural Network under Differential Privacy, adopts a Functional Approach. It involves the following four main phases:

1. Find a polynomial representation of the objective function;
2. Compute the sensitivity;
3. Add noise to the polynomial representation;
4. Minimize the perturbed objective function.

Preliminaries: Neural Network Representation

- For simplicity we leave out bias terms
- $w_{i,j}$ corresponds to the weight of the connection between nodes j and i such that $j \prec i$ (j is a predecessor of i).

$$z_{\mathbf{x},i} = \phi_i\left(a_{\mathbf{x},i}\right)$$
$$a_{\mathbf{x},i} = \sum_{j \prec i} w_{i,j} \cdot z_{\mathbf{x},j}$$

- $z_{\mathbf{x},i}$ (resp. $a_{\mathbf{x},i}$) represents the application of a_i to $\mathbf{x}$
- ϕ_i represents the activation function relative to unit i.

Polynomial representation of the objective function

We consider the following 2nd-order Taylor expansion $\hat{E}(\mathcal{D}, \mathbf{w})$ of the error function $E(\mathcal{D}, \mathbf{w}) = \sum_{\mathbf{x}\in\mathcal{D}} E_{\mathbf{x}}(\mathbf{w})$.

$$\hat{E}(\mathcal{D}, \mathbf{w}) \approx E(\mathcal{D}, \hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^T \nabla E(\mathcal{D}, \hat{\mathbf{w}}) + \frac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^T \nabla^2 E(\mathcal{D}, \hat{\mathbf{w}})(\mathbf{w} - \hat{\mathbf{w}})$$

where: $\nabla E(\mathcal{D}, \hat{\mathbf{w}}) \equiv \frac{\partial E}{\partial w_{i,j}}|_{\mathbf{w}=\hat{\mathbf{w}}}$ and $(\nabla^2 E(\mathcal{D}, \hat{\mathbf{w}}))_{i,j} \equiv \frac{\partial^2 E}{\partial w_i \partial w_j}|_{\mathbf{w}=\hat{\mathbf{w}}}$ are the Jacobian and Hessian matrices, respectively. Denoting by $g_{i,j}$ (resp., $h_{i,j}$) an element of the Jacobian (resp., Hessian) matrix, we obtain:

$$g_{i,j} = \sum_{\mathbf{x}\in\mathcal{D}} \delta_{\mathbf{x},i} z_{\mathbf{x},j} \tag{1}$$

$$\delta_{\mathbf{x},i} = z'_{\mathbf{x},i} \sum_{v:i\prec v} \delta_{\mathbf{x},v} w_{v,i} \tag{2}$$

$$h_{i,j} = \sum_{\mathbf{x}\in\mathcal{D}} b_{\mathbf{x},i} z^2_{\mathbf{x},j} \tag{3}$$

$$b_{\mathbf{x},i} = z''_{\mathbf{x},i} \sum_{v:i\prec v} w_{v,i}\delta_{\mathbf{x},v} + (z'_{\mathbf{x},i})^2 \sum_{v:i\prec v} w^2_{v,i} b_{\mathbf{x},v} \tag{4}$$

Polynomial Represenation of the objective function

To reduce the computational cost we only consider diagonal elements of the Hessian [3]. As for the output units, their output depends from the specific loss function considered. As an example, when considering the Least Squares Error, we obtain:

$$
\begin{aligned}
E(\mathbf{x}, \mathbf{w}) =& \frac{1}{2} \sum_{c=1}^{d} (y_{\mathbf{x},c} - t_{\mathbf{x},c})^2 \\
y_{\mathbf{x},c} =& y_c(\mathbf{x}; \mathbf{w})
\end{aligned}
$$

where $y_{\mathbf{x},c}$ is the c-th component given by the network when giving as input the vector $\mathbf{x}$ and $t_{\mathbf{x},c}$ is the c-th target value. Hence, for output units we have:

$$
\begin{aligned}
\delta_{\mathbf{x},i} =& (\psi(a_{\mathbf{x},i}) - t_{\mathbf{x},i})\psi'(a_{\mathbf{x},i}) && (5) \\
b_{\mathbf{x},i} =& (\psi'(a_{\mathbf{x},i}))^2 + (\psi(a_{\mathbf{x},i}) - t_{\mathbf{x},i})\psi''(a_{\mathbf{x},i})) && (6)
\end{aligned}
$$

where ψ is the activation function for output units.

Compute the sensitivity

We need to estimate its sensitivity to add noise into the polynomial representation of the error function.

Lemma

Let $\mathcal{D}$, $\mathcal{D}'$ be any two databases differing in at most one tuple, and

$$\hat{E}(\mathcal{D}, \mathbf{w}) = \Big(\sum_{\mathbf{x}\in\mathcal{D}}\sum_{j\prec i}\delta_{\mathbf{x},i}z_{\mathbf{x},j}\Big)w_{i,j} + \frac{1}{2}\Big(\sum_{\mathbf{x}\in\mathcal{D}}\sum_{j\prec i}b_{\mathbf{x},i}z^2_{\mathbf{x},j}\Big)w^2_{i,j}$$

$$\hat{E}(\mathcal{D}', \mathbf{w}) = \Big(\sum_{\mathbf{x}'\in\mathcal{D}'}\sum_{j\prec i}\delta_{\mathbf{x}',i}z_{\mathbf{x}',j}\Big)w_{i,j} + \frac{1}{2}\Big(\sum_{\mathbf{x}'\in\mathcal{D}'}\sum_{j\prec i}b_{\mathbf{x}',i}z^2_{\mathbf{x}',j}\Big)w^2_{i,j}$$

the polynomial representation of the error function on $\mathcal{D}$ and $\mathcal{D}'$, respectively. Let $\mathbf{x}$ be an arbitrary tuple.

$$||\hat{E}(\mathcal{D}, \mathbf{w}) - \hat{E}(\mathcal{D}', \mathbf{w})||_1 \leq 2\sum_{j\prec i}\max_{\mathbf{x}}\left(|\delta_{\mathbf{x},i}z_{\mathbf{x},j}| + |b_{\mathbf{x},i}z^2_{\mathbf{x},j}|\right)$$

An overview of the Algorithm

Algorithm 1 `FunctionalNetDP` (Privacy budget ϵ)

1: Initialize $\mathbf{w}$
2: $g, h \leftarrow$ Polynomial Representation of the loss function
3: Find $\widetilde{g}$ and $\widetilde{h}$ via `addNoise`($\mathbf{w}$,ϵ) /* Algorithm 2 */
4: Compute $\tilde{\mathbf{w}}=\underset{\mathbf{w}}{argmin}\ \widetilde{E}(\mathcal{D}, \mathbf{w})$
5: **return** $\tilde{\mathbf{w}}$ / set of weights that minimizes $\widetilde{E}(\mathcal{D}, \mathbf{w})$ /

Algorithm 2 `addNoise`(Weights $\mathbf{w}$, privacy budget ϵ)

1: Let $\mathcal{D}$ be the dataset
2: Set $S(\widetilde{E})=2\sum_{j\prec i} \max_{\mathbf{x}} \left(|\delta_{\mathbf{x},i} z_{\mathbf{x},j}|+|b_{\mathbf{x},i} z^2_{\mathbf{x},j}|\right)$
3: Find g via eq. (1) and h via eq. (3) using $\mathcal{D}$
4: g$\leftarrow g/min(1, ||g||^2/C)$ /* clip Gradient */
5: h$\leftarrow h/min(1, ||h||^2/C)$ /* clip Hessian */
6: $\widetilde{g} \leftarrow g$+$Lap(S(\widetilde{E}) \mid \epsilon)$
7: $\widetilde{h} \leftarrow h$+$Lap(S(\widetilde{E}) \mid \epsilon)$
8: **return** $\widetilde{g}$ and $\widetilde{h}$

Theorem

Algorithm 1 satisfies ϵ-differential privacy.

A fine-grained analysis of the Laplace noise (I)

Assuming that for both ϕ and ψ the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$ is used, then we have that:

- $\sigma(x) \in [0, 1]$
- $\sigma'(x) \in [0, 0.25]$,
- $\sigma(x)'' \in [-0.1, 0.1]$

Moreover, we assume that:

- $w_{i,j} \in [0, 1]$
- for each dimension d of the input tuple $\mathbf{x}^{\mathbf{d}} \in [0, 1]$

To bound the amount of Laplace noise needed, we need to bound the components $\sum_{j \prec i} |\delta_{\mathbf{x},i} z_{\mathbf{x},j}|$ and $\sum_{j \prec i} |b_{\mathbf{x},i} z^2_{\mathbf{x},j}|$ in the previous Lemma.

Let m be the depth of the network. For each layer $j \in \{1, ..., m\}$ we denote by δ^j (resp., b^j) the coefficient of a generic unit in the layer j. Moreover, s_j represents the number of units at layer j. We now analyze the amount of noise as per our previous sensitivity analysis.

Lemma

The noise to be introduced by the $\sum_{j \prec i} |\delta_{\mathbf{x},i} z_{\mathbf{x},j}|$ *component is* $\leq 0.25^m \times \prod_{j=1}^{j=m} s_j$.

Lemma

The noise to be introduced by the $\sum_{j \prec i} |b_{\mathbf{x},i} z^2_{\mathbf{x},j}|$ *component is:*

$$\leq \prod_{j=1}^{j=m} s_j \left(0.1^m + 0.001625 \times m\right)$$

A fine-grained analysis (II)

Theorem

For a network of m layers with s_j units in each layer, where $j \in [1, n]$, the amount of Laplacian noise to ensure differential privacy is:

$$\leq 2\left(0.25^m \prod_{q=1}^{q=m} s_q + \prod_{j=1}^{j=m} s_j \times \left(0.1^m + 0.001625 \times m\right)\right)$$

The above analysis shows that the amount of noise to ensure differential privacy is dominated by the number of units in the network.

Concluding Remarks

- We have reported on the usage of differential privacy in neural networks and discussed our preliminary findings in adding Laplacian noise to a low-polynomial representation of the error function.
- We want to underline the following aspects emerging from our preliminary analysis :
  1. The sensitivity basically depends on the topology of the network.
     - Finding a better bound is in our research agenda
  2. Algorithm 1 works by approximating the Hessian to its diagonal components as done by other approaches .
     - We are considering the usage of 2nd order methods (e.g., Hessian Free optimization [5])

References I

[1] Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang.
Deep learning with differential privacy.
In *Proc. of CCS*, pages 308–318. ACM, 2016.

[2] Christopher M Bishop.
Pattern recognition.
Machine Learning, 2006.

[3] Y. Le Cun.
Modeles Connexionistes de l'Apprentissage.
PhD thesis, Universite' de Paris, 1987.

[4] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin.
Exploring strategies for training deep neural networks.
JMLR, 10:1–40, 2009.

[5] James Martens.
Deep learning via hessian-free optimization.
In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

[6] Jun Zhang, Zhenjie Zhang, Xiaokui Xiao, Yin Yang, and Marianne Winslett.
Functional mechanism: regression analysis under differential privacy.
Proceedings of the VLDB Endowment, 5(11):1364–1375, 2012.