Exploring the use of Lipschitz Neural Networks for Automating the Design of Differentially Private Mechanisms

Yonadav Shavit
Harvard University
yonadav@g.harvard.edu

Boriana Gjura
Harvard University
bgjura@g.harvard.edu

ABSTRACT

Designing differentially-private mechanisms that provide both privacy and utility is a challenging task that requires human expertise. Automating mechanism design could ease the adoption of differential privacy in new contexts. In this work, we frame the task of finding a differentially private mechanism as a machine learning problem. Specifically, we construct mechanisms using recently-popularized Lipschitz Neural Networks and outline the foundations for optimizing such mechanisms via gradient descent. We apply this approach to learn mechanisms for several queries on simple datasets, and discuss our results.

CCS CONCEPTS

• **Security and Privacy → Differential Privacy Applications and Implementations**.

KEYWORDS

differential privacy, lipschitz neural networks

1 INTRODUCTION

Designing differentially-private mechanisms is a challenging task that often requires human expertise, especially for novel statistics or non-standard noise requirements. Ideally, it would be great if this could be done automatically: we could define a family of possible release mechanisms with a given level of privacy and a metric for computing the utility of each mechanism, and then have an algorithm search through them automatically to find the best one.

In this spirit, this work frames the task of finding a differentially private mechanism as a machine learning optimization problem, where we are learning the parameters of a release mechanism such that it maximizes the utility of the generated responses while maintaining differential privacy.

Specifically, we will define our DP mechanism using a neural network. The input to the network will be the dataset, concatenated into a vector[1], and the output of the network will be used to generate a response. We introduce two approaches for building a mechanism using neural networks: one based on the Laplace mechanism, and another based on the exponential mechanism.

Crucially, we need to bound the Lipschitz constant of the neural network in order to bound the global sensitivity of the result with respect to the inputted data points. To this end, we leverage Lipschitz Neural Networks [Gouk et al. 2018], which use a reparameterization trick to bound the overall L_1-Lipschitz constant of the neural network with respect to each of the input features.

To make the resulting mechanism output high-utility responses, we need to define the utility of a given differentially private mechanism. Broadly, we frame this in terms of a 'response-loss' L: if for a dataset x the true query answer was a but the given response was b, we say that the response incurs loss $L(x, a, b)$. We define the utility of a differentially private mechanism as the expected response loss over the randomness of the mechanism. This quantity will serve as the loss that our neural network will be trained to minimize. Note that in order to learn a mechanism that is accurate on our private dataset, we will in practice need to train the network on a distribution of datasets that is similar to our private dataset.

We implement our approach to learn mechanisms for several query types on toy datasets, and discuss the results. While the results do not exceed the performance of existing expert-designed DP mechanisms, we identify several observations that merit further study on the use of Lipschitz NNs for differential privacy.

1.1 Related work.

Loss functions are used in [Ghosh et al. 2012] to quantify how well an oblivious mechanism performs in answering differentially-private queries, and derives certain optimality guarantees about the geometric mechanism in restricted scenarios. In our work, we consider general private mechanisms and only assume a practitioner's knowledge of the distribution of datasets.

In [Blocki et al. 2013], the authors define restricted sensitivity of a function by doing the analysis over a restricted class of datasets and then smoothly extending it outside that class. The extension is guaranteed to be differentially private but not necessarily accurate. In our approach we hope that if we train in a sufficiently-broad distribution, the mechanism we learn might generalize outside the distribution, thus in practice yielding comparably better accuracy guarantees outside the training set.

In the direction of testing and enforcing the Lipschitz property, [Jha and Raskhodnikova 2013] describe how to test whether a function over a finite domain is Lipschitz or not. They provide an explicit 'filter' mechanism that given a function f, releases a function g that is $k-$Lipschitz, where k is provided by a distrusted client. Since we are working specifically with neural networks, we are able to enforce this property more simply by normalizing weights.

[1] In this work, we'll focus on single-valued data, though these methods can be extended to vector-valued data.

TPDP '19, Nov '19, London, UK

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 PRIVACY MECHANISMS AS AN OPTIMIZATION PROBLEM

Given a query over a fixed-size dataset, we want to quantify in how 'far' a given response is from the true query answer.

Definition 2.1 (Response-loss). For a dataset $x \in \mathbb{R}^n$ and query $q : \mathbb{R}^n \to \mathcal{R}$, let $L(x, q(p), r)$ denote the loss when the true query result for $p \in X$ is $q(p)$ and the response is r, where $L : \mathbb{R}^n \times \mathcal{R} \times \mathcal{R} \to \mathbb{R}$.

In our experiments, we used both mean squared error and quantile error as response-losses.

We are interested in finding a high-utility differentially private mechanism for a given dataset distribution and a given privacy budget ε. We propose two approaches.

2.1 Adding Laplace noise to an optimized function

The Laplace mechanism adds noise proportional to the sensitivity of the function and the inverse of the desired privacy parameter [Dwork et al. 2006]. To ensure that our sensitivity is bounded, we best approximate the function in a given dataset distribution using a Lipschitz Neural Network (see Sec. 3) and then add appropriate Laplace noise to ensure differential privacy.

Definition 2.2 (Expected response loss of a function). For a given query $q : \mathbb{R}^n \to \mathbb{R}$ and a dataset x chosen from a distribution of dataset $\mathcal{D}$, let $f : \mathbb{R}^n \to \mathbb{R}$ be a function approximating q. We define the *expected response loss*, abbreviated as ERL, of a Laplace mechanism based on this function as

$$ERL(f, q, \mathcal{D}) = \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{z\sim\mathcal{Z}}\left[L\left(x, q(x), f(x) + z\right)\right],$$

where z is the noise variable drawn from the noise distribution $\mathcal{Z}$.

The optimal function f^* in a class of functions $\mathcal{F}$ is defined naturally as the function in that class that incurs the smallest expected response loss. In our experiments, we specify a range of width c where the data comes from. When we use a neural network with Lipschitz constant of k, chosen independently of the data at hand, and add Laplace noise $z \sim Lap\left(\frac{kc}{\varepsilon}\right)$, then z is independent of $f(p)$ and $q(p)$. This gives

$$\begin{aligned}\mathbb{E}[(r + z - q(x))^2] &= \mathbb{E}[(r - q(x))^2] + 2\mathbb{E}[z]\mathbb{E}[r - q(x)] + \mathbb{E}[z^2] \\ &= \mathbb{E}[(r - q(x))^2] + \frac{2k^2c^2}{\varepsilon^2},\end{aligned}$$

which shows that to minimize the expected response loss of the differentially private mechanism, it suffices to minimize

$$\mathbb{E}\left[(r - q(x))^2\right] = \mathbb{E}_{x\sim\mathcal{D}}\left[L\left(x, q(x), f(x)\right)\right]$$

We model the mechanism as $f_\varepsilon + z$, where $f_\varepsilon : \mathbb{R}^n \to \mathbb{R}$ and $z \sim Lap\left(\frac{kc}{\varepsilon}\right)$. Explicitly, the neural network will take as input an $n \times 1$ vector (n 1-dimensional dataset elements) and output a single real value. To bound the sensitivity of the network, we use Lipschitz neural networks, which bound the Lipschitz constant of the neural network.[2] We approximate the expected ERL over draws from the dataset distribution by computing the average ERL on sampled minibatches of datasets drawn from $\mathcal{D}$, and use a gradient descent to minimize the expected response loss.

[2]This can be modified to support variable-sized datasets by making the network a recurrent neural network (RNN).

2.2 Learning the utility function of the exponential mechanism

Alternatively, we can use the exponential mechanism, introduced in [McSherry and Talwar 2007], by having the neural network approximate the utility function. This mechanism is defined with respect to some utility function

$$u : \mathbb{R}^n \times \mathcal{R} \to \mathbb{R}$$

which maps dataset/output pairs to utility scores and measures how 'good' every answer is. The mechanism then makes high quality outputs exponentially more likely at a rate that depends on the sensitivity of the utility function and the privacy parameter, and samples from that distribution.

We again use bounded-constant Lipschitz neural networks to ensure that the utility function has bounded sensitivity. The expected response loss of the utility function is calculated as

$$ERL(u, q, \mathcal{D}) = E_{x\sim\mathcal{D}}\left[\sum_{r\in R} L\left(x, q(x), r\right) \times P_{exp}(r, u, x)\right],$$

where $P_{exp}(r, u, x) = \frac{e^{\frac{\varepsilon}{ck}u(x,r)}}{\sum_{r\in R} e^{\frac{\varepsilon}{ck}u(x,r)}}$, which measures how likely it is that the mechanism will output a response r.

It's important to note that exponential mechanisms are general mechanisms, in that every differentially private algorithm is captured by the exponential mechanism if we choose some appropriate utility function! In particular, assume we have an $\varepsilon-$differentially private algorithm $\mathcal{A} : \mathbb{R}^n \to \mathcal{R}$ for a query $q : \mathbb{R}^n \to \mathcal{R}$. Then there exists a utility function:

$$u(x, r) = \log Pr[\mathcal{A}(x) = r]$$

for every dataset x and $r \in \mathcal{R}$, where the induced exponential mechanism is 2ε differentially private.

Analogous to the previous approach, we model the utility function as a neural network of the form $u_\varepsilon : \mathbb{R}^n \times \mathcal{R} \to \mathbb{R}$.

3 LIPSCHITZ NEURAL NETWORKS

In order to use a neural network in a DP mechanism, we need to bound the sensitivity of the output of the network to each input data-point. Specifically, as long as the data comes from a bounded region $|x - x'| \leq c$ for all neighboring $x, x' \in \mathbb{R}$ for some constant c, then the global sensitivity of a function f with Lipschitz-constant[3] k_f is bounded by $GS(f) = ck_f$.

A neural network can be understood as a composition of a series of sequential functions (matrix multiplications and non-linearities).[4] To compute the Lipschitz constant of a neural network, we rely on the composition of Lipschitz-constrained functions: if f is k_1-Lipschitz and g is k_2-Lipschitz, $(f \circ g)$ is k_1k_2 Lipschitz.

We can compute the Lipschitz constant of an unconstrained NN by computing the constant with respect to each layer, and multiplying these all together to get the overall network's constant. Conveniently, the classic non-linearity ReLU ($h(x) = \max(x, 0)$) is 1-Lipschitz. However, the weight matrix multiplications turn out to be another matter. For an unconstrained NN, this Lipschitz constant

[3]For the rest of this paper, 'Lipschitz-ness' refers to L_1-Lipschitz-ness.

[4]In this work, we will consider only fully-connected networks, though extensions extensions for convolutional and residual networks exist [Gouk et al. 2018].

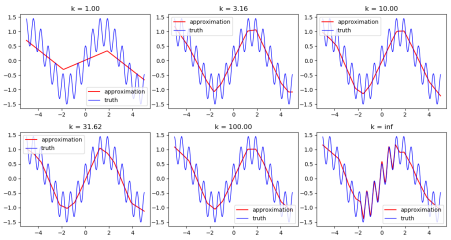

Figure 1: Lipschitz NNs trained to minimize predictive MSE on the function $f(x) = \sin x + \frac{1}{2}\cos(10x)$, with increasing Lipschitz parameters. $f(\cdot)$ is interesting because it is composed of two signals, one low-frequency (at most $k = 1$ Lipschitz) and another high-frequency (at most $k = 10$ Lipschitz). As is clear, the Lipschitz NNs fail to recover the optimal k-Lipschitz fit for a given k, though increasing k does result in a better fit.

may end up being very large; see [Gouk et al. 2018] for a deeper exploration.

The idea behind Lipschitz NNs [Gouk et al. 2018] is to bound the Lipschitz constant of each weight matrix. We enforce a similar constraint by reparameterizing the weight matrices. To force a given matrix multiplication to be k-Lipschitz, we store our original parameters as a weight matrix W, but instead of multiplying the layer input z by W, we multiply z by a reparameterized matrix W':

$$W'_{ij} = \frac{W_{ij}}{\max\left(1, \frac{\|W_{\cdot i}\|_1}{k}\right)} \tag{1}$$

This parameterization ensures that $\phi^{W'}$ is k-Lipschitz by uniformly squeezing the weights in each column such that they always absolute-sum to k. These operations (max, $\|\cdot\|$) can be straightforwardly incorporated into a neural network, and their gradient computed via automatic differentiation.

Thus, to train a b-layer neural network with an overall Lipschitz constant of k with ReLu non-linearities, we need only apply this reparameterization to each of the layers, forcing each individually to be $\sqrt[b]{k}$-Lipschitz.

In theory, Lipschitz NNs offer a straightforward means for learning bounded-Lipschitz approximators using traditional unconstrained optimization techniques. In practice we have found that the constrained space of valid networks meaningfully complicates optimization, a conclusion shared by other explorations of Lipschitz NNs [Gouk et al. 2018], which found that k-Lipschitz NNs were only able to recover functions with Lipschitz constants substantially smaller than k. See Fig. 1 for an example of their performance on a toy problem.

4 METHODOLOGY

4.1 Data Distributions

To explore the basic feasibility of our mechanism-learning approaches, we created a simple distribution over datasets, where each dataset consisted of n single-valued data points. In all our experiments, $n = 100$. We concatenated the points in each dataset

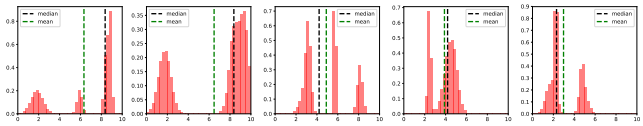

Figure 2: Draws from the mixture of truncated Gaussians distribution used to train the neural networks, along with their means and medians.

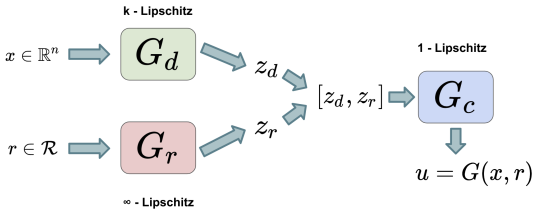

Figure 3: Architecture of the exponential-method network.

to form dataset vectors x.[5] We focus on learning mechanisms for simple statistics: the mean, variance, median, and 10th percentile of the dataset.

In all our experiments, we drew each new dataset from a hierarchical model that parameterized a mixture of truncated Gaussians on the interval $[0, 10]$. The parameters of each truncated Gaussian (mean, variance, and relative weight) were randomly drawn for each new dataset; for complete details, see forthcoming work. Our training datasets used 4-Gaussian mixtures. We also tested our learned mechanisms' generalization on 2-Gaussian mixtures.

4.2 Neural Network Architecture and Training

All our neural networks used ReLU activations, and 100-node layers. For the exponential method, we constructed the discrete set of possible responses $\mathcal{R}$ as 50 points uniformly distributed on the interval $[0, 10]$.

For the Laplace mechanism, we implement the neural network as a standard two-layer NN. Our utility function neural network (UNN) was built by combining three sub-networks G_d, G_r, and G_c. G_d is a two-layer network that takes in the dataset $x \in \mathbb{R}^n$, and converts it into a latent representation $z_d = G_d(x)$. If the overall network is required to be k-Lipschitz, we force G_d to be k-Lipschitz. Separately, the potential response r is processed by the one-layer sub-network G_r to yield a latent representation of the response $z_r = G_r(r)$. Finally, these two results are concatenated and fed into the final two-layer subnetwork G_c, such that the output $G_c([z_d, z_r]) = u$. G_c must be 1-Lipschitz to preserve the fact that the overall network remains k-Lipschitz w.r.t. x (no such requirement exists for r). A diagram visualizing the setup is available in Fig. 3.

We used two different response losses:

- classical squared error $(q(x) - r)^2$,
- quantile error $\sum_{p \in x} \mathbb{1}[min(q(x), r) < p < max(q(x), r)]$, which measures the number of points between the true and released statistic, where $\mathbb{1}$ is the indicator operator.

[5] Another sensible strategy would be to have the vector x be a histogram representation of the different bins in the data range; we leave this alternative for future work.

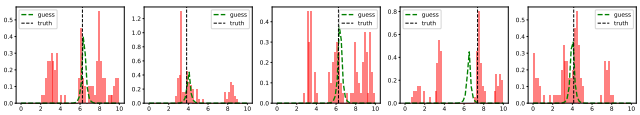

Figure 4: The probability mass of responses (in green) from a median-releasing mechanism with Lipschitz constant $k = 10$ trained using quantile loss, on a series of fresh datasets (pink), along with the true median (black).

We only use the quantile error when learning the utility function for the exponential mechanism, for the variance and 10th-percentile queries. In all other experiments, we use the mean squared error.

Before training, we need to specify the overall Lipschitz constant k we want to enforce in the network, and thus the global sensitivity of the network. For the exponential-mechanism NN, we also need to specify the privacy loss parameter ε ahead of time[6].

During training, we compute the average expected response loss on 32-dataset batches (sampled from 10^4 pre-drawn samples), and then updated the parameters by computing the gradient of the parameters w.r.t. this loss using the Adam update rule [Kingma and Ba 2014]. In practice, the Lipschitz NNs were sensitive to initialization; if the loss did not improve after the first epoch, we reinitialized, which eventually tended to converge to a better solution.

5 EXPERIMENTS

In Figure 4, we visualize the performance of an exponential mechanism trained to release the median. We can see that the mechanism consistently approximates the true median - when it misses, it does so only in a direction with relatively small quantile loss. The mechanism appears to have automatically learned to generated symmetric noise; many of our training runs exhibited this behavior, though some learned more complicated response distributions.

We plot the performance of our approach in learning mechanisms for several queries in Figure 5. We can see several clear trends.

First, when the Lipschitz constant provided to the network is too small, the network converges to the data-blind baseline. This is roughly what we'd expect of NNs that are over-constrained. In general, our experiments show that we need to take k larger than the actual Lipschitz constant of the ideal expert-designed mechanism.

Second, in many of the plots, the expected response loss doesn't increase even under extremely high Lipschitz constants (and thus global sensitivity values). This is counterintuitive; we'd expect that as e.g. the Laplace mechanism's injected noise increased, the utility would fall. While this happens a little (the best loss usually doesn't correspond to the largest Lipschitz constant), in most instances the performance remains roughly the same. Why increasing noise does in some cases not substantially increase expected response loss is a subject for further investigation.

Finally, we observe that while the train and test dataset distributions tend to have different ERLs, mechanisms that perform better

[6]Training an exponential-mechanism NN requires fixing ε, as this affects the weighting in the exponential mechanism and thus the expected response loss of the current mechanism. There may be expected response loss functions for which the ordering of utility functions by expected response loss is preserved as ε increases, and thus the optimization generalizes across values of ε; this is a subject for future work.

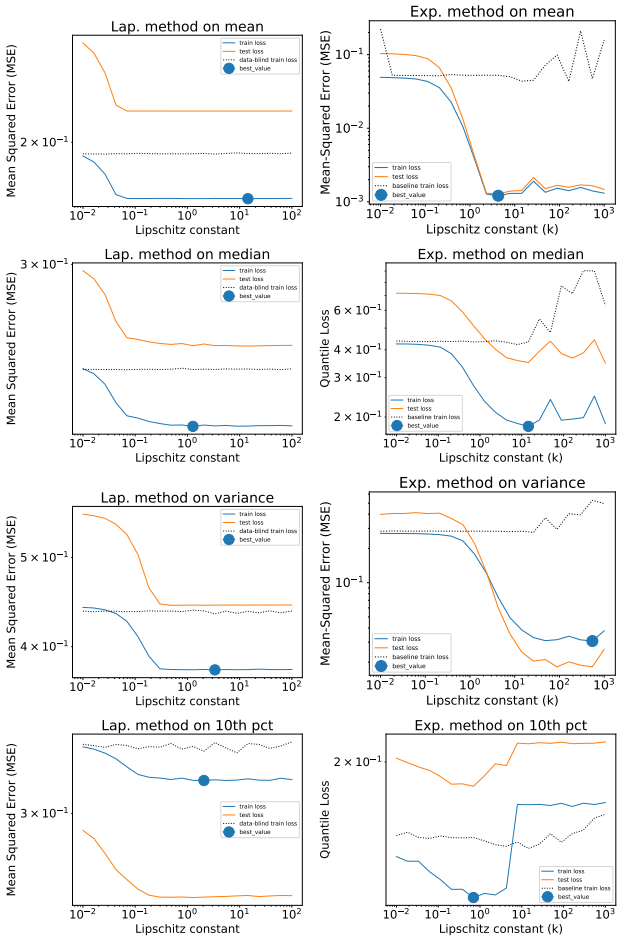

Figure 5: The expected response loss of the Laplace (left) and exponential (right) methods, averaged over 5 training runs. The parameter k with the best training loss is designated with a blue circle. We also as a baseline plot the performance of the same mechanism, trained without seeing the data and thus relying only on the properties of the dataset distribution (dotted black). We plot the learned mechanism's performance on the training dataset distribution (4-Gaussian mixtures, blue), and its generalization to a different distribution (2-Gaussian mixtures, orange).

on the training set tend to also work better on the test set. This is heartening, as it suggests that mechanisms trained on one distribution of datasets may learn a generally-useful way of computing the given query, and may generalize even if the true private dataset does not exactly look like one of the training dataset draws.

Acknowledgements. We thank Salil Vadhan, James Honaker and Jayshree Sarathy for their valuable feedback throughout this project.

REFERENCES

Jeremiah Blocki, Avrim Blum, Anupam Datta, and Or Sheffet. 2013. Differentially private data analysis of social networks via restricted sensitivity. In *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. ACM, 87–96.

Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 486–503.

Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. 2012. Universally utility-maximizing privacy mechanisms. *SIAM J. Comput.* 41, 6 (2012), 1673–1693.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. 2018. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368* (2018).

Madhav Jha and Sofya Raskhodnikova. 2013. Testing and reconstruction of Lipschitz functions with applications to data privacy. *SIAM J. Comput.* 42, 2 (2013), 700–731.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Frank McSherry and Kunal Talwar. 2007. Mechanism design via differential privacy. In *null*. IEEE, 94–103.