Model Reconstruction from Model Explanations

Smitha Milli Ludwig Schmidt Anca D. Dragan Moritz Hardt
University of California, Berkeley
{smilli,ludwig,anca,hardt}@berkeley.edu

Abstract

We show through theory and experiment that gradient-based explanations of a model quickly reveal the model itself. Our results speak to a tension between the desire to keep a proprietary model secret and the ability to offer model explanations.

On the theoretical side, we give an algorithm that provably learns a two-layer ReLU network in a setting where the algorithm may query the gradient of the model with respect to chosen inputs. The number of queries is independent of the dimension and nearly optimal in its dependence on the model size. Of interest not only from a learning-theoretic perspective, this result highlights the power of gradients rather than labels as a learning primitive.

Complementing our theory, we give effective heuristics for reconstructing models from gradient explanations that are orders of magnitude more query-efficient than reconstruction attacks relying on prediction interfaces.

1 Introduction

Commercial machine learning models increasingly support consequential decisions in numerous domains including medical diagnosis, employment, and criminal justice. In such applications, there is now growing demand for methods that explain a model's decision. The secrecy of a model strongly fuels this demand.

At the same time, there are a number of valid reasons a company might wish to keep its machine learning models secret. The competitive value of the product is one consideration. Revealed models may also be easier to *game*, resulting in diminished predictive power [6, 8]. Yet another reason is that the model might leak sensitive information about the data it was trained on [12, 5].

In this work, we point out a tension between keeping a model secret and explaining its decisions. We show that a popular class of existing methods to explain a model's decision quickly reveals the model itself in what is typically an undesired side effect.

Numerous explanation methods have been proposed in an ongoing line of research. Among these methods, *saliency maps* are a widespread technique to highlight characteristics of an input deemed relevant for the prediction of a model. The most basic saliency map is to compute the gradient of the model with respect to a chosen input [2, 14] and numerous variants add different transformations to the raw gradients leading to some disagreement over which of these heuristics is preferable in what context [20, 16, 15, 17]. Abstracting away from these implementation details, we focus on

reconstructing models given the basic underlying primitive, which is gradients of the model with respect to its inputs.

1.1 Our contributions

Our contributions are twofold, spanning both a theoretical and experimental component.

Learning from input gradients. On the theoretical side, we introduce a model of learning from *input gradient queries*. In this model, a learning algorithm can observe gradients of an unknown model at chosen query inputs. This model turns out to be rich in its mathematical structure and connections to standard learning models, such as learning from *membership queries*, in which the learner can request the model's prediction at a given input.

In our setting, since the gradient provides more information than a single label, there is hope that learning algorithms can get by with far fewer queries. We prove that this is indeed the case. To build up intuition with a simple example, consider a linear model $f(x) = \langle w, x \rangle$, specified by a weight vector $w \in \mathbb{R}^d$. The gradient of the model with respect to any input x is just equal to the model parameters $w = \nabla_x f(x)$. Thus, we can learn a linear model from a single input gradient query.

Going beyond linear models, We analyze two-layer neural networks with ReLU transitions of the form $f(x) = \langle w, \mathrm{ReLU}(Ax) \rangle$ where $A \in \mathbb{R}^{h \times d}$. Here, $\mathrm{ReLU}(u) = \max\{u, 0\}$ applies coordinate-wise to a vector. The problem of learning such networks has received much renewed interest in the last few years as it poses a non-trivial challenge en route to deeper non-linear models [11, 18, 21].

Theorem 1 (informal)**.** *Assuming the rows of the weight matrix A are linearly independent, our algorithm recovers a functionally equivalent model from $O(h \log h)$ input gradient queries and function evaluations with high probability.*

The $O(h \log h)$ queries our theorem requires is optimal to within a logarithmic factor, since it takes $dh + h$ parameters to specify the model, and each query reveals only $O(d)$ numbers. Furthermore, compared to membership queries, gradient queries reduce the number of queries needed by approximately a factor of d, since it takes $\Omega(dh)$ membership queries to specify the model.

Although our algorithm enjoys an intuitive geometric interpretation, the proof requires a delicate argument, as well as an anti-concentration bound that may be useful independently.

Practical reconstruction methods. In a second step, we explore practically effective heuristics to reconstruct a model from input gradient queries. Our experiments show that reconstructing models from explanations is not just a theoretical concern. If a company were to provide an *explanation API* with standard saliency maps, it would effectively give up the underlying model, which it may not be willing to do for reasons mentioned above. This situation parallels an ongoing investigation on *stealing models from prediction APIs* [19]. However, as our results show, with explanation APIs we need far fewer queries, thus greatly exacerbating the threat of model leakage.

Our experiments focus on a heuristic for learning from input query gradients. While our theoretical method is specific to two-layer networks, our heuristic is agnostic to the shape of the target model. At the outset, our heuristic simply queries a number of input gradients and fits a model against the observed gradients in much the same way we would fit a model against labels. We find that this

heuristic reduces the number of queries needed to learn models on MNIST and CIFAR10 by orders of magnitude, even in cases where the model class is unknown or the data distribution is unknown.

Conclusion. Our work demonstrates that establishing usable explanation methods for machine learning models faces another hurdle in commercial applications. Whatever criteria of explanation quality we choose must be weighed against the risk of model leakage resulting from the method at hand. We see our work as only a first step in this new direction that raises many intriguing questions.

Does our theoretical result extend to depth-3 networks? Ignoring computational efficiency, what is the optimal query complexity? In particular, can we learn a k-layer ReLU network with h units at each layer from only $\tilde{O}(kh)$ queries? Can we design useful explanation methods resilient to model reconstruction attacks? Although a natural and important question to ask, there is no currently agreed upon measure of explanation quality, which makes it difficult to formally study this trade-off.

2 Problem statement: reconstructing a two-layer ReLU network

We consider the problem of finding a classifier $\hat{f}$ identical to an unknown classifier f when given access to membership and gradient queries. That is, we assume access to an oracle that given a query input x returns the evaluation of f at x and the gradient $\nabla_x f(x)$ of f with respect to x.

We analyze the case where the function $f : \mathbb{R}^d \to \mathbb{R}$ is represented by a one hidden-layer neural network with ReLU activations:

$$f(x) = \sum_{i=1}^{h} w_i \max(A_i^\top x, 0)\,. \tag{1}$$

Here, the model parameters are $A \in \mathbb{R}^{h \times d}$ and $w \in \mathbb{R}^h$. We use A_i to denote the i-th row of A. We make the following three assumptions:

1. The rows $A_1, \dots A_h$ are unit vectors.
2. No two rows A_i and A_j with $i \neq j$ are collinear, i.e., $\langle A_i, A_j \rangle \leq 1 - c$ for some $c > 0$.
3. The rows $A_1, \dots, A_h$ are linearly independent.

The first two assumptions are without loss of generality, as they follow from simple reparameterizations of the network that involve scaling w or A or reducing the hidden dimension.

Our main result is the following theorem, which shows that our sample complexity for learning the function with gradient queries has no dependence on the input dimension d.

Theorem 1. *Suppose, the unknown function f satisfies our assumptions. Then, with probability $1 - \delta$, Algorithm 1 succeeds to find a function $\hat{f}$ such that $\hat{f} = f$ in $O(h \log \frac{h}{\delta})$ queries. If the Algorithm fails, then it notifies of the failure.*

Section 3 contains our algorithm and proof of correctness. In Appendix C we show that our algorithm can also be converted to one which learns the function f in $O(dh \log \frac{h}{\delta})$ membership queries by using membership queries to approximate gradients of f.

3 Algorithm

Before we formally introduce our algorithm, we briefly provide some high-level intuition. First, note that we can express our two-layer ReLU networks as

$$f(x) = \sum_{i=1}^{h} g(x)_i w_i A_i^\top x\,, \tag{2}$$

where $g(x) = \mathbb{I}\{Ax \geq 0\}$. The *separating hyperplanes* defined by the normal vectors $A_1, \dots A_h$ split the input space into cells represented by the possible values of $g(x)$. Within each such cell, the function f is linear. See Figure 1 for an example visualization of these cells.

Our algorithm can be separated into two steps. First, we find the separating hyperplanes of f. In particular, we recover unsigned, weighted normal vectors $w_i A_i$ or $-w_i A_i$ for $i \in [h]$. The second step then recovers the sign information for these normal vectors. More precisely, the two steps are the following:

1. Recover a matrix $Z \in \mathbb{R}^{h \times d}$ such that $Z_{p(i)} = w_i A_i$ or $Z_{p(i)} = -w_i A_i$ for some permutation p of $[h]$. (Algorithm 1a)
2. Recover a vector $s \in \{-1, 0, 1\}^{2h}$ such that $f(x) = \begin{bmatrix}\max(Zx, 0)^\top & \max(-Zx, 0)^\top\end{bmatrix} s$. (Algorithm 1b)

Together, the matrix Z and vector s identify the function f. We analyze the first step in Section 3.1 and the second step in Section 3.2.

Algorithm 1: Recovery of f

Function *learnModel(h, ϵ, l)*:
  $Z \leftarrow$ *recoverZ*(h, ϵ, l)
  $s \leftarrow$ *recoverS*(Z)
  return Z, s

3.1 Step one: recovering the separating hyperplanes

Algorithm 1a finds the separating hyperplanes by exploiting the structure of the gradient of f:

$$\nabla f(x) = \sum_{i=1}^{h} g(x)_i w_i A_i\,,$$

where $g(x) = \mathbb{I}\{Ax \geq 0\}$ as before. Note that points within the same cell have the same gradient. So if we find two points x and y with different gradients, we know at least one separating hyperplane must be between x and y. Moreover, if the points x and y are sufficiently close to each other, then it is likely that there is only one separating hyperplane between them. In that case, we can then use the difference of gradients to recover a hyperplane (up to signs). This is because each gradient is simply a sum of a subset of $\{w_i A_i\}_{i=1}^h$, and so the difference $\nabla f(y) - \nabla f(x)$ is equal to either $w_i A_i$ or $-w_i A_i$ for some $i \in [h]$.

In this way, Algorithm 1a isolates changes in the gradient of f to recover $w_i A_i$ up to a sign for every $i \in [h]$. Figure 1 provides an illustrated explanation of the algorithm, which we briefly sketch below:

Algorithm 1a: Recovery of Z

Function *recoverZ(h, ϵ, l)*:
  Pick $u, v \sim \mathcal{N}(0, I_d)$ and let $Z \in \mathbb{R}^{h \times d}$
  $t_l, t_r \leftarrow -l, l$
  for $i = 1, \dots h$ **do**
    $Z_i, t_l \leftarrow$ binarySearch(t_l, t_r, ϵ)
  return Z
Function *binarySearch(t_l, t_r, ϵ)*:
  while $t_l \leq t_r$ **do**
    $t_m \leftarrow (t_l + t_r)/2$
    $x_l \leftarrow u + t_l v, \quad x_m \leftarrow u + t_m v, \quad x_r \leftarrow u + t_r v$
    if $t_r - t_l \leq \epsilon$ **then**
      return $\nabla f(x_r) - \nabla f(x_l),\ t_r$
    if $\|\nabla f(x_l) - \nabla f(x_m)\|_2 > 0$ **then**
      $t_r \leftarrow t_m$
    else if $\|\nabla f(x_m) - \nabla f(x_r)\|_2 > 0$ **then**
      $t_l \leftarrow t_m$
    throw Failure
  throw Failure

Algorithm 1b: Recovery of s

Function *recoverS(Z)*:
  Pick $X \in \mathbb{R}^{d \times h}$ such that $\nabla f(x_1) = \cdots = \nabla f(x_h)$ and $\text{Rank}(ZX) = h$. (See Appendix B)
  $M \leftarrow \begin{bmatrix} \max(ZX, 0)^\top & \max(-ZX, 0)^\top \\ \max(-ZX, 0)^\top & \max(ZX, 0)^\top \end{bmatrix}$
  Solve for $s \in \mathbb{R}^{2h}$ such that $Ms = [f(x_1), \dots f(x_h), f(-x_1), \dots f(-x_h)]$
  return s

1. Pick $u, v \sim \mathcal{N}(0, I_d)$.
2. Run a binary search with resolution ϵ along a portion of the line segment between $u - lv$ and $u + lv$ for some $l \in \mathbb{R}$ to find two points x_l and x_r that are sufficiently close ($\|x_r - x_l\|_2 \leq \epsilon \|v\|_2$), but have differing gradients. Add $\nabla f(x_r) - \nabla f(x_l)$ as a row to the matrix Z. With high probability, $\nabla f(x_r) - \nabla f(x_l)$ is equal to $w_i A_i$ for some $i \in [h]$.
3. Repeat Step (2) h times to recover all rows $w_i A_i$ up to their sign, which become the rows of the matrix Z.

The proof of correctness relies on showing that with high probability, the following two events hold: (i) The points at which the gradient of f changes are spaced sufficiently far apart. (ii) The same gradient change points are within some line segment of u and v that is not too big. The change points can then be found with a binary search that is bounded within a range that is not too large and uses step sizes that are not too small. In the next lemma, we prove correctness of the binary

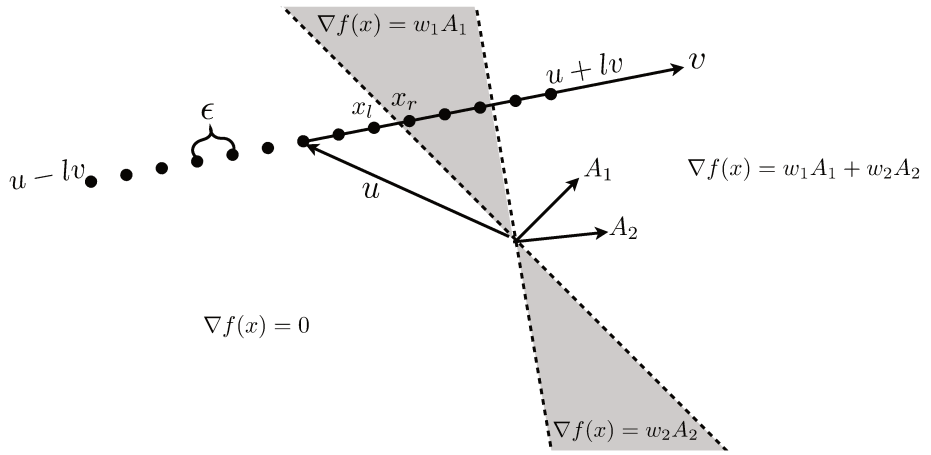

Figure 1: An illustration of Algorithm 1a when the input domain of the function f is $\mathbb{R}^2$ and the hidden dimension h is equal to two. The two hyperplanes with normal vector A_1 and A_2 separate the input space into four cells where the gradient of f is constant. Algorithm 1a picks two random vectors u and v and searches for a change in the gradient of f using a binary search along a line segment between u and v. When two points are found that are sufficiently close, but have differing gradients, then the difference in their gradients is added as a row to the recovered matrix Z. For example, $\nabla f(x_r) - \nabla f(x_l) = w_1 A_1$ is added to Z. By running the binary search h times, Algorithm 1a recovers $w_i A_i$ up to a sign for all $i \in [h]$.

search given that the change points are spaced appropriately.

Lemma 1. *Let $u, v \in \mathbb{R}^d$ be such that $\langle A_i, v \rangle \neq 0$ for all $i \in [h]$. For each $i \in [h]$, also let $t_i \in \mathbb{R}$ be such that $\langle A_i, u + t_i v \rangle = 0$. If for all $i, j \neq i$ we have $|t_i - t_j| \geq \epsilon$ and $|t_i| \leq l$, then Algorithm 1a returns a matrix $Z \in \mathbb{R}^{h \times d}$ such that $Z_{p(i)} = w_i A_i$ or $Z_{p(i)} = -w_i A_i$ for some permutation p of $[h]$.*

Proof. Let $k_1, \ldots, k_h$ be the indices such that $t_{k_1} < t_{k_2} < \cdots < t_{k_h}$. To prove the lemma we will show that on the i-th call to *binarySearch*, either $-w_{k_i} A_{k_i}$ or $w_{k_i} A_{k_i}$ is added as a row to matrix Z.

First, we make the following assumption, which we will later prove: assume that $t_{k_i} = \min_{j : t_j \geq t_l^{(i)}} t_j$ where $t_l^{(i)}$ is the value of the variable t_l at the start of the i-th call to *binarySearch*. Given this assumption, the i-th call to *binarySearch* adds $-w_{k_i} A_{k_i}$ or $w_{k_i} A_{k_i}$ to the matrix Z. To see this, note that on each iteration of the while loop in *binarySearch* either the variable t_l increases or the variable t_r decreases, and thus *binarySearch* always terminates. However, t_l dose not increase past t_{k_i} and t_r does not decrease past t_{k_i}. So, when the condition for termination of the while loop is met we have $|t_l - t_r| \leq \epsilon$, $t_l \leq t_{k_i}$, and $t_r \geq t_{k_i}$. Since $|t_{k_j} - t_{k_i}| \geq \epsilon$ for all $j \neq i$, the row $\nabla f(t_r) - \nabla f(t_l)$ returned by *binarySearch* is equal to either $w_{k_i} A_{k_i}$ or $-w_{k_i} A_{k_i}$.

Now we revisit the assumption that $t_{k_i} = \min_{j : t_j \geq t_l^{(i)}} t_j$. We prove the assumption by induction. The base case $i = 0$ is clearly true: $t_{k_1} = \min_j t_j = \min_{j : t_j \geq t_l^{(i)}} t_j$ because $t_l^{(1)} = -l$ and $-l \leq t_{k_1} < t_{k_2} < t_{k_l} \leq l$. On the $(i+1)$-th call to *binarySearch* the variable t_l is set to the value of t_r when the i-th call to *binarySearch* terminated. When the i-th call to *binarySearch* finishes, the value of the variable t_r is above $\min_{j : t_j \geq t_l^{(i)}} t_j = t_{k_i}$, but less than $t_{k_{i+1}}$. Thus, $t_{k_{i+1}} = \min_{j : t_j \geq t_l^{(i+1)}} t_j$.

Therefore, the returned matrix Z is such that $Z_{p(i)} = w_i A_i$ or $Z_{p(i)} = -w_i A_i$ where the permutation p of $[h]$ is defined by $p(i) = j$ where $k_j = i$. □

The next two lemmas (proved in Appendix A) establish the necessary anti-concentration and concentration bounds for showing that the change points are spaced sufficiently far apart (Lemma 2), but still within some line segment of u and v that is not too big (Lemma 3).

Lemma 2. *Let $a, b \in \mathcal{S}^{d-1}$ be unit vectors such that $|\langle a, b\rangle| \leq 1 - c$ for some scalar $c \in [0, 1]$. Suppose we pick random vectors $u, v \sim \mathcal{N}(0, I_d)$. Let $t_1, t_2 \in \mathbb{R}$ be scalars such that $\langle a, u + t_1 v\rangle = 0$ and $\langle b, u + t_2 v\rangle = 0$.[1] Then,*

$$P(|t_1 - t_2| \leq \epsilon) \leq 3^{\frac{4}{3}} \left(\frac{\epsilon}{c}\right)^{\frac{2}{3}}.$$

Lemma 3. *Let $a \in \mathcal{S}^{d-1}$ be a unit vector. Suppose we pick random vectors $u, v \sim \mathcal{N}(0, 1)$. Let $t \in \mathbb{R}$ be the value such that $\langle a, u + tv\rangle = 0$. Then,*

$$P(|t| \geq l) \leq \frac{2}{\pi l}.$$

Finally, the proof of our main theorem for Algorithm 1a follows by combining the probabilistic guarantees of Lemmas 2 and 3 with the deterministic proof of correctness in Lemma 1.

Theorem 2. *With probability $1 - \delta$, Algorithm 1a succeeds in $O(h \log \frac{h}{\delta})$ queries. If the Algorithm succeeds, it returns a matrix $Z \in \mathbb{R}^{h \times d}$ such that $Z_{p(i)} = w_i A_i$ or $Z_{p(i)} = -w_i A_i$ for some permutation p of $[h]$. If the Algorithm fails, then it notifies of the failure.*

Proof. By Lemma 1, if $|t_i - t_j|$ and $|t_i| \leq l$ for all i and $j \neq i$, then Algorithm 1a succeeds. The probability of this event can be lower-bounded as the following.

$$\begin{aligned}
&P(\forall i, j \neq i : |t_i - t_j| \geq \epsilon,\, |t_i| \leq l) \\
&\geq 1 - \sum_{i=1}^{h} \sum_{j \neq i} P(|t_i - t_j| \leq \epsilon) - \sum_{i=1}^{h} P(|t_i| \geq l) && \text{(Union bound)} \\
&\geq 1 - 3^{\frac{4}{3}} \left(\frac{\epsilon}{c}\right)^{\frac{2}{3}} h^2 - \sum_{i=1}^{h} P(|t_i| \geq l) && \text{(Lemma 2)} \\
&\geq 1 - 3^{\frac{4}{3}} \left(\frac{\epsilon}{c}\right)^{\frac{2}{3}} h^2 - \frac{2}{\pi l} h && \text{(Lemma 3)}
\end{aligned}$$

Let $\delta = 3^{\frac{4}{3}} \left(\frac{\epsilon}{c}\right)^{\frac{2}{3}} h^2 - \frac{2}{\pi l} h$. Set $l = h^2$. Then, solving for ϵ yields $\epsilon = 3^{-2} c \frac{(\delta + \frac{2\pi}{h})^{\frac{3}{2}}}{h^3}$. So, Algorithm 1a succeeds with probability $1 - \delta$ and uses less than $h \log\left(\frac{l}{\epsilon}\right)$ queries, which is upper bounded as the

[1] With probability one such a t exists.

following.

$$
\begin{aligned}
h \log\left(\frac{l}{\epsilon}\right) &= h \log\left(\frac{h^2}{3^{-2} c \frac{(\delta + \frac{2\pi}{h})^{\frac{3}{2}}}{h^3}}\right) \\
&= 5h \log\left(\frac{h}{3^{-2} c (\delta + \frac{2\pi}{h})^{\frac{3}{2}}}\right) \\
&\leq O\left(h \log \frac{h}{\delta}\right)
\end{aligned}
$$

□

3.2 Step two: recovering the signs of the normal vectors

Algorithm 1a recovers unsigned, weighted normal vectors: $w_i A_i$ or $-w_i A_i$ for $i \in [h]$. But to identify the function f, we still need the sign of these vectors. In Algorithm 1b, we recover a vector $s \in \{-1, 0, 1\}^{2h}$ that encodes this sign information. Precisely, Algorithm 1b returns a vector s such that

$$
f(x) = \begin{bmatrix} \max(Zx, 0)^\top & \max(-Zx, 0)^\top \end{bmatrix} s \,.
$$

where

$$
s_i = \begin{cases} \operatorname{sgn}(w_i) & 1 \leq i \leq h,\ z_i = |w_i| A_i \\ 0 & h+1 \leq i \leq 2h,\ z_i = |w_i| A_i \\ 0 & 1 \leq i \leq h,\ z_i = -|w_i| A_i \\ \operatorname{sgn}(w_i) & h+1 \leq i \leq 2h,\ z_i = -|w_i| A_i \end{cases} .
$$

It is clear that if Algorithm 1b returns the vector s, then the function f is identified. Algorithm 1b solves $2h$ linear equations to determine the vector s. To prove correctness of Algorithm 1b, we show that the $2h$ query points picked in the algorithm lead to a determined set of linear equations.

Lemma 4. *Let $Z \in \mathbb{R}^{h \times d}$ be a matrix such that $Z_{p(i)} = w_i A_i$ or $Z_{p(i)} = -w_i A_i$ for a permutation p of $[h]$. Let x_i denote the i-th column of a matrix $X \in \mathbb{R}^{d \times h}$. Suppose $\nabla f(x_1) = \cdots = \nabla f(x_h)$, $(ZX)_{ij} \neq 0$, and $Rank(ZX) = h$ for all $i, j \in [h]$. Then, the $2h \times 2h$ matrix defined as*

$$
M = \begin{bmatrix} \max(ZX, 0)^\top & \max(-ZX, 0)^\top \\ \max(-ZX, 0)^\top & \max(ZX, 0)^\top \end{bmatrix} \tag{3}
$$

is full-rank.

Proof. Since $\nabla f(x_1) = \cdots = \nabla f(x_h)$ and $(ZX)_{ij} \neq 0$, we know $\mathbb{I}\{Zx_1 > 0\} = \cdots = \mathbb{I}\{Zx_h > 0\}$, and that we could always negate rows of the matrix Z so that $\mathbf{1} = \mathbb{I}\{Zx_1 > 0\} = \cdots = \mathbb{I}\{Zx_h > 0\}$.

Thus, we can assume without loss of generality that $(ZX)_{ij} > 0$ for all $i, j \in [h]$. Then, the matrix M can be expressed as the following.

$$M = \begin{bmatrix} (ZX)^\top & 0 \\ 0 & (ZX)^\top \end{bmatrix}$$

The determinant of the matrix is $\det(M) = \det((ZX)^2 - 0) = \det^2(ZX) > 0$. Thus, M is a full-rank matrix. □

In Appendix B we describe a simple linear program that can be used to pick a matrix X that satisfies the conditions of the above Lemma 4. Since Algorithm 1b picks such a matrix X, Lemma 4 immediately implies our main theorem proving correctness of Algorithm 1b.

Theorem 3. *If Algorithm 1b is given a matrix $Z \in \mathbb{R}^{h \times d}$ such that $Z_{p(i)} = w_i A_i$ or $Z_{p(i)} = -w_i A_i$ for a permutation p of $[h]$, then it returns a vector $s \in \{-1, 0, 1\}^{2h}$ such that the function f is equal to $f(x) = \begin{bmatrix} \max(Zx, 0)^\top & \max(-Zx, 0)^\top \end{bmatrix} s$.*

Proof. Algorithm 1b uses $2h$ queries to construct a $X \in \mathbb{R}^{2h \times 2h}$ that satisfies the conditions of Lemma 4. Thus, the resulting set of $2h$ linear equations are determined and Algorithm 1b returns the unique vector s corresponding to its solution. □

Together, Theorem 2 proving correctness of Algorithm 1a and Theorem 3 proving correctness of 1b imply our main Theorem 1 that proves correctness of Algorithm 1.

Theorem 1. *Suppose the unknown function f satisfies the assumptions in Section 2. Then, with probability $1 - \delta$, Algorithm 1 succeeds to find a function $\hat{f}$ such that $\hat{f} = f$ in $O(h \log \frac{h}{\delta})$ queries. If the Algorithm fails, then it notifies of the failure.*

Proof. By Theorem 2, with probability $1 - \delta$, Algorithm 1a returns a matrix Z that satisfies the conditions of Theorem 3 in $O(h \log \frac{h}{\delta})$ queries. By Theorem 3, Algorithm 1b then returns a vector s such that $f(x) = \begin{bmatrix} \max(Zx, 0)^\top & \max(-Zx, 0)^\top \end{bmatrix} s$ in $O(h)$ queries. Thus, overall Algorithm 1 succeeds with probability $1 - \delta$ in $O(h \log h)$ queries. □

4 Experimental design

While our theoretical analysis provides insight into the power of gradient queries over membership queries, it is specific to a two-layer ReLU network. To complement our theory, we also experimentally investigate the impact of gradients on reconstructing models used in practice.

In order to compare to reconstructing with membership queries alone, our method for learning with gradients is a modification of a simple heuristic used to reconstruct models from membership queries: training a new classifier $\hat{f}$ to match the outputs of f [19, 10]. When we have access to gradients we can also train the classifier $\hat{f}$ to match the gradients of f by minimizing a loss on the gradients: $\ell_G(x) = \|\nabla f(x) - \nabla \hat{f}(x)\|_2^2$. Furthermore, we can trade off between the gradient loss ℓ_G with a loss on the membership queries $\lambda \ell_M$ to create a joint loss $\ell_J(x) = \ell_G(x) + \lambda \ell_M(x)$.

We test how gradient queries help by measuring the accuracy of $\hat{f}$ when trained using $\ell_J(x)$ versus when trained only on the membership query loss, $\ell_M(x)$. In our experiments $\ell_M(x)$ is the cross-entropy loss between $f(x)$ and $\hat{f}(x)$. Next, we describe our experimental design in detail.

Manipulated factors. We manipulate three independent variables. First, we manipulate the *type of query*. We test membership only queries as well as membership and gradients. Further, because in practice explanations often provide a processed version of the gradients, instead of the raw gradients, we also test membership and gradients processed with SmoothGrad, a saliency map denoising technique [15]. Instead of returning the raw gradient $\nabla f(x)$, SmoothGrad returns an average of gradients around the input x: $\widetilde{\nabla} f(x) = \sum_{i=1}^{N} \frac{1}{N} \nabla f(x + z_i)$ where $z_i \sim \mathcal{N}(0, \sigma I)$.

Second, we manipulate the *complexity of the task* to test whether gradients help more or less on more complex tasks. We experiment on both MNIST and CIFAR10. Finally, we manipulate the *complexity of the model class* to test whether gradients help more when the model is simpler. We train three models on each of the two tasks that are chosen to display a range of complexity.

Dependent measure. We measure the accuracy of our reconstructed classifier $\hat{f}$ on a test set of 10,000 images from the task (MNIST or CIFAR10).

Experimental procedure. We split our datasets into three parts:

- A training set of images and ground-truth labels for the true classifier f. The training set for MNIST has 50,000 examples and for CIFAR10 has 40,000 examples.
- A training set of 10,000 images for the reconstructed classifier $\hat{f}$. Note that $\hat{f}$ does not have access to ground-truth labels, so it must query f for labels.
- A test set of 10,000 images and ground-truth labels for f and $\hat{f}$.

We first train models to serve as the true classifier f. We train three types of models on MNIST: a 1-layer network (multinomial logistic regression), a 2-layer neural network with ReLu activations, and a network with two convolutional layers (each followed by a max-pool layer) followed by two dense layers. We also train three types of models on CIFAR10: the same convolutional network used for MNIST (with the input dimension changed appropriately), a VGG11 network [13], and a ResNet-18 network [9].

Next, we train a new classifier $\hat{f}$ from the same model class as the true classifier f. The inputs x given to $\hat{f}$ are randomly sampled from the training set for $\hat{f}$. After training, we compute the accuracy of our reconstructed classifier $\hat{f}$ on the test set.

Follow-up experiments: unknown model class and data distribution An adversary trying to reconstruct the classifier f may not know the model class of f or the data distribution. So, in follow-up experiments we (1) reconstruct the classifier f with a classifier $\hat{f}$ from a different model class and (2) reconstruct the classifier f using Gaussian generated queries. In these follow-up experiments we analyze the same factors, but with a subset of conditions.

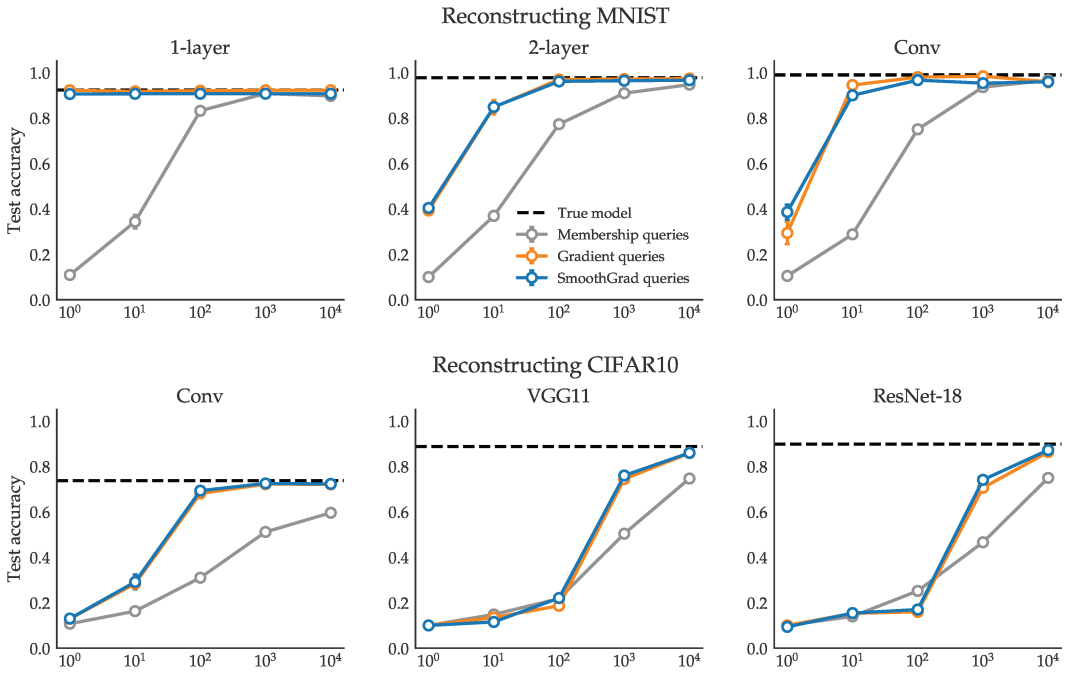

Figure 2: Access to gradients improves the accuracy of the recovered model. The improvement is approximately the same even with gradients processed by SmoothGrad.

5 Experimental results and discussion

5.1 Main experiments: gradient queries versus membership queries

Figure 2 shows the results of our main experiments, described in Section 4.

Type of query. Across all experiments, training with gradient queries leads to orders of magnitude fewer queries required to learn the model. For example, for the MNIST convolutional model we get to 95% accuracy in 10 gradient queries, compared to 1000 membership queries. We find practically no difference between gradient queries and SmoothGrad queries, despite picking the hyperparameters for SmoothGrad that produced the best saliency maps (See Appendix D).

Complexity of model class. We find that the gap in performance between gradient queries and membership queries is larger for models of lower complexity.

As an extreme case, consider the 1-layer network on MNIST. We find a 1000x decrease in the number of queries required. With gradient queries it takes only one query to reconstruct the model (get the same performance as the original classifier). This makes sense because with gradient queries the 1-layer network is identifiable in one query, compared to 784 membership queries.[2]

On MNIST with the 2-layer or convolutional network we find a 100x decrease in the number of queries needed to reconstruct the model. On CIFAR10 we find that the convolutional network (which

[2]The 1-layer network is $f(x) = \sigma(w^\top x)$ where σ is the sigmoid function and $w \in \mathbb{R}^{784}$. The model parameters w are equal to $\frac{1}{f(x)(1-f(x))}\nabla f(x)$, and thus, identifiable in one gradient and membership query.

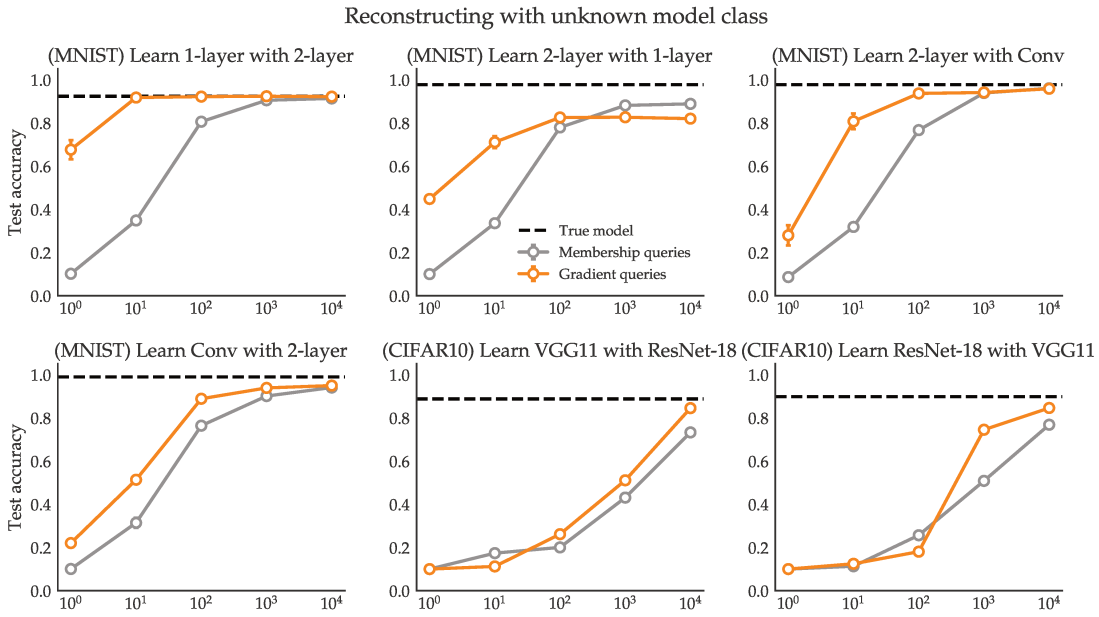

Figure 3: Gradients still help when the model is unknown, but they help more when the reconstructed classifier is from a model class that is more complex than the model class of the true classifier.

is the same as the convolutional network used for MNIST) also has at least a 100x decrease in the number of queries needed. On the other hand, VGG11 and Resnet-18 show only a 10x decrease in the number of queries needed to reach 75% accuracy.

Complexity of task. We find that the relative reduction in queries needed seems to depend on the complexity of the model class, rather than the complexity of the task. But, not surprisingly, the absolute number of queries needed increases with the complexity of the task.

On both MNIST and CIFAR10 gradient queries lead to a 100x decrease for reconstructing the convolutional network, suggesting that for the relative decrease in query complexity depends more on the complexity of the model class than the complexity of the task. However, as might be expected, for both gradient and membership queries the absolute number of queries needed increases as the complexity of the task increases. On MNIST the convolutional model is reconstructed in 10 gradient queries, compared to 1000 membership queries. On CIFAR10 the convolutional model is reconstructed in 100 gradient queries, compared to 10,000 membership queries.

5.2 Unknown model class

In the scenario where we do not know the true model class beforehand, we experiment with:

- MNIST: Reconstructing the 1-layer model with the 2-layer network (and vice versa).
- MNIST: Reconstructing the 2-layer model with the convolutional network (and vice versa).
- CIFAR10: Reconstructing the VGG11 model with the ResNet-18 network (and vice versa).

We refer the reader to Section 4 for details on the models. Figure 3 displays our results.

We find that gradient queries seem to help more when the the model class of $\hat{f}$ is more complex than the

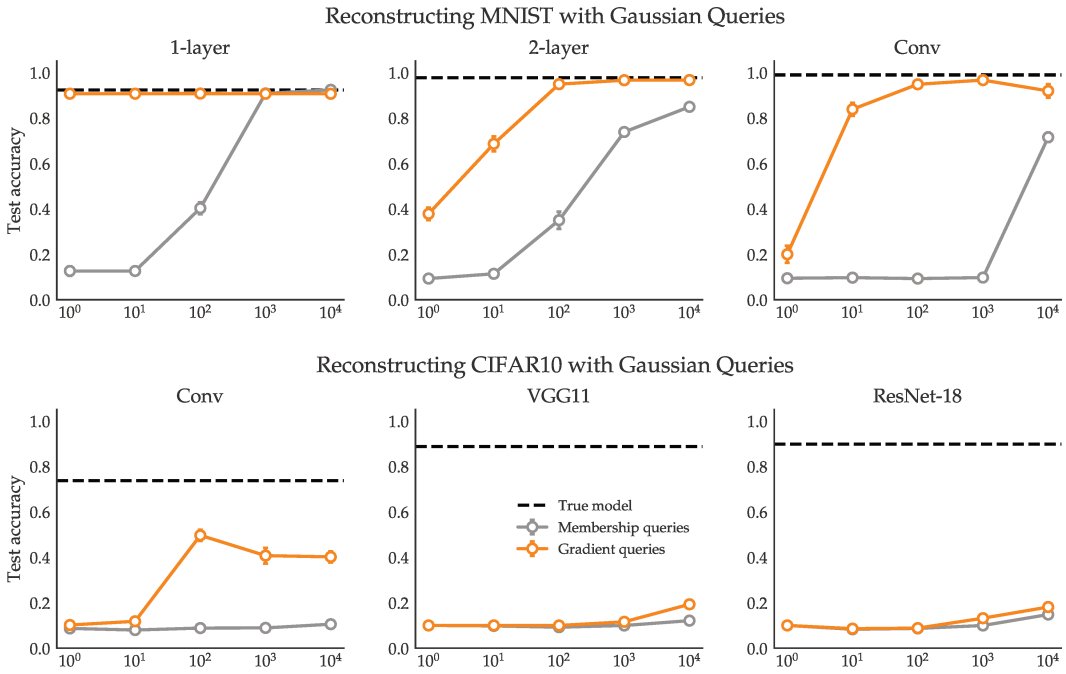

Figure 4: When querying with Gaussian generated inputs, we seem to see a *larger* gap between the performance of gradient queries and the performance of membership queries.

true classifier f. For example, we see a 100x decrease in the number of queries needed to reconstruct MNIST 1-layer with a 2-layer network. But, we only get an initial 10x decrease in the number of queries needed to reconstruct MNIST 2-layer with a 1-layer network. Similarly reconstructing the 2-layer network with the convolutional network works much better than reconstructing the convolutional network with the 2-layer network.

We have been fairly loose when referring to the relative complexities of different models, and it is unclear to us how to compare VGG11 and ResNet-18 in terms of complexity. Interestingly however, we find that although gradient queries still lead to a 10x decrease when reconstructing ResNet-18 with VGG11, they help very little when reconstructing a VGG11 model with a ResNet-18 network.

5.3 Unknown data distribution

We now analyze the setting where we do not know the data distribution. Instead we query using randomly generated Gaussian queries, i.e $x \sim \mathcal{N}(0, I_d)$. Figure 4 displays our results.

On MNIST we find that Gaussian queries lead to a *greater* gap in performance between gradient and membership queries, compared to when using images from the data distribution.[3] On the MNIST 2-layer network, we see at least a 1000x decrease, compared to the 100x decrease we saw in Section 4 when using queries from the data distribution. On the MNIST convolutional network, we see that in 10 gradient queries we get to 84% accuracy. On the other hand, it takes 10,000 membership queries

[3]On the 1-layer network we see the same relative decrease because it is identifiable with a single gradient + membership query or 784 membership queries, independent of the distribution the queries are generated from.

to learn at all, and even then we get to only 71%. Thus, we seem to get at least a 1000x decrease, compared to the 100x reduction we saw when using queries from the data distribution.

On CIFAR10 it is harder to interpret the results because the performance degrades so much for both gradient and membership queries. However, at least in the convolutional network, the gap between gradient and membership queries also seems to increase. The reconstructed model gets to 50% accuracy in 10 gradient queries, but only to 11% accuracy in 10,000 membership queries.

6 Related work

Tramèr et al. show how models can be reconstructed in practice through *prediction APIs* [19]. Our work addresses the complementary threat of model leakage through a hypothetical *explanation API*. While differential privacy can help guard against attacks from prediction APIs [7], it is not clear if this is a viable approach for preventing reconstruction from explanations.

Learning a model via a prediction API instantiates the framework of *learning with membership queries*, in which the learner gets to actively query an oracle for labels to inputs of its choosing [1]. In our work, we propose a complementary learning framework: *learning from input gradient queries*. Similar to membership queries and prediction APIs, we believe that learning from gradients is likely to be the theoretical framework underpinning reconstruction from explanation APIs.

We give a near-optimal algorithm for learning a two-layer network with ReLU activations through gradient queries. The geometric intuition for our algorithm is similar to the work of Baum for learning two-layer linear threshold networks with membership queries [3].

References

[1] Dana Angluin. Queries and concept learning. *Machine learning*, 1988.

[2] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *Journal of Machine Learning Research (JMLR)*, 2010.

[3] Eric B Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks*, 1991.

[4] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[5] Nicholas Carlini, Chang Liu, Jernej Kos, Úlfar Erlingsson, and Dawn Song. The secret sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv preprint arXiv:1802.08232*, 2018.

[6] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.

[7] Cynthia Dwork and Vitaly Feldman. Privacy-preserving prediction. *Conference on Learning Theory (COLT)*, 2018.

[8] Moritz Hardt, Nimrod Megiddo, Christos Papadimitriou, and Mary Wootters. Strategic classification. In *ACM Conference on Innovations in Theoretical Computer Science (TCS)*, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2017.

[11] Itay Safran and Ohad Shamir. Spurious local minima are common in two-layer relu neural networks. *International Conference on Machine Learning (ICML)*, 2018.

[12] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy (SP)*, 2017.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[15] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[16] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

[18] Yuandong Tian. An analytical formula of population gradient for two-layered relu network and its applications in convergence and critical point analysis. *International Conference on Machine Learning (ICML)*, 2017.

[19] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *USENIX Security Symposium*, 2016.

[20] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2014.

[21] Kai Zhong, Zhao Song, Prateek Jain, Peter L Bartlett, and Inderjit S Dhillon. Recovery guarantees for one-hidden-layer neural networks. *International Conference on Machine Learning (ICML)*, 2017.

A Omitted proofs for Algorithm 1a

First, we prove the following two lemmas that will be useful in proving the anti-concentration and concentration bounds in Lemma 2 and Lemma 3.

Lemma 5. *(Anti-concentration of difference of χ^2_2 variables)*
Let $Q, R \sim \chi^2_2$. Then, $P(|Q - R| \leq \epsilon) \leq \epsilon$ for $\epsilon > 0$.

Proof. Recall that the cumulative distribution function of a χ^2_d random variable Q is

$$P(Q \leq x) = \frac{\gamma(\frac{d}{2}, \frac{x}{2})}{\Gamma(\frac{d}{2})}$$

where $\gamma(s, z) = \int_0^z t^{s-1} e^{-t} dt$ is the lower incomplete gamma function and $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$ is the gamma function. When $d = 2$, $P(Q \leq x)$ simplifies to $\int_0^{x/2} e^{-z} dz$. Thus,

$$\begin{aligned} P(|Q - R| \leq \epsilon) &= P(R - \epsilon \leq Q \leq R + \epsilon) \\ &\leq P(0 \leq Q \leq 2\epsilon) \\ &= \int_0^\epsilon e^{-x} dx \\ &\leq \int_0^\epsilon dx \\ &= \epsilon \end{aligned}$$

□

Lemma 6. *(Distribution of product of independent Gaussians)*
Let $X, Y \sim \mathcal{N}(0, 1)$. Then XY can be written as

$$XY = \frac{1}{2}(Q - R)$$

where $Q, R \sim \chi^2_1$ are independent.

Proof. XY can be rewritten as

$$XY = \frac{1}{4}((X + Y)^2 - (X - Y)^2).$$

Since $\mathrm{Cov}(X + Y, X - Y) = 0$, we know $X + Y$ and $X - Y$ are independent random variables from a $\mathcal{N}(0, 2)$ distribution. Thus, we can express $(X + Y)^2$ and $(X - Y)^2$ as $(X + Y)^2 = 2Q$ and $(X - Y)^2 = 2R$ where Q, R are independent χ^2_1 random variables. Thus, $XY = \frac{1}{2}(Q - R)$. □

Lemma 2. *Let $a, b \in \mathcal{S}^{d-1}$ be unit vectors such that $|\langle a, b\rangle| \leq 1 - c$ for some scalar $c \in [0, 1]$. Suppose we pick random vectors $u, v \sim \mathcal{N}(0, I_d)$. Let $t_1, t_2 \in \mathbb{R}$ be scalars such that $\langle a, u + t_1 v\rangle = 0$ and $\langle b, u + t_2 v\rangle = 0$.[4] Then,*

$$P(|t_1 - t_2| \leq \epsilon) \leq 3^{\frac{4}{3}} \left(\frac{\epsilon}{c}\right)^{\frac{2}{3}}.$$

[4]With probability one such a t exists.

Proof. Solving for the scalars t_1 and t_2 yields

$$t_1 = -\frac{\langle a, u\rangle}{\langle a, v\rangle},\ t_2 = -\frac{\langle b, u\rangle}{\langle b, v\rangle}.$$

Let $a^\perp$ be a unit vector orthogonal to a. The vector b can be expressed as $b = \beta_1 a + \beta_2 a^\perp$ where $\beta_1, \beta_2 \in \mathbb{R}$ and $\beta_1 = \langle a, b\rangle$. The coefficient β_2 can be lower bounded as the following.

$$\beta_2 = \sqrt{1-\beta_1^2} \geq \sqrt{2c - c^2} \geq c$$

Using this expression for the vector b we can rewrite $|t_1 - t_2|$ as

$$\begin{aligned} |t_1 - t_2| &= \left|\frac{\langle a, u\rangle}{\langle a, v\rangle} - \frac{\langle b, u\rangle}{\langle b, v\rangle}\right| \\ &= \frac{|\beta_2(\langle a, u\rangle\langle a^\perp, v\rangle - \langle a, v\rangle\langle a^\perp, u\rangle)|}{|\langle a, v\rangle(\beta_1\langle a, v\rangle + \beta_2\langle a^\perp, v\rangle)|} \\ &= \frac{|\beta_2(X_1Y_2 - X_2Y_1)|}{|X_2(\beta_1 X_2 + \beta_2 Y_2)|}, \end{aligned} \tag{4}$$

where $X_1 = \langle a, u\rangle, X_2 = \langle a, v\rangle, Y_1 = \langle a^\perp, u\rangle, Y_2 = \langle a^\perp, v\rangle$ are independent $\mathcal{N}(0,1)$ random variables. To bound $P(|t_1 - t_2| \leq \epsilon)$ we can bound the numerator and denominator of (4) separately. For all $k > 0$, the following inequality holds.

$$P(|t_1 - t_2| \geq \epsilon) \geq P\left(|\beta_2(X_1Y_2 - X_2Y_1)| \geq k\sqrt{\epsilon},\ |X_2(\beta_1 X_2 + \beta_2 Y_2)| \leq \frac{k}{\sqrt{\epsilon}}\right)$$

Applying a union bound to the complementary event yields,

$$P(|t_1 - t_2| \leq \epsilon) \leq P(|\beta_2(X_1Y_2 - X_2Y_1)| \leq k\sqrt{\epsilon}) + P\left(|X_2(\beta_1 X_2 + \beta_2 Y_2)| \geq \frac{k}{\sqrt{\epsilon}}\right). \tag{5}$$

Applying Lemma 6 to the independent products X_1Y_2 and X_2Y_1 simplifies the numerator to

$$|\beta_2(X_1Y_2 - X_2Y_1)| = \left|\frac{\beta_2}{2}(Q - R)\right|,$$

where $Q, R \sim \chi^2_2$ are independent Chi-squared random variables. Then by Lemma 5,

$$P(|\beta_2(X_1Y_2 - X_2Y_1)| \leq k\sqrt{\epsilon}) \leq P\left(|Q - R| \leq \frac{2k\sqrt{\epsilon}}{\beta_2}\right) \leq \frac{2k\sqrt{\epsilon}}{\beta_2}.$$

To upper bound the tail probability of the denominator (the second term in Equation 5) note that

$$\mathbb{E}[(X_2(\beta_1 X_2 + \beta_2 Y_2))^2] = 3\beta_1^2 + \beta_2^2.$$

Then by Markov's Inequality,

$$P\left(|X_2(\beta_1 X_2 + \beta_2 Y_2)| \geq \frac{k}{\sqrt{\epsilon}}\right) = P\left((X_2(\beta_1 X_2 + \beta_2 Y_2))^2 \geq \frac{k^2}{\epsilon}\right) \leq \frac{(3\beta_1^2 + \beta_2^2)\epsilon}{k^2} = \frac{3\epsilon}{k^2}.$$

Therefore,

$$P(|t_1 - t_2| \leq \epsilon) \leq \frac{2k\sqrt{\epsilon}}{\beta_2} + \frac{3\epsilon}{k^2}.$$

Minimizing the right-hand side with respect to k yields

$$P(|t_1 - t_2| \leq \epsilon) \leq 3^{\frac{4}{3}} \left(\frac{\epsilon}{\beta_2}\right)^{\frac{2}{3}} \leq 3^{\frac{4}{3}} \left(\frac{\epsilon}{c}\right)^{\frac{2}{3}} \leq O\left(\left(\frac{\epsilon}{c}\right)^{\frac{2}{3}}\right).$$

□

Lemma 3. *Let $a \in \mathcal{S}^{d-1}$ be a unit vector. Suppose we pick random vectors $u, v \sim \mathcal{N}(0,1)$. Let $t \in \mathbb{R}$ be the value such that $\langle a, u + tv\rangle = 0$. Then,*

$$P(|t| \geq l) \leq \frac{2}{\pi l}.$$

Proof. $t = -\frac{\langle a,u\rangle}{\langle a,v\rangle}$ follows a standard Cauchy distribution. The cumulative distribution function of a standard Cauchy random variable X is $P(X \leq a) = \frac{1}{\pi}\arctan(a) + \frac{1}{2}$. Thus,

$$\begin{aligned}
P(|t| \geq l) &= P(t \leq -l) + P(t \geq l) \\
&= \left(\frac{1}{\pi}\arctan(-l) + \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{1}{\pi}\arctan(l)\right) \\
&= 1 - \frac{2}{\pi}\arctan(l) \\
&\leq 1 - \frac{2}{\pi}\left(\frac{\pi}{2} - \frac{1}{l}\right) \\
&= \frac{2}{\pi l}
\end{aligned}$$

□

B Picking query points in Algorithm 1b

For completeness, we show that we can easily find a matrix X which satisfy the requirements of Lemma 4 through the following steps:

1. Pick a random $v \in \mathbb{R}^d$. Let $g(v) = 1\{Zv \geq 0\}$ and $\mathcal{C} = \{x \mid g(v) = g(x)\}$ be the cell containing v.
2. Find the center $y_0 \in \mathbb{R}^d$ and radius $r \in \mathbb{R}$ of the largest ℓ_2 ball within $\mathcal{C} \cap [0,1]^d$. The center of this ball is known as the *Chebyshev center* of $\mathcal{C} \cap [0,1]^d$. It is well known that the center y_0 and radius r can be solved for through a linear program [4]. We include the linear program for

our case below.

$$\begin{aligned}&\max_{y_0, r} r\\ &\text{subject to}\\ &(z_i^\top y_0 + r)\,\mathrm{sgn}(z_i^\top v) \geq 0\\ &\mathbf{0} \leq y_0 \leq \mathbf{1}\end{aligned}$$

3. Construct a set of d linearly independent vectors $\mathcal{Y} = y_1, \ldots, y_d \in \mathcal{C}$ as follows.

$$\begin{aligned}y_i &= y_0 + \Delta^i\\ \Delta^i_j &= \begin{cases} r/2 & i = j \\ 0 & i \neq j \end{cases}\end{aligned}$$

4. Let $Y \in \mathbb{R}^{d \times d}$ be the matrix whose columns are formed by the vectors in $\mathcal{Y}$. Since Y has rank d, $\mathrm{Rank}(ZY) = h$. Thus, we can pick h vectors $x_1, \ldots, x_h$ from $\mathcal{Y}$ such that $\mathrm{Rank}(ZX) = h$ where X is the matrix whose columns are the vectors $x_1, \ldots, x_h$.

C Reconstruction from membership queries

We now consider how to reconstruct the two-layer ReLU neural network described in Section 2 with membership queries alone, rather than membership and gradient queries. We show that we can convert our algorithm into one that learns with membership queries by estimating the gradients of f with membership queries.

C.1 Membership query version of Algorithm 1

We define the membership query version of Algorithm 1, referred to as Algorithm 1-MQ, by replacing any use of the gradient $\nabla f(x)$ with an estimate of the gradient, $\widehat{\nabla} f(x)$, computed with d membership queries. We estimate the gradient by estimating each component separately through a finite difference approximation:

$$\widehat{\nabla} f(x)_j = \frac{f(x + \Delta^j) - f(x)}{s},$$

where $i, j \in [d]$, $s \in \mathbb{R}^+$ and $\Delta^j_i = \begin{cases} s & \text{if } i = j \\ 0 & \text{o.w.} \end{cases}$.

Our main result shows that we can recover the function f in $O(dh \log \frac{h}{\delta})$ membership queries:

Theorem 4. *With probability $1 - \delta$, if $s \leq \frac{\delta\epsilon}{2(2-\delta)l\epsilon}$, where $l, \epsilon \in \mathbb{R}$ are parameters of the binary search in Algorithm 1a, then Algorithm 1-MQ returns a function $\hat{f}$ such that $\hat{f} = f$ in $O(dh \log \frac{h}{\delta})$ membership queries.*

The next subsection contains our proofs.

C.2 Proofs

The proof of Theorem 4 relies on showing that we can pick an s small enough so that with high probability all estimates of the gradient are equal to the exact gradient. We show this by proving that if all points used in estimating a gradient lie in the same cell defined by the separating hyperplanes of f, then the estimate of the gradient $\widehat{\nabla} f(x)$ is equal to the gradient $\nabla f(x)$. If s is small enough, then all points evaluated for a gradient estimate will lie in the same cell, and thus the exact gradient will be recovered. By choosing s small enough, we can ensure that all gradients estimated by Algorithm 1-MQ are equal to the exact gradient with high probability.

First, we show that if all points sampled in estimating the gradient lie in the same cell, then the estimate of the gradient $\widehat{\nabla} f(x)$ is equal to the gradient $\nabla f(x)$:

Lemma 7. *Suppose for all $j \in [d]$, $x + \Delta^j$ lies in the same cell as x, i.e,*

$$\mathbb{I}\{Ax \geq 0\} = \mathbb{I}\{A(x + \Delta^j) \geq 0\}\,.$$

Then, $\widehat{\nabla} f(x) = \nabla f(x)$.

Proof. Recall that the function f can be expressed as

$$f(x) = w^\top \mathrm{Diag}(\mathbb{I}\{Ax \geq 0\})Ax\,.$$

Thus, the j-th component of the gradient of f is

$$\nabla f(x)_j = w^\top \mathrm{Diag}(\mathbb{I}\{Ax \geq 0\})a_j\,,$$

where a_j is the j-th column of A. Our estimate of the gradient is

$$\begin{aligned}
\widehat{\nabla} f(x)_j &= \frac{f(x + \Delta^j) - f(x)}{s} \\
&= \frac{w^\top \mathrm{Diag}(\mathbb{I}\{A(x + \Delta^j) \geq 0\})A(x + \Delta^j) - w^\top \mathrm{Diag}(\mathbb{I}\{Ax \geq 0\})Ax}{s} \\
&= \frac{w^\top \mathrm{Diag}(\mathbb{I}\{Ax \geq 0\})A(x + \Delta^j) - w^\top \mathrm{Diag}(\mathbb{I}\{Ax \geq 0\})Ax}{s} \\
&= \frac{w^\top \mathrm{Diag}(\mathbb{I}\{Ax \geq 0\})A\Delta^j}{s} \\
&= \frac{w^\top \mathrm{Diag}(\mathbb{I}\{Ax \geq 0\})a_j s}{s} \\
&= w^\top \mathrm{Diag}(\mathbb{I}\{Ax \geq 0\})a_j \\
&= \nabla f(x)_j\,.
\end{aligned}$$

Therefore, $\widehat{\nabla} f(x) = \nabla f(x)$. □

The next lemma shows that if s is small enough, then all points evaluated used to estimate a gradient lie in the same cell, and thus the exact gradient is recovered.

Lemma 8. *Suppose $s \in \mathbb{R}^+$ is such that $|Ax| \geq s\mathbf{1}$, then $\widehat{\nabla} f(x) = \nabla f(x)$.*

Proof. We simply need to prove that $\mathbb{I}\{A(x+\Delta^j) \geq 0\} = \mathbb{I}\{Ax \geq 0\}$ for all $j \in [d]$ and then the result follows by Lemma 7. Since the rows of the weight matrix A are unit norm, we know $|sa_j| \leq |s\mathbf{1}| \leq |Ax|$ where a_j is the j-th column of A. Thus, $\mathbb{I}\{A(x+\Delta^j) \geq 0\} = \mathbb{I}\{Ax + sa_j \geq 0\} = \mathbb{I}\{Ax \geq 0\}$. The result then follows from Lemma 7. □

Next, given a particular value of s, we bound the probability that all gradients we estimate with our algorithm are exactly equal to the true gradient.

Lemma 9. *Let $\mathcal{X} = \{u + i\epsilon v \mid |i| \leq l/\epsilon, i \in \mathbb{Z}\}$ be the set of points Algorithm 1a may query. Then,*

$$P(\forall x \in \mathcal{X} \;\; \widehat{\nabla} f(x) = \nabla f(x)) \geq 1 - \frac{2lhs}{\epsilon}\,.$$

Proof. First we will establish a bound for one row a of the weight matrix A.

$$\begin{aligned}
&P(\exists x \in \mathcal{X} : |\langle a, x\rangle| \leq s) \\
&\leq \sum_{x \in \mathcal{X}} P(|\langle a, x\rangle| \leq s) && \text{Union bound} \\
&= \sum_{|i| \leq l/\epsilon, i \in \mathbb{Z}} P(|\langle a, u + i\epsilon v\rangle| \leq s) \\
&= \sum_{|i| \leq l/\epsilon, i \in \mathbb{Z}} P(|Z_i| \leq s) && Z_i \sim \mathcal{N}(0, 1 + (\epsilon i)^2) \\
&\leq \frac{2l}{\epsilon} P(|Z| \leq s) && Z \sim \mathcal{N}(0,1) \\
&\leq \frac{2ls}{\epsilon} && \text{Gaussian anti-concentration}
\end{aligned}$$

A union bound on all rows of the weight matrix A then shows that

$$P(\exists x \in \mathcal{X}, i \in [h] : |\langle A_i, x\rangle| \leq s) \leq \sum_{i=1}^{h} P(\exists x \in \mathcal{X} : |\langle A_i, x\rangle| \leq s) \leq \frac{2lhs}{\epsilon}\,.$$

Thus, by Lemma 8,

$$P(\forall x \in \mathcal{X} \;\; \widehat{\nabla} f(x) = \nabla f(x)) = P(\forall x \in \mathcal{X} \;\; |Ax| \geq s\mathbf{1}) \geq 1 - \frac{2ls}{\epsilon}\,.$$

□

Finally, we show that by picking s small enough so that all gradients estimate are exact with high probability, the sample complexity of Algorithm 1-MQ becomes $O(dh \log \frac{h}{\delta})$ membership queries.

Theorem 4. *With probability $1 - \delta$, if $s \leq \frac{\delta\epsilon}{2(2-\delta)l\epsilon}$, where $l, \epsilon \in \mathbb{R}$ are parameters of the binary search in Algorithm 1a, then Algorithm 1-MQ returns a function $\hat{f}$ such that $\hat{f} = f$ in $O(dh \log \frac{h}{\delta})$ membership queries.*

Proof. Algorithm 1 only uses gradients of f in Algorithm 1a and Algorithm 1 succeeds if and only if Algorithm 1a succeeds. Thus, we can bound the success of Algorithm 1-MQ by bounding the probability that all gradients used in Algorithm 1a are estimated exactly.

In $O(h \log \frac{2h}{\delta}) = O(h \log \frac{h}{\delta})$ gradient queries we can guarantee that Algorithm 1a succeeds with probability $1 - \frac{\delta}{2}$. The probability Algorithm 1-MQ succeeds then becomes the following.

$$
\begin{aligned}
&P(\text{Algorithm 1-MQ succeeds}) \\
&= P(\text{Algorithm 1a succeeds}|\text{Exact gradients})P(\text{Exact gradients}) \\
&\geq \left(1 - \frac{\delta}{2}\right)\left(1 - \frac{2lh}{\epsilon}s\right) && \text{(Lemma 9)} \\
&= \left(1 - \frac{\delta}{2}\right)\left(1 - \frac{2lh}{\epsilon}\left(\frac{\delta\epsilon}{2(2-\delta)lh}\right)\right) \\
&\geq 1 - \delta
\end{aligned}
$$

Since, it takes d membership queries to compute each gradient that Algorithm 1a requires, the sample complexity becomes $O(dh \log \frac{h}{\delta})$ membership queries.

□

D SmoothGrad

Instead of returning the raw gradient $\nabla f(x)$, SmoothGrad [15] returns an average of gradients around the input x:

$$\widetilde{\nabla} f(x) = \sum_{i=1}^{N} \frac{1}{N} \nabla f(x + z_i),$$

where $z_i \sim \mathcal{N}(0, \sigma^2 I)$ and $N > 0$. SmoothGrad has two hyperparameters: (1) σ the standard deviation of the Gaussian noise and (2) N the number of samples to pick.

As shown in Figure 5, we found that the best value of σ for MNIST was 1000 times σ_D, the standard deviation of the images in the dataset. On CIFAR10 using either the VGG-11 or ResNet-18 network, no value of σ seems to produce a sharp map (Figures 6 and 7). So for our CIFAR10 experiments, we set σ equal to the standard deviation of the dataset σ_D. In the original SmoothGrad paper, Smilkov et al. find that the best value of σ for MNIST is about 70% the spread of the dataset, while on ImageNet it is only 10-20%. So the difference between the value of σ we use on MNIST and the value of σ we use on CIFAR10 seems to qualitatively match the difference in the value of σ Smilkov et al. use on MNIST and ImageNet.

We expect that SmoothGrad may eventually degrade the performance of the reconstructed model as σ increases. But at least for the values of σ we test, which are already quite large relative to the standard deviation of the dataset, and seem to match values that may be used in practice, we see no degradation in performance when using gradients preprocessed by SmoothGrad.

Regarding the number of samples, N, Smilkov et al. state that the estimated gradient becomes smoother as N increases, but that they find diminishing returns for $N > 50$. For computational reasons we set $N = 10$ in our experiments, however, this should only make it *harder* to learn, since the outputs of SmoothGrad become noisier.

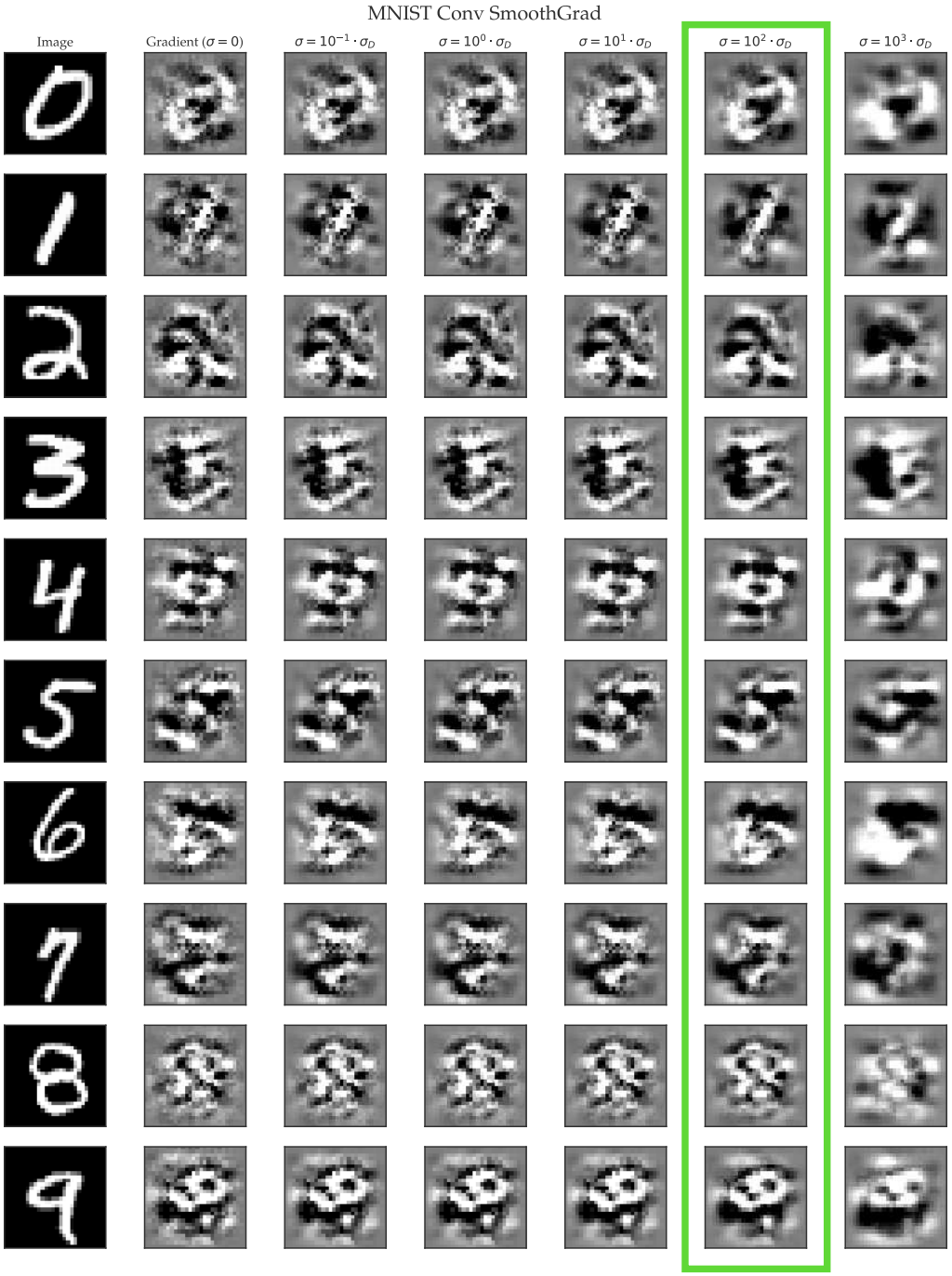

Figure 5: Saliency maps computed with SmoothGrad on the MNIST convolutional network described in Section 4 using $N = 100$. For our experiments in Section 4, we choose σ to be 1000 times the standard deviation σ_D of the dataset (highlighted column).

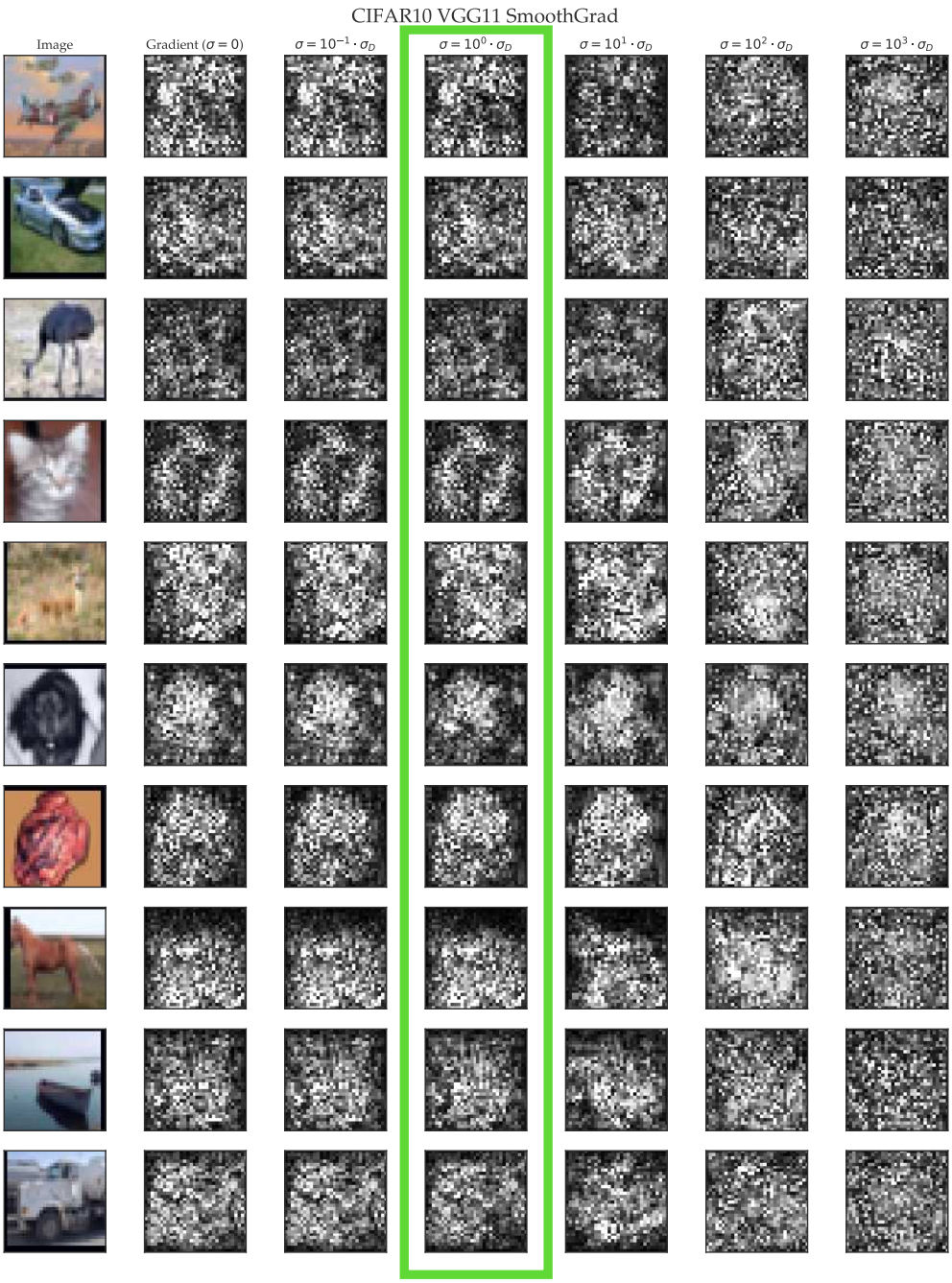

Figure 6: Saliency maps computed with SmoothGrad on the CIFAR10 VGG-11 network using $N = 100$. Following [15], for CIFAR10, which has RGB images, we visualize the absolute value of the output of SmoothGrad. For our experiments in Section 4, we choose σ to be equal to the standard deviation σ_D of the dataset (highlighted column).

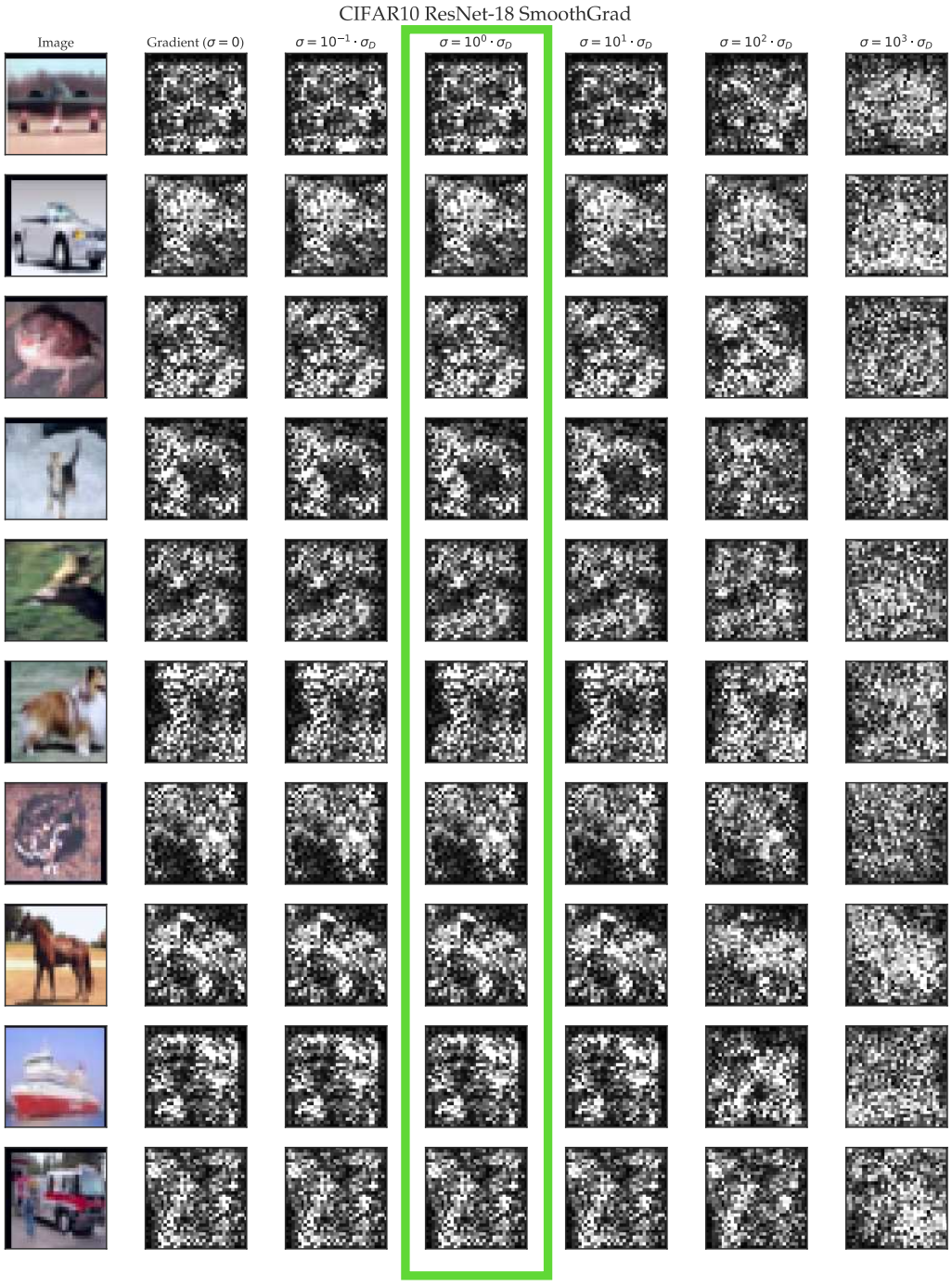

Figure 7: Saliency maps computed with SmoothGrad on the CIFAR10 ResNet-18 network using $N = 100$. Following [15], for CIFAR10, which has RGB images, we visualize the absolute value of the output of SmoothGrad. For our experiments in Section 4, we choose σ to be equal to the standard deviation σ_D of the dataset (highlighted column).