Lecture 19: Privacy - Overview and Model Stealing

Lecturer: Vinod Vaikuntanathan **Date:** April 16, 2026
Scribes: Duru Ozer, Richard Yun

1. The Privacy & Attack Landscape

The space of possible privacy attacks on models can roughly be divided into two categories:

1. **Privacy Attacks on Training:**
   - **Model Inversion & Reconstruction Attacks:** Given white-box access to model's weights, the attacker is trying to reconstruct the training data.

   One possible defense against these type of attacks is the following definition:

   Definition 1.1. (Differential Privacy) A dataset has differential privacy, if for any chosen data point, an adversary cannot distinguish a model trained on the entire dataset versus a model trained on the dataset with that particular data point excluded. In other words, no single data point is too significant for the output.

   Another related mitigation problem is:

   - **Unlearning:** After training the model on a dataset, we might want to "unlearn" some data point(s) in the original dataset, so that an attacker can no longer extract any information about these data points at all.

     An example of this could be the EU's 'right to be forgotten'. What if anyone can ask that all of the information related to their past is removed from a model we trained?

     This is currently an open problem, and it lacks formal definitions.

2. **Main Focus of Today: Privacy Attacks on Inference:**

   Given oracle access to a model, the attacker is trying to reconstruct the model.

   Note this is inevitable given infinite queries since the underlying model is learnable. However, we are also interested in efficiency, i.e. does reconstructing a model use significantly less data points compared to training a model from scratch?

Intuition

Oracle access is possibly stronger than training on a given dataset, because we get to choose and adapt what distribution we will train on. This is in contrast with training from scratch, where we would be restricted to sampling from the original distribution of the dataset.
The oracle access might also give us the probability distribution for each output token (if it is a generative model) or label (if it is a classification model). This would give us more information than only having the input and the correct output, which was the only information available when training from scratch.

2. Model Stealing

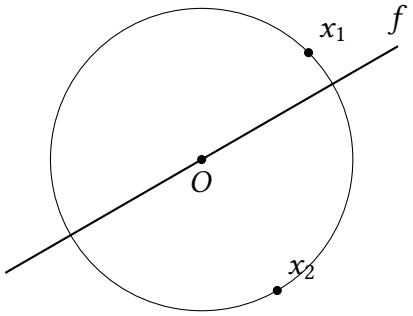
Definition 2.1 (Model Stealing)**.** Let $f : \mathcal{X} \to \mathcal{Y}$ be a target model, such that $L_{\mathcal{D}}(f) \leq \varepsilon$, where L is the loss function. An adversary is given black-box oracle access to f, meaning they can query $x \in \mathcal{X}$ and observe outputs $f(x)$.
The goal of model stealing is to reconstruct a surrogate model $\hat{f}$ such that

$$\hat{f} \approx f \quad \text{over } \mathcal{X},$$

where

1. (Success) $L_{\mathcal{D}}(\hat{f}) \leq O(\varepsilon)$
2. (Efficiency) the process uses much fewer queries than training the model from scratch, without adaptive queries.

Example 2.1. Linear Seperator:

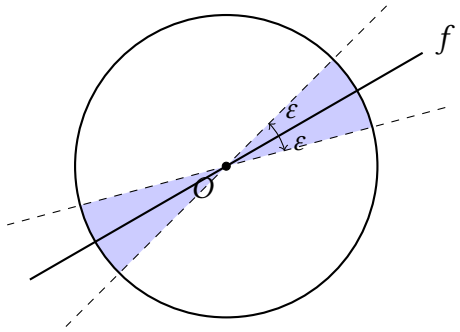


Definition 2.2. (Linear Seperator) On the example above, a linear seperator f can be defined by a line going through the origin. It aims to classify points on the unit circle based on their position relative to f, such that $f(x_1) = 1, f(x_2) = 0$

Training a Linear Separator from Scratch

Claim 2.2. *Let $\mathcal{D}$ be uniform on unit circle and let $\mathcal{H}$ be the set of all lines passing through O. Then, the sample complexity to learn a linear separator that is ε-close to f is $= \Omega\left(\frac{1}{\varepsilon}\right)$*

Proof.



In order to be learn a seperator that is ε-close to f, we need at least two data points that fall within ε distance of f on both sides.

If our samples are from the distribution $\mathcal{D}$, the probability of sampling such a point is $\Theta(\varepsilon)$. By expectation, we will need at least $\Theta\left(\frac{1}{\varepsilon}\right)$ samples to get such points with high probability. □

Stealing a Linear Seperator

Claim 2.3. *Given oracle access to a model, is is possible to learn a linear separator that is ε close to f with query complexity $O\left(\log \frac{1}{\varepsilon}\right)$*

Proof. We can do binary search with precision ε to find the linear seperator:

Algorithm 1: Stealing a Linear Seperator Using Active Learning

$L \leftarrow 0, R \leftarrow \pi$;
while $R - L > \varepsilon$ **do**
  $M \leftarrow \frac{L+R}{2}$;
  if $f(M) \neq f(L)$ **then**
    $R \leftarrow M$;
  else
    $L \leftarrow M$;
  end
end
return *some line f' in the range (L, R) that passes through O*;

Remark

The binary search in this example illustrates how active learning given oracle access to the original model can make stealing more efficient. This intuition is critical for many other model stealing algorithms as well.

Example 2.4. Learning (Noisy) Linear Functions:

Let M be a noisy linear model such that

$$M(a) = \langle a, s \rangle + e$$

where $a, s \in \mathbb{F}_2^n$ and $e \sim Bin(p)$

In other words,

$$M(a) = \begin{cases} \langle a, s \rangle & w.p. 1 - p \\ \overline{\langle a, s \rangle} & w.p. p \end{cases}$$

Training a Noisy Linear Function from Scratch

Let $\mathcal{D}$ be uniform over $\mathbb{F}_2^n$ and let $\mathcal{H}$ be the set of all possible vectors s, i.e $\mathbb{F}_2^n$.

Theorem 2.5. *(Naive Algorithm) A noisy linear function s can be learned w.h.p with $O(n)$ samples in $O(2^n)$ time.*

Proof. Enumerate all 2^n possibilities for s. For each possibility, if it is not equal to s, it will be wrong in roughly half of the samples, while s will be wrong in only a small portion of the samples. □

Aside: Rigorous Analysis of 2.5 (Not Covered in Lecture)

For each s' s.t. $s' \neq s$

$$\Pr_{\substack{a \sim \mathcal{D} \\ e \sim Bin(p)}} [\langle a, s' \rangle = \langle a, s \rangle + e] \leq \frac{1}{2} + p$$

On the other hand, for $s' = s$

$$\Pr_{\substack{a \sim \mathcal{D} \\ e \sim Bin(p)}} [\langle a, s' \rangle = \langle a, s \rangle + e] \geq 1 - p$$

Then, by Chernoff, given $c \cdot n = \Theta(n)$ samples for some large enough constant c, we will be able to distinguish between $s' = s$ and $s' \neq s$ with probability $1 - e^{\Theta(n)}$. Taking union bound over all 2^n possibilities for s, we can distinguish the correct s from all other possibilities with probability $1 - e^{\Theta(n)}$.

Training a Noisy Linear Function from Scratch

Theorem 2.6. *(Blum-Kalai-Wasserman, 2001 [1]) A noisy linear function s can be learned w.h.p with $O(2^{n/\log n})$ samples in $O(2^{n/\log n})$ time.*

Remark

The result from 2.6 remains the best algorithm in terms of time complexity. By interpolating between 2.5 and 2.6, one can achieve an algorithm with $poly(n)$ samples and $O(2^{n/\log\log n})$ time complexity.

Stealing A Noisy Linear Function

Theorem 2.7. *Given oracle access to a noisy linear function s, it is possible to learn s in polynomial time with query complexity $\tilde{O}(n)$.*

Proof. Consider the following algorithm that learns each entry of s separately:

Algorithm 2: Stealing s Using Active Learning

$s \leftarrow \vec{0}$;
for $i \in 1, 2, ..., n$ **do**
  Sample $a_1, a_2, ..., a_k \in \mathbb{F}_2^n$;
  Let u_i be the vector with 1 at the i-th entry and 0 everywhere else;
  $cnt \leftarrow 0$;
  for $j \in 1, 2, ..., k$ **do**
    Query $x_1 = \langle a_j, s \rangle + e$;
    Query $x_2 = \langle a_j + u_i, s \rangle + e$;
    if $x_1 \neq x_2$ **then**
      $cnt \leftarrow cnt + 1$
    end
  end
  if $cnt \geq T$ **then**
    $s_i \leftarrow 1$;
  else
    $s_i \leftarrow 0$;
  end
end
return s

Analysis:

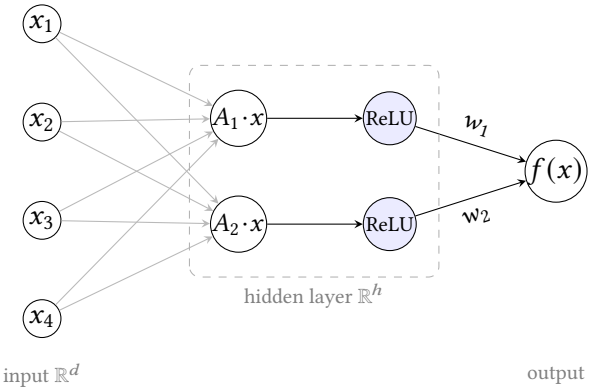
For each a_j, the probability of getting the error term $e = 0$ on both x_1 and x_2 has probability at least $1 - 2p$. (union bound) So, with probability at least $1 - 2p$, $s_i = x_1 - x_2$.

For small p, $1 - 2p > \frac{1}{2}$. Thus, for $k = c \cdot \log n$ for c large enough, we can distinguish between $s_i = 0$ and $s_i = 1$ with probability $1 - \frac{1}{\text{poly}(n)}$. Doing this for each $i \in [n]$, we get all the bits of s correctly with probability $1 - \frac{1}{2}$ using $\tilde{O}(n)$ queries. □

Aside

The generalized form of this method is used in the proof of the Golreich-Levin theorem. [2]

Example 2.8. Learning (depth-2) ReLU Networks



Definition 2.3. (Depth-2 ReLU Network) Recall a depth-2 ReLU network with h hidden units is a function

$$f(\boldsymbol{x}) = \begin{bmatrix} w_1 w_2 \dots w_h \end{bmatrix} \cdot ReLU \left(\begin{bmatrix} A_1 \cdot x \\ A_2 \cdot x \\ \dots \\ A_h \cdot x \end{bmatrix} \right)$$

where $\boldsymbol{x} \in \mathbb{R}$ is the input vector, $\boldsymbol{w} \in \mathbb{R}^h$ is the output weights vector and $\mathbf{A} \in \mathbb{R}^{h \times d}$ is the hidden-layer weight matrix.

Remark

Note that normally our second layer is $Ax + b$ but we omit the bias vector $\mathbf{b}$ for the sake of simplicity.

Also, recall that the Rectified Linear Unit (ReLU) is applied to each entry separately, and it is defined as:

$$ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} = \operatorname{sgn}(x) \cdot x$$

Stealing a Depth-2 ReLU Network

Theorem 2.9. *(Milli-Schmidt-Dragan-Hardt, 2001 [3]) Given oracle access to a depth-2 ReLU network f satisfying*

- *$A_1, \dots, A_h$ are unit vectors (w.l.g),*
- *no two A_i's are collinear: $\langle A_i, A_j \rangle \leq 1 - c$,*
- *$A_1, \dots, A_h$ are linearly independent,*

the algorithm recovers f using

$$O\left(h \log \frac{h}{c}\right)$$

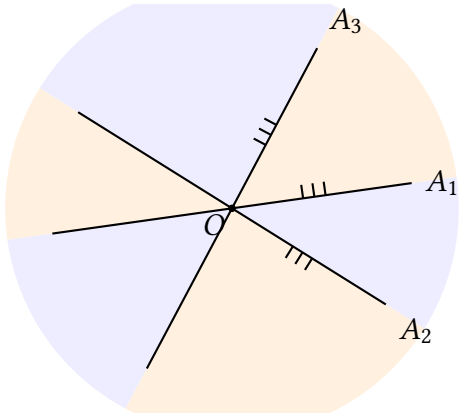
queries.

Remark

(Chen-Klivans-Meka, 2020 [4]) shows that these are not necessary assumptions. In general, you can learn any depth 2 ReLU network in time proportional to the number of neurons in the hidden layer.

Proof. **The Geometry.** Each row $A_i \in \mathbb{R}^d$ defines a hyperplane $\{x : A_i \cdot x = 0\}$ through the origin. The h hyperplanes partition $\mathbb{R}^d$ into regions, and within each region the activation pattern $\left(\mathbf{1}[A_i \cdot x \geq 0]\right)_{i=1}^{h}$ is constant. Since ReLU is linear on each side of its threshold, f is a linear function on each region, with a different gradient in each.

For example, with $d = 2$ and $h = 3$, the three hyperplanes carve $\mathbb{R}^2$ into six sectors:



The hash marks point into the positive half-space $\{x : A_i \cdot x \geq 0\}$ where the i-th ReLU is active. Each of the six shaded sectors corresponds to a distinct activation pattern, and on each sector f is linear with its own gradient. The algorithm exploits this: by walking along a line L and looking for places where the slope of f changes (the kinks), we detect exactly the points where L crosses one of the hyperplanes.

Aside

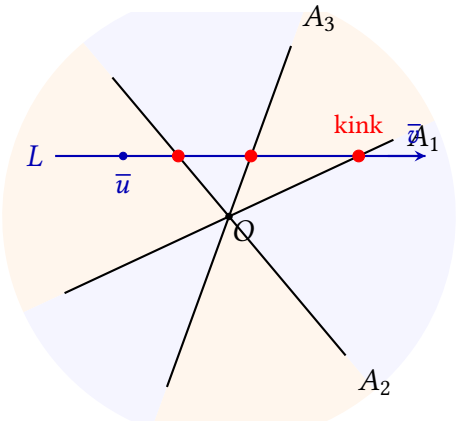
Key Observation. In the region where every $A_i \cdot x$ is positive, the ReLU acts as the identity, so

$$f(x) = \begin{bmatrix} w_1 w_2 \dots w_h \end{bmatrix} \cdot \begin{bmatrix} A_1 \\ A_2 \\ \dots \\ A_h \end{bmatrix} \cdot x = \left(\sum w_i A_i \right) x$$

Taking the gradient $\nabla f(x)$ in this region recovers $\sum_i w_i A_i$. By probing different activation patterns, we can isolate the individual directions.

Goal: recover $Z = [\pm w_1 A_1, \pm w_2 A_2, \dots, \pm w_h A_h]$. Including the signs of all weights, which we cover in step 8

Step (a): Recovering directions via line search



The line $L : \overline{x} = \overline{u} + t\overline{v}$ cuts across the partition, crossing each of the h hyperplanes at most once. Each crossing (red dot) is a point where exactly one ReLU flips its activation, so the gradient of f along L changes there: these are the *kinks* of the one-dimensional restriction $f|_L(t) = f(\overline{u} + t\overline{v})$.

To locate a single kink, binary search exploits the fact that $f|_L$ has constant slope inside each linear piece. This means that a kink in an interval $[t_a, t_b]$ shows up as a slope mismatch between its endpoints. We restrict to $t \in [-\ell, \ell]$ with $\ell = \text{poly}(R)$ large enough that all h kinks fall inside. A coarse sweep then partitions $[-\ell, \ell]$ into subintervals narrow enough that each contains at most one kink, but wide enough that we don't end up querying a dense grid along the whole line — the separation assumption $\langle A_i, A_j \rangle \leq 1 - c$ is what makes such a width possible, since it forces consecutive kinks to be $\Omega(c/h)$ apart. Within each subinterval whose endpoints have mismatched slopes, we bisect: estimate slope(t) from two queries via $\frac{f|_L(t+\eta) - f|_L(t)}{\eta}$, compare the midpoint slope to the left endpoint's, and recurse into whichever half still shows a mismatch. After $O(\log(\ell/\varepsilon)) = O(\log(h/c))$ rounds the interval has shrunk to width $\varepsilon = \text{poly}(c/h)$, and the kink is pinpointed at some $t_i^\star$.

Pinning down $t_i^\star$ tells us *where* the i-th hyperplane meets L, but it doesn't yet tell us which way A_i points. For that, look at how ∇f changes across the kink. The gradient at any point is just $\sum_{j \in S} w_j A_j$ (a sum over whatever units happen to be active there) and crossing the kink flips exactly one bit of S, namely whether unit i is in or out. So every other term cancels in the subtraction:

$$\nabla f(x^\star + \delta\overline{v}) - \nabla f(x^\star - \delta\overline{v}) \;=\; \pm\, w_i A_i.$$

Remark

We note that the exact dependence on the parameter bound ℓ and e was not specified in lecture; we proceed with assumptions of validity.

Algorithm 3: Line search for ReLU boundaries

Sample $\overline{u} \sim \mathcal{N}(0, I_d)$ and a direction $\overline{v}$;
Define the line $L: \ \overline{x} = \overline{u} + t \cdot \overline{v}$;
Restrict to $t \in [-\ell, \ell]$ with $\ell = \text{poly}(R)$;
foreach *kink of* f *along* L **do**
    Binary search with precision $\varepsilon = \text{poly}\left(\frac{c}{h}\right)$;
    Recover the direction $\pm w_i A_i$;
end
return $Z = [\pm w_1 A_1, \ldots, \pm w_h A_h]$;

The total query complexity is $O\left(h \log \frac{h}{c}\right)$.

Step (b): Computing signs

Step (a) returned vectors $z_1, \ldots, z_h$ with $z_i = \pm w_i A_i$ — the line search recovers each direction up to a global sign. Step (b) determines the correct sign for each z_i, equivalently $\text{sgn}(w_i) \in \{\pm 1\}$ (recall $\|A_i\| = 1$, so $|w_i| = \|z_i\|$ is already known).

Recall $\text{sgn}(w) \in \{\pm 1\}$. Given $z = \pm wA$, we determine the sign as follows.

Algorithm 4: Sign recovery

Pick x such that $\langle z, x \rangle > 0$ and $z_j^T x = 0$ for $j \neq i$;
Query the model to obtain $f(x)$;
if $f(x) > 0$ **then**
    return $z = wA$;
else
    return $z = -wA$;
end

Remark

Note that x should also satisfy $z_j^T x = 0$ for $j \neq i$ by assumption that all our A_i are linearly independent (Theorem 2.9).

Correctness. We distinguish the two cases:

- **Case 1:** $z = wA$**.**

$$\langle z, x \rangle = w \langle A, x \rangle > 0 \implies \langle A, x \rangle > 0,$$

  so the ReLU is active, and

$$f(x) = w \langle A, x \rangle = \langle z, x \rangle > 0.$$

- **Case 2:** $z = -wA$**.**

$$\langle z, x \rangle = -w \langle A, x \rangle > 0 \implies \langle A, x \rangle < 0,$$

  so the ReLU is inactive, and

$$f(x) = 0.$$

Therefore:

$$f(x) = \begin{cases} \langle z, x \rangle > 0 & \text{if } z = wA, \\ 0 & \text{if } z = -wA. \end{cases}$$

This recovers the correct sign of z, and hence determines wA. □

Remark

This recovery procedure does not extend to deeper networks: ReLU is a universal gate, and one can implement a pseudorandom function with a low-depth ReLU network. By clever design, the ReLU network does not leak extra information when queried as real inputs, and only returns pseudorandom output at integer inputs. Such a function is, by definition, indistinguishable from random via oracle queries, ruling out efficient stealing.

References

[1] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *J. ACM*, 50(4):506–519, July 2003. ISSN 0004-5411. doi: 10.1145/792538.792543. URL https://doi.org/10.1145/792538.792543.

[2] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC '89, page 25–32, New York, NY, USA, 1989. Association for Computing Machinery. ISBN 0897913078. doi: 10.1145/73007.73010. URL https://doi.org/10.1145/73007.73010.

[3] Smitha Milli, Ludwig Schmidt, Anca D Dragan, and Moritz Hardt. Model reconstruction from model explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 1–9, 2019.

[4] Adam R Sitan Chen. Efficiently learning any one hidden layer relu network from queries. *ArXivorg*, 2021.