References

Differential Privacy

- Deep Learning with Differential Privacy (https://arxiv.org/pdf/1607.00133.pdf)
    - important paper
- DP-SGD explained (https://mukulrathi.com/privacy-preserving-machine-learning/deep-learning-differential-privacy/)
    - good article

Leaking training examples

- Extracting Training Data from LLM (https://arxiv.org/pdf/2012.07805.pdf)

Implementing Privacy

- TensorFlow Privacy (https://www.tensorflow.org/responsible_ai/privacy/tutorials/classification_privacy

Training a model with sensitive information

A key ingredient in the ability of a Machine Learning model to solve a task is the quantity and quality of training examples.

Each person has "sensitive" data (e.g., health, income) that

- might be valuable for training a ML model
- but whose "leaking" (public disclosure) would be harmful

The potential for harm implies that

- if an individual's sensitive data is to be used for training a ML model
- there needs to be a guarantee that an individual's sensitive data will not be disclosed

Beyond the potential for causing harm, removing the possibility of disclosure has benefit.

Imagine the potential for medical advancement

- if we were able to contribute our medical, genealogical, social data
- as examples for training models to predict likelihood
    - of disease
    - success of a treatment

Extracting Training Data from LLM (https://arxiv.org/pdf/2012.07805.pdf)

We first ask: do Large Language Models (LLM) leak training examples ?

Memorization of training examples (a prerequisite to the possibility of leaking) is often thought of as a consequence of model overfitting.

As the number of weights of an LLM increases, the potential for memorization would seem to increase.

On the other hand, overfitting is also associated with seeing the same example in many epochs of training.

- But LLM training is on very few (perhaps a single) epoch
- The datasets are so big, that even one epoch provides many gradient updates (one per mini-batch)
    - Reducing the need for multiple epochs

The paper tries to quantify how much training data leaks from the GPT-2 model

- The training data comes from public sources that are identified
    - mitigates the harm of the authors revealing leaked data
- But the exact training data is *not* open-source
    - can't verify a suspected leak against actual training examples
    - but a Web search can confirm the source

Methodology

The authors propose a number of techniques to extract potential training examples.

They are all based on using

- the LLM's autoregressive behavior
- to generate "candidate" sequences of text
- evaluating whether the candidate sequence is a training example

Recall the autogregressive LLM computes

$$p(\mathbf{y}_{(t)}|\mathbf{y}_{([1:t-1])})$$

That is, it

- creates a probability distribution $p(\mathbf{y}_{(t)})$ of the next (at position t) token
- conditional on all prior outputs $\mathbf{y}_{([1:t-1])}$
    - we are including the "seed" sequence in $\mathbf{y}$

The actual token $\mathbf{y}_{(t)}$ chosen for position t output is sampled from this distribution.

Common sampling strategies:

- Greedy sampling: the token from the Vocabulary $\mathbf{V}$ with maximum probability
$$\mathbf{y}_{(t)} = \mathbf{V}_j \text{ where } j = \operatorname{argmax} p(\mathbf{y}_{(t)})$$
- Top-n sampling: sample from the n $\mathbf{V}$ tokens with highest probability in $p(\mathbf{y}_{(t)})$

Candidate generation

The simplest approach to generating candidate sequences:

- Seed with the special `[START]` token
- Top-n sampling from probability distribution for sequences of length 256

The Top-n sampling favors high probability sequences, which limits diversity.

To increase diversity

- access the logits (the "scores" for each vocabulary item
- which are converted each logit l_j to probability $p(\mathbf{V}_j)$ using the Softmax

$$p(\mathbf{V}_j) = \text{softmax}(l_j) = \frac{\exp(l_j)}{\sum_{j'=1}^{|\mathbf{V}|} \exp(l_{j'})}$$

- "flatten" the output probabilities through a softmax "temperature" τ

$$\text{softmax}_\tau(l_j) = \frac{\exp(l_j/\tau)}{\sum_{j'=1}^{|\mathbf{V}|} \exp(l_{j'}/\tau)}$$

    - high temperature τ reduces the probability spread

But the simplest seed ［START］ creates sequences that are not conditioned on any "real" text.

The results may be highly likely in distribution yet not produce an actual training example

- which is a biased choice: using whatever selection criteria for examples was applied
    - e.g., unlikely to unconditionally generate a sequence with highly technical or foreign words

The authors also try seeds from actual Internet text.

The authors also observe on the influence of Prompt Engineering in order to get an LLM to generate good sequences.

Thus, the naive seeds (no Prompt Engineering) are less likely to generate training examples

- so estimates for how many examples a model has memorized are *under*-estimates

Here is a prompt that successfully recalls an individual's contact info:

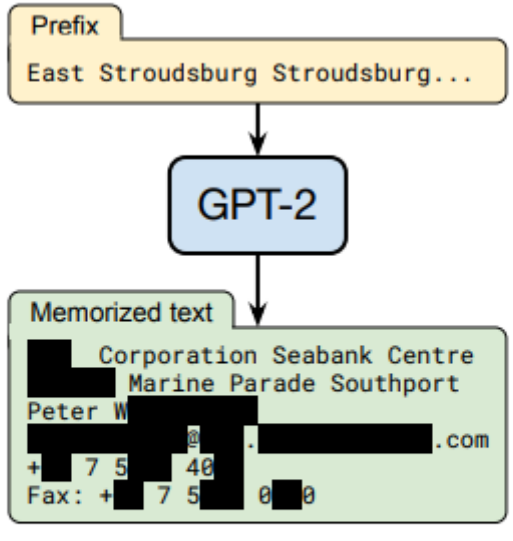

Figure 1: **Our extraction attack.** Given query access to a neural network language model, we extract an individual person's name, email address, phone number, fax number, and physical address. The example in this figure shows information that is all accurate so we redact it to protect privacy.

Attribution: https://arxiv.org/pdf/2012.07805.pdf#page=1

Candidate filtering

A large number of candidates needs to be reduced to those most likely to be training examples.

A variety of metrics are used for filtering.

- Low Perplexity $\mathcal{P}$

$$\mathcal{P}(\mathbf{y}_{[1:T]}) = \exp(-\frac{1}{T}\sum_{t=1}^{T}\log p(\mathbf{y}_{(t)}|\mathbf{y}_{([1:t-1])}))$$

  - $p(\mathbf{y}_{(t)}|\mathbf{y}_{([1:t-1])})$ is high for high confidence output $\mathbf{y}_{(t)}$
  - thus, for high confidence sequences
    - sum is high
    - perplexity is low (negative of sum is a power) for highly confident sequences

- Comparison against another LLM
    - High confidence does not equate to likelihood of being a training example
        - may just be a common sequence
            - the sequences over the digits 0 through 9
    - If a second LLM also computes high probability for a sequence
        - the sequence is less likely to be a training example
        - more likely to be a common sequence

Results

In the experiments in the paper

- about 33% of the candidates were correctly identified as being memorized

Remember: the candidates were chosen as having indicators of being memorized

- so this is the fraction of *suspicious* outputs that were, in fact, memorized

Some of the memorized content is not surprising, as the detailed analysis shows

Category	Count
US and international news	109
Log files and error reports	79
License, terms of use, copyright notices	54
Lists of named items (games, countries, etc.)	54
Forum or Wiki entry	53
Valid URLs	50
Named individuals (non-news samples only)	46
Promotional content (products, subscriptions, etc.)	45
High entropy (UUIDs, base64 data)	35
Contact info (address, email, phone, twitter, etc.)	32
Code	31
Configuration files	30
Religious texts	25
Pseudonyms	15
Donald Trump tweets and quotes	12
Web forms (menu items, instructions, etc.)	11
Tech news	11
Lists of numbers (dates, sequences, etc.)	10

Table 1: Manual categorization of the 604 memorized training examples that we extract from GPT-2, along with a description of each category. Some samples correspond to multiple categories (e.g., a URL may contain base-64 data). Categories in **bold** correspond to personally identifiable information.

Attribution: https://arxiv.org/pdf/2012.07805.pdf#page=9

Many examples belong to "often-quoted" categories

- Software/Publication licenses
    - required to be quoted verbatim as condition of use
- News headlines
- URL's
- Passages from religious texts, Wikipedia

But there were also some potentially harmful categories

- Names of individuals (other than those in the news)
- Contact info

Can we guarantee the absence of leaking ?

A naive approach (which **does not** guarantee against harm) is *anonymization*.

- For each example
    - replace Personally Identifying Information (PII - such asname, id number, address, phone number) by random values
    - keep the sensitive attributes in the example

The scenario that causes harm is one in which an adversary has *auxiliary* (external) information

- an "adversary" is able to link PII (via external information) to some of the attributes (sensitive or not) in an example
- DOB, gender and zipcode are together sufficient in may cases
- thereby "unmasking" the anonymized PII attributes

For example

- Crypto-currency (e.g., Bitcoin) is anonymous but with weak protection against harm
    - Your id for blockchain transactions is not PII
    - Having an account at a crypto exchange is **not** anonymous (KYC)
    - exchanging your Bitcoin for cash at an exchange
        - links information external to the Blockchain (the PII you disclose to the exchange)
        - to your transactions on the blockchain
- Want to know if your favorite celebrity is [a good tipper of taxi drivers ? (https://www.gawker.com/the-public-nyc-taxicab-database-that-accidentally-track-1646724546)](https://www.gawker.com/the-public-nyc-taxicab-database-that-accidentally-track-1646724546)

Is there a way that

- we, as individuals
- or the custodians of our sensitive data
- can contribute our sensitive data as training examples
- with a *guarantee* of no harm ?

Differential Privacy

Differential Privacy is a method that provides a *mathematical* guarantee of privacy.

It is beyond the scope of this module to describe in detail but we summarize

Privacy

Rather than just focusing just on the behavior of the trained model, we focus on the *entire process*

- training: data and training algorithm
- inference

We refer to the entire process as a *mechanism*.

Remember: training has multiple non-deterministic steps that affect the weights of the final model

- randomized split into train/test
- randomization of minibatches

Each different random choice will potentially change the final trained model.

There are also variants of Gradient Descent, each with choices of hyper-parameters

- e.g., learning rate (which may be a schedule rather than a constant)

Hence, the final weights of the model are dependent on many aspects of the process.

We want to make a mathematical statement about privacy that is relative to the entire mechanism rather than just the final (random) model.

ϵ Differential Privacy ($\epsilon - DP$) (https://arxiv.org/pdf/1607.00133.pdf#page=2)

In Machine Learning, the *Membership Inference* task

- is a binary classification
- as to whether a particular example e is in the training dataset for an ML model M.

Our ultimate goal is to be able to bound the probability of being able to determine Membership Inference for any example, given an ML model.

The mathematical notion of *Differential Privacy* is a way of mathematically stating whether a *mechanism* leaks data.

Let

- $\mathcal{M} : \mathcal{D} \mapsto \mathcal{R}$ be a mechanism
- $\mathcal{M}(D_1)$ be the mechanism $\mathcal{M}$ trained on set of examples D_1
- $\mathcal{M}(D_2)$ be the mechanism $\mathcal{M}$ trained on set of examples D_2 differing from D_1 by a single example e

$$D_2 = D_1 \cup \{e\}$$

- both implement functions with no argurments that have range $\mathcal{R}$

Mechanism $\mathcal{M}$ achieves ϵ *Differentially Private (DP)* if

$$p(\mathcal{M}(D_1) \in S) \quad \leq \quad \exp(\epsilon) * p(\mathcal{M}(D_2) \in S)$$

- for all subsets $S \subseteq \mathcal{R}$
- for all D_1 , D_2 differing by a single example

In words

- the functions computed by $\mathcal{M}$ when trained on the two nearly-similar datasets
- are indistinguishable
    - proportional difference in probability of the output for the two possible inputs is bounded by a factor of $\exp(\epsilon)$

Simple algebra allows us to express the equation in multiple equivalent forms

$$\begin{aligned}
p(\mathcal{M}(D_1) \in S) &\leq \exp(\epsilon) * p(\mathcal{M}(D_2) \in S) \\
\log \frac{p(\mathcal{M}(D_1) \in S)}{p(\mathcal{M}(D_2) \in S)} &\leq \epsilon \qquad \text{equiva} \\
\log p(\mathcal{M}(D_1) \in S) - \log p(\mathcal{M}(D_2) \in S) &\leq \epsilon \qquad \text{equiva}
\end{aligned}$$

Thus, if we can prove that $\mathcal{M}$ achieves ϵ

- the probability of determining Membership Inference is bounded
- the possibility of leaking a training example is bounded

The bound ϵ is called the *privacy budget*.

- it quantifies the privacy loss

One way to make a function differentially private is to add noise to the output.

- return

  $$\mathcal{M}(D) + \mathcal{N}(0, \sigma)$$

  rather than

  $$\mathcal{M}(D)$$

- The output is no longer a deterministic function of the training set.
- More noise is necessary for a smaller privacy budget.

The amount of noise depends on many aspects

- In a multi-layer network
  - layer l may implement a function achieving ϵ_l-DP
  - the privacy budget ϵ of the composed layers is a function of ϵ_l for all layers l
- Allowing repeated queries may require a larger privacy budget
  - we can estimate $\mathcal{M}(D)$ by averaging many samples of $\mathcal{N}(0, \sigma^2)$

Asides: technical details

Aside (history)

This definition was originally motivated by concern for privacy in databases

- D_1 , D_2 are databases differing by a single row
- $\mathcal{M}$ is a *query function* on the database
    - return attributes conditional on predicates of the attributes and database
    - this definition of query function would seem to be a concrete query with no parameters
        - all parameter-like values are bound to specific values
- Doing so allows the following theorem to be stated as a consequence of D_1 , D_2 rather than the query

Aside (relating to ML models)

Let's try to relate this to ML models.

Our usual definition of an ML model is as a function from features to label.

- for any feature vector
- rather than for *no arguments* as in the database "query" above
    - by analogy: $\mathcal{M}(D_1), \mathcal{M}(D_1)$ are the results of calling our ML model
        - on a pre-defined set of examples that are built into the function as arguments
        - rather than passed as a parameter
- it would be more interesting if $\mathcal{M}(D_1), \mathcal{M}(D_1)$
    - did not have the single example built into the function
    - but rather was an arbitrary parameter
    - i.e. if both were functions from $\mathcal{D} \mapsto \mathcal{R}$

Aside (randomized functions)

$\mathcal{M}$ can compute a *randomized* function

- hence the range is a set
- for example:
    - the output of the model is sampled from a probability distribution

Differentially Private Stochastic Gradient Descent (DP-SGD) (https://arxiv.org/pdf/1607.00133.pdf#page=3)

One necessary ingredient in creating models that achieve ϵ-DP

- is to ensure that all steps in the mechanism achieve ϵ-DP.

Here we discuss *Differentially Private SGD (DP-SGD)*

- a variant of SGD that achieve ϵ-DP

In Gradient Descent (for a model with weights Θ)

- there is a per-example loss for example i: $\mathcal{L}_{\Theta}^{(\mathbf{i})}$
- a gradient of the per-example loss with respect to weights: $\nabla_{\Theta}\mathcal{L}_{\Theta}^{(\mathbf{i})}$
- an update of weights Θ
    - in the negative direction of the per-example loss (negative: to *minimize* loss)
    - modulated by learning rate α

Aside

$\nabla_{\Theta}\mathcal{L}_{\Theta}^{(\mathbf{i})}$ is a vector (one element per element of Θ)

- so we should really refer to the magnitude of the *norm*

$$||\nabla_{\Theta}\mathcal{L}_{\Theta}^{(\mathbf{i})}||_2$$

An example i with a very large magnitude $\nabla_{\Theta} \mathcal{L}_{\Theta}^{(\mathbf{i})}$

- can affect the final Θ greatly
- potentially revealing the presence of example i in the training dataset

Thus, DP-SGD *clips* the norm of each example's gradient, returning

$$\nabla_\Theta \mathcal{L}_\Theta^{(\mathbf{i})} * \max(1, \frac{||\nabla_\Theta \mathcal{L}_\Theta^{(\mathbf{i})}||_2}{C})$$

where C is a bound on the maximum allowable magnitude.

Additionally: random noise is added to the clipped gradient to achieve privacy, returning

$$\nabla_\Theta \mathcal{L}_\Theta^{(\mathbf{i})} * \max(1, \frac{||\nabla_\Theta \mathcal{L}_\Theta^{(\mathbf{i})}||_2}{C}) + \mathcal{N}(0, \sigma^2)$$

Implementing Privacy in TensorFlow

TensorFlow has an easy to use

- module (https://www.tensorflow.org/responsible_ai/privacy/tutorials/classification_privacy) to implement ϵ-DP.

One begins by importing the modules and functions

```
import tensorflow_privacy

from tensorflow_privacy.privacy.analysis import compute_dp_sgd_privacy
```

And specifying the use of DP-SGD as the optimizer for training

```
optimizer = tensorflow_privacy.DPKerasSGDOptimizer(
    l2_norm_clip=l2_norm_clip,
    noise_multiplier=noise_multiplier,
    num_microbatches=num_microbatches,
    learning_rate=learning_rate)

model.compile(optimizer=optimizer, loss=loss, metrics=['accuracy'])
```

One can compute the privacy budget (ϵ achieved) by

```
    compute_dp_sgd_privacy.compute_dp_sgd_privacy(n=train_data.shape[0],
                                                  batch_size=batch_size,
                                                  noise_multiplier=noise_multip
lier,
                                                  epochs=epochs,
                                                  delta=1e-5)
```

In [2]:
```
print("Done")
```

```
Done
```