Generative Models for Effective ML on Private, Descentralized Datasets

Thalles Santos Silva

8 November 2022

Universidade Estadual de Campinas
Instituto de Computação

Generative Models for Effective ML on Private, Descentralized Datasets

Sean Augenstein
Google Inc.
saugenst@google.com

H. Brendan McMahan
Google Inc.
mcmahan@google.com

Daniel Ramage
Google Inc.
dramage@google.com

Swaroop Ramaswamy
Google Inc.
swaroopram@google.com

Peter Kairouz
Google Inc.
kairouz@google.com

Mingqing Chen
Google Inc.
mingqing@google.com

Rajiv Mathews
Google Inc.
mathews@google.com

Blaise Aguera y Arcas
Google Inc.
blaisea@google.com

Published as a conference paper at ICLR 2020

What we are going to cover

1. Introduction and motivation

2. Generative models as proxies to direct data examination

3. Using generative models instead of direct data inspection

4. Experiment: DP federated RNNs for generating natural language Data

5. Experiment: DP federated GANs for generating image data

Introduction and motivation

Introduction and motivation

- A key aspect of deep learning is the utilization of large datasets.
- Unlike shallow learning algorithms, deep learning shines when massive amounts of data are available.
- In fact, the behavior of a deep network is tightly coupled to the data it trains on.

Deep learning for real-world applications

- To improve real-world ML applications, engineers develop intuition about: datasets, models, and how the two interact.
- When facing problems either training or serving a ML model, an engineers' first step is to perform what we call: error analysis.

Error Analysis

- This process consists of **inspecting individual examples** to discover bugs, generate hypotheses, improve labeling, or similar.

Why manual inspection?

- Manual inspection of raw data, of representative samples, of outliers, of misclassifications, is an essential tool for:
  - Identifying and fixing problems in the data.
  - Generating new modeling hypotheses.
  - Assigning or refining human-provided labels.

What is the assumption here?

This is all good, but ML development is based on a big assumption…
which is…

Data centrality

Data centrality in ML development

- The centrality of data to ML development is recognized by the attention paid to data analysis, curation, and debugging.
  - Textbooks devote chapters to methodologies for “Determining Whether to Gather More Data”
  - The literature conveys practical lessons learned on ML system anti-patterns that increase the chance of data processing bugs

What if I am developing in a FL context?

In general, an assumption of unrestricted access to training or inference data is made.

One of the main assuptions of ML development

- However, manual inspection of raw data is problematic for privacy-sensitive datasets and impossible in FL setups
- In Federated Learning:
    - The developer and the global model sits in a server, while the raw traning data is stored in the edge, distributed across a fleet of devices.
    - The engineer only has access to aggregated outputs such as metrics or model parameters.

How can we effectively debug ML models when training data is privacy sensitive or decentralized?

Generative models as proxies to direct data examination

Generative models as proxies to direct data inspection

- The idea: to use auxiliary privacy-preserving generative models, as a proxy to perform direct data examination for debugging data errors in training and inference.
- Debugging tasks that require data inspection can be done by generating synthetic examples from a privacy-preserving federated generative model.

Common problems in deep learning development

- The paper describes six common tasks (T1–T6) where a developer would typically use direct data access.
- The choice of these tasks is validated by Humbatova et al. (2019), a recent survey providing a taxonomy of faults in deep learning systems
- Two of the largest classes of faults are:
  - Preprocessing of Training Data
  - Training Data Quality

Six common issues in ML development where engineers usually need direct data access.

Table 1: ML modeler tasks typically accomplished via data inspection. In Section 2.1 we observe that selection criteria can be applied programmatically to train generative models able to address these tasks.

Task		Selection criteria for data to inspect
T1	Sanity checking data	Random training examples
T2	Debugging mistakes	Misclassified examples (by the primary classifier)
T3	Debugging unknown labels/classes, e.g. out-of-vocabulary words	Examples of the unknown labels/classes
T4	Debugging poor performance on certain classes/slices/users	Examples from the low-accuracy classes/slices/users
T5	Human labeling of examples	Unlabeled examples from the training distribution
T6	Detecting bias in the training data	Examples with high density in the serving distribution but low density in the training distribution.

Sanity checking and model debugging (T1–T4)

T1	Sanity checking data	Random training examples

- Engineers will often inspect some random examples and observe their properties before training a model (T1);
  - It might be the first step in debugging.
- Typical things to look at are:
  - Are the size, data types, and value ranges as expected?
  - For text data, are words or word pieces being properly separated and tokenized?

Sanity checking and model debugging (T1–T4)

T2	Debugging mistakes	Misclassified examples (by the primary classifier)

- When a model is misbehaving, looking at a particular subset of the input data is natural.
- For example, in a classification task, an engineer might inspect misclassified examples to look for issues in the features or labels (T2).

Sanity checking and model debugging (T1–T4)

T3	Debugging unknown labels/classes, e.g. out-of-vocabulary words	Examples of the unknown labels/classes

- For tasks where the full set of possible labels is too large, e.g. a language model with a fixed vocabulary, an engineer might examine a sample of out-of-vocabulary words (T3)

Sanity checking and model debugging (T1–T4)

T4	Debugging poor performance on certain classes/slices/users	Examples from the low-accuracy classes/slices/users

- In production, it is important to monitor accuracy over fine grained slices of data, such as, by country, by class label, by time of day, etc.
- If low accuracy is observed on a slice, it is natural to examine examples selected from that segment of the data (T4).
  - For example, if training on data which can be grouped by user, it is natural to look at data from users with low overall accuracy.

Data labeling (T5)

T5	Human labeling of examples	Unlabeled examples from the training distribution

- Supervised learning problems require labeled data.
- Usually, human annotators inspect and manually assign class labels to observations.
- Because FL systems do not allow users' data to be sent to the cloud for labeling, we could synthesize realistic, representative examples from the decentralized data and label them (T5).

Detecting bias in training data (T6)

T6	Detecting bias in the training data	Examples with high density in the serving distribution but low density in the training distribution.

- The distributional difference between the datasets is a source of bias (T6).
- Training generative models on client devices might help us understand the differences in the training/inference data distribution between servers and clients.
- These examples—even at low fidelity, could indicate whether a model is fair and point to where additional training data collection is required.

Using generative models instead of direct data inspection

General methodology

- In a situation where an engineer would inspect examples based on a particular criteria (Table 1), this criteria is expressed as a programmatic data selection procedure used to construct a training dataset for a generative model.
- For FL, this might involve both selecting only a subset of devices to train the model, or filtering the local dataset held on each device.

The paper considers three tasks: T2, T3 and T4

The work combines three technologies:

- Generative Models.
- Federated Learning (FL).
- Differential Privacy (DP).

Differentially private federated generative models

- To work with decentralized data, generative models are trained via FL
  - Raw user data never leaves the edge device.
- A randomly chosen subset of devices download the current model, and each locally computes a model update based on their own data.

Differentially private federated generative models

- Model updates are sent back to the coordinating server, where they are aggregated and used to update the global model.
- Differential privacy ensures generative models will not memorize the training data.

Experiment: DP federated RNNs for generating natural language Data

An application of debugging during training with RNNs

- Recurrent Neural Networks (RNNs) are a ubiquitous form of deep network, used to learn sequential content (e.g., language modeling).
- An interesting property of RNNs is that they embody both discriminative and generative behaviors in a single model.

RNNs as discriminative models

- Given a sequence of tokens $x_0, \ldots, x_i$ and their successor x_{i+1} the RNN reports the probability of x_{i+1} conditioned on its predecessors (based on observing dataset D).
- x_0 Typically, is a special 'beginning-of-sequence' token.

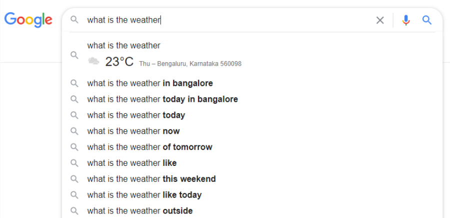
RNNs as generative models

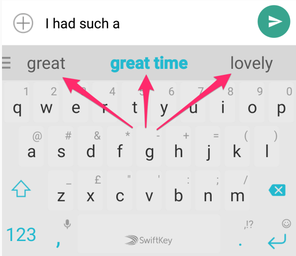
- The probability distribution of a token can be used to generate the next token
- The process can be repeated to generate additional tokens in the sequence.
- In this way, RNN word language models (word-LMs) can generate sentences and RNN character languagemodels (char-LMs) can generate words.

$$p(x_n, x_{n-1}, ..., x_1, x_0) = \prod p(x_{i+1}|x_i, ..., x_0; D) \cdot p(x_0|D) \tag{1}$$

An application of a mobile keyboard app

- Consider a mobile keyboard app that uses a word-LM to offer next-word predictions to users based on previous words.





An application of a mobile keyboard app

- The app takes as input raw text, performs preprocessing (e.g. tokenization and normalization), and then feeds the processed list of words as input to the word-LM.
- The word-LM has a fixed vocabulary of words V.
- Any words in the input outside of this vocabulary are marked as "out-of-vocabulary" (OOV).

Suppose a bug is introduced in tokenization which incorrectly concatenates the first two tokens in some sentences into one token.

A tokenization bug...

- E.g., '`Good day to you.`' is tokenized into a list of words as *['Good day' , 'to' , . . .]*, instead of *['Good' , 'day' , 'to' , . . .]*.
- Because these concatenated tokens do not match words in vocabulary *V*, they will be marked as OOVs.
  - OOVs occur at a relatively consistent rate
  - This bug would produce a spike in OOV frequency (metric to be tracked)

If we were in the a centralized environment...

- Some of the OOV tokens could be inspected, and the erroneous concatenation realized.
- But here the dataset is private and decentralized, so inspection is precluded.

Simulation and solution

- To simulate the scenario, authors used a dataset derived from Stack Overflow questions and answers.
- It provides a realistic proxy for federated data, since we can associate all posts by an author as a particular user.
- They artificially introduced the bug in the dataset by concatenating the first two tokens in 10% of all sentences, across users.

Training the generative models

- Two federated generative models are used in complementary fashion for debugging.
    - The **primary** model, the DP word-LM trained via FL for next word prediction.
    - The **auxiliary** model for **T3**, a DP char-LM trained only on OOV words in the dataset

Training the generative models

- Both the DP word- and char-LMs are trained independently on the bug-augmented and bug-free dataset, producing four trained models in total.
- These models are now used to synthesize useful and complementary information to debug the primary model.

Debugging using the word-LM trained on OOV tokens

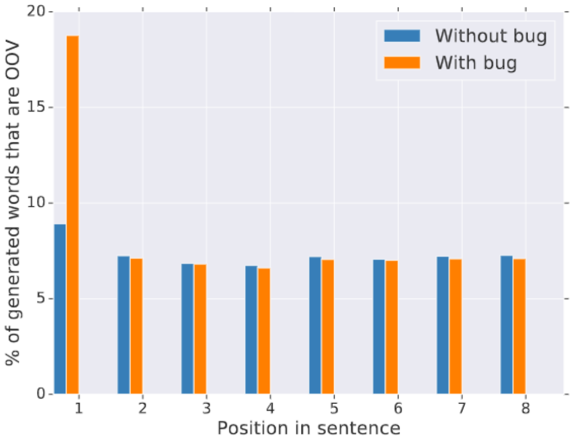


Figure 1: Percentage of samples generated from the word-LM that are OOV by position in the sentence, with and without bug.

Using the char-LM to generate OOV words

Table 2: Top 10 generated OOV words by joint character probability (computed using Equation 1), with and without the bug. Number accompanying is joint probability. The model is trained with case-insensitive tokenization.

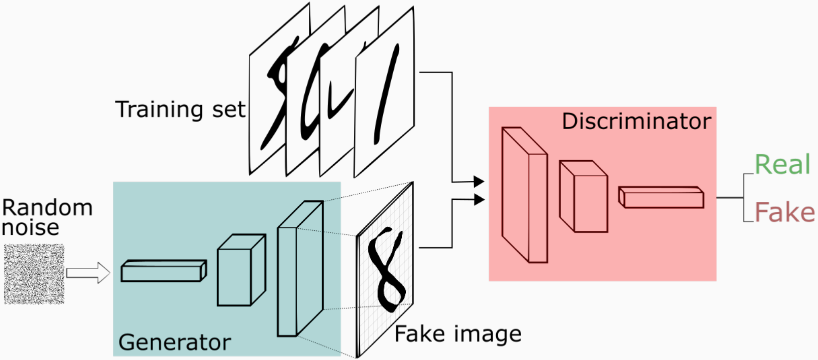
Without bug (prob 10^{-3})		*With 10% bug (prob 10^{-3})*	
`regex`	5.50	`i have`	8.08
`jsfiddle`	3.90	`i am`	5.60
`xcode`	3.12	`regex`	4.45
`divs`	2.75	`you can`	3.71
`stackoverflow`	2.75	`if you`	2.81
`listview`	2.74	`this is`	2.77
`textbox`	2.73	`here is`	2.73
`foreach`	2.34	`jsfiddle`	2.70
`async`	2.27	`i want`	2.39
`iis`	2.21	`textbox`	2.28

Experiment: DP federated GANs for generating image data

Debugging a federated vision classifier in production using GANs

- Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a SOTA form of deep generative model, particularly in the image domain.
- GANs work by alternately training two networks.
    - The **generator** maps a random input vector in a low-dimensional latent space into a high-dimensional output like an image.
    - The **discriminator**, judges whether an input image is ”real” (from orginal dataset) or ”fake” (from the generator).

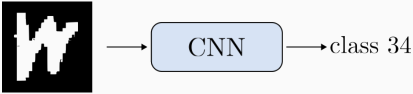
Generative adversarial networks - GANs



- Each network tries to defeat the other;
- The generator's training objective is to create content that the discriminator is unable to discern from real content.
- The discriminator's training objective is to improve its ability to discern real content from generated content.

An application of a mobile banking app to scan checks

- Consider the scenario of a banking app that uses the mobile phone's camera to scan checks for deposit.
- This app:
    1. Takes raw images of handwriting,
    2. does some pixel processing, and
    3. feeds the processed images to a pre-trained on-device CNN to infer labels for the handwritten characters

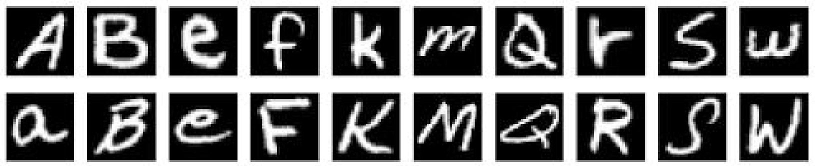


An application of a mobile banking app to scan checks

- This CNN is the ”primary” model.
- In production, an engineer can monitor its performance via metrics like user correction rate.
    - i.e. how often do users manually correct letters/digits inferred by the primary model, to get coarse feedback on accuracy.

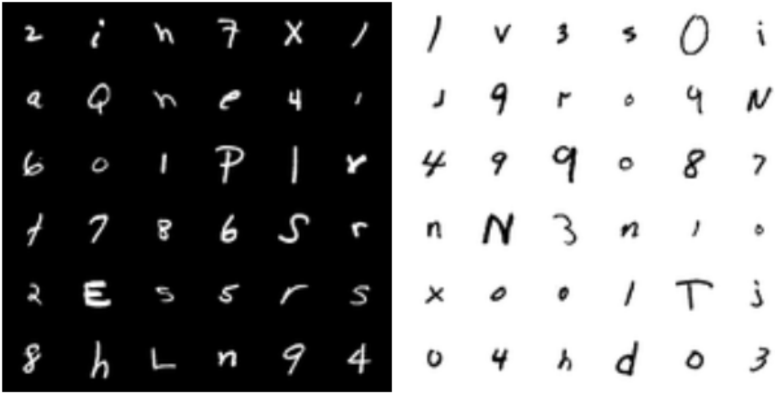
The EMNIST dataset

- To simulate users processed data, the authors used the Federated EMNIST dataset (Caldas et al., 2018).
    - 10 Digits
    - 26 Uppercase letters
    - 26 Lowercase letterr
    - Total: 731,668 images



Suppose a new software update introduces a bug that incorrectly flips pixel intensities during preprocessing, inverting the images presented to the primary mode

The pixel inversion bug

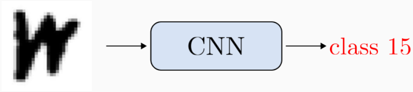


(a) Expected.
(Without Bug)

(b) Inverted.
(Bug)

An application of a mobile banking app to scan checks

- This change to the primary model's input data causes it to incorrectly classify most handwriting.



- As the update rolls out
  - The user correction rate metric spikes
  - The developer knows there is a problem, but does not know about its nature.

An application of a mobile banking app to scan checks

- If the data were public and inference performed on the app's server, the misclassified handwriting images could be inspected, and the pixel inversion bug realized.
- But this cannot be done when the data is private and decentralized.

Using gans as proxies to debug generative models

- They trained two DP federated GANs:
    - One on a subset of images that tents to perform best when passed to the primary model.
    - One on a subset of images that tends to perform worst.
- By contrasting images synthesized by the two GANs, we can understand what is causing degraded classification accuracy for some app users

Synthesized images by primary and auxiliary models

(a) Trained on high-accuracy users. (b) Trained on low-accuracy users.

The End

Thank you!

Bibliography

- Augenstein, S., McMahan, H. B., Ramage, D., Ramaswamy, S., Kairouz, P., Chen, M., Mathews, R., et al. (2019). Generative models for effective ml on private, decentralized datasets. *arXiv preprint arXiv:1911.06679*.
- Cohen, G., Afshar, S., Tapson, J., & Van Schaik, A. (2017). Emnist: Extending mnist to handwritten letters. *2017 international joint conference on neural networks (IJCNN)*, 2921–2926.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144.
- Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., & Tonella, P. (2020). Taxonomy of real faults in deep learning systems. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 1110–1121.