CS230

Incorporating Differential Privacy Techniques into Neural Networks

Tim Gianitsos
Department of Computer Science
Stanford University
tgianit@stanford.edu

Abstract

Differential privacy is a formalism that allows us to quantify privacy. Techniques that employ differential privacy ensure the that we can compute useful statistics about an entire dataset with minimal risk that an individual can be identified. In a machine learning context, information about individual training examples may exist in the model parameters, and clever analysis of the model will expose this information. By using differentially private techniques during the training process, we can bound the amount of personally identifiable information that can leak from our model.

1 Introduction

1.1 Motivation

Imagine you have access to a dataset from a few million people with only the following information: birthdays, genders, medical histories, and zip codes. However, you are not given any names, or other identifying information. The question is, does the mayor exercise?

This is the kind of question that Latanya Sweeney was able to answer from some publicly available voter registration and insurance data. Remarkably, 87% percent of U.S. citizens can be uniquely identified from their zip code, gender, and date of birth alone, and 53% of U.S. citizens can be identified with city, gender, and date of birth alone [1].

More data allows us to answer more questions. However, concomitant with more answering-power is also the risk of compromising privacy. We would like to be able to release data about individuals, or group statistics computed on those individuals so that useful insights can be gleamed from them. However, the Fundamental Law of Information Recovery [2] mandates that any information released about a data set increases the certainty in identifying individual records. If some statistic or model is representative of a dataset, it always reduces privacy.

1.2 Machine learning

The risk to privacy also applies to machine learning models, which can accumulate information about their training data. For example, SVMs include raw data points from the training set as part of their model [3]. Fredrikson et al. [4] use a *model inversion attack* on a trained neural network to demonstrate that mere access to a facial recognition model and a person's identifier (name or ID

CS230: Deep Learning, Spring 2019-2020, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

Figure 1: Images from The ORL Database of Faces. A model inversion attack reconstructs (left) data from the training set (right)

number) can allow one to reconstruct images of the person at a resolution granular enough to identify them. This is not limited to white-box access to the model itself, but can also apply to black-box settings where we can only see a model's inputs and prediction confidences. Fredrikson et al. were also able to perform a *model inversion attack* in a black-box setting with decision trees trained on people's risk taking propensity (e.g. *have you cheated on a spouse?*, and *do you smoke?*) [4]

There are two constraints that we are trying to optimize. First, we want to maximize privacy. Second, we want to maximize accuracy of our computation on the data. The goal of differential privacy is to skew data in mathematically rigorous ways so that an adversary's certainty about individual records is increased only marginally while at the same time the data fidelity remains high.

2 Differential Privacy

2.1 Definition

We say that a randomized mechanism $\mathcal{M}$ is "ϵ-differentially private" if it satisfies the following inequality[5]

$$\frac{P(\mathcal{M}(D_1) \in S)}{P(\mathcal{M}(D_2) \in S)} \leq e^{\epsilon} \tag{1}$$

where D_1 and D_2 are datasets that differ in only one element. This means that if we have two *marginally* different datasets, the probability that some quantity is computed on one dataset should not differ much from the probability of computing the same quantity on the other dataset. The privacy loss ϵ is a theoretical bound that quantifies privacy. We choose it to constrain the universe of mechanisms available to operate on our data. As ϵ tends toward zero, the mechanism we choose is more stochastic so as to guarantee more privacy, but this costs us accuracy. As ϵ tends toward infinity, the result resembles that which we would have obtained if we never used differential privacy.

This definition is symmetric on the dataset (i.e. the dataset in the numerator can be swapped with the dataset in the denominator, but the inequality must still hold). Both sides of the inequality are strictly non-negative, so we can apply a logarithm. The inequality will still hold because logarithms are monotonic. That is, for any monotonic function f, and $a, b \in \mathbb{R}$

$$a \leq b \iff f(a) \leq f(b)$$

Therefore:

$$ln\frac{P(\mathcal{M}(D_1) \in S)}{P(\mathcal{M}(D_2) \in S)} \leq ln(e^{\epsilon})$$
$$ln(P(\mathcal{M}(D_1) \in S)) - ln(P(\mathcal{M}(D_2) \in S)) \leq \epsilon \tag{2}$$

2.2 Techniques

There are several techniques that can be applied during training that ensure privacy remains within a reasonable bound. Abadi et al. [6] use a differentially private version of PCA to reduce the size of

their data and improve training times. They also analyze the effect of hyper-parameters on privacy, though this effect is negligible.

For our approach, we will focus on applying differentially private stochastic gradient descent (DPSGD). This technique incorporates stochasticity in the gradient update. The stochasticity allows us to bound the sensitivity of the gradient to individual examples (i.e. high sensitivity to individual examples would violate our inequality {2}). For each gradient, we compute its per-example sensitivity, and if the sensitivity is too high, we scale down the gradient making sure that its norm doesn't exceed some threshold defined by ϵ. We use Facebook Research's[7] implementation of a `Privacy Engine` in *PyTorch* to compute the mathematical moments of the privacy loss function.

3 Experiments

3.1 Datasets

- **The ORL Database of Faces**
  Contains images of people for use in personal facial recognition systems. This was used in [4] to perform model inversion attacks.
  Classes: 40 individual people
  Distribution: Exactly uniform between classes (10 images per person)
  Split: Train: 280, Validation: 120
  Size: 112 x 92 pixel images (8-bit grayscale)
- **The MNIST dataset**
  Handwritten digits from the Modified National Institute of Standards and Technology.
  Classes: 10 digits (zero through nine)
  Distribution: Uniform between classes
  Split: Train: 60,000, Test: 10,000
  Size: 28 x 28

3.2 Approach

First, we present the effect that varying the privacy loss ϵ has on classifier performance. These results are obtained from the MNIST dataset.

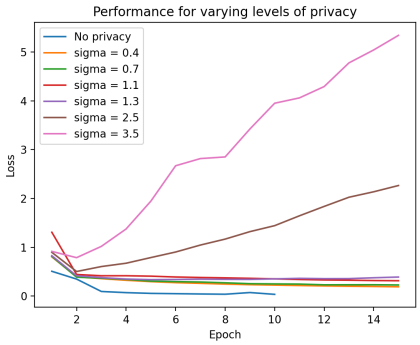
Second, we experiment on the ORL Database of Faces used by Fredrikson et al.[4]. We train a single layer softmax, a convolutional network, and a multilayer perceptron on the dataset, and for each we perform a *model inversion attack* which can extract training images from the weights of the model.

Third, we again train the softmax classifier on the ORL Database of Faces, but this time we train with differential privacy. We perform *model inversion attacks*, and observe the effect that the privacy loss ϵ has on our ability to recognize the extracted image. These results are consequential for the following reason: although Abadi et al. guarantee good performance for "modest" ϵ, they make no attempt to define "modest", nor do they associate ϵ with a human-interpretable metric. Our approach is to actually perform the kinds of attacks that differential privacy claims to protect against, and in this way we can qualify our choice of ϵ with a human-interpretable criteria; "does this value of ϵ result in a recognizable image?".

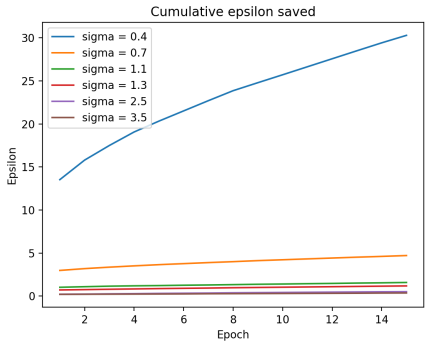
4 Results

4.1 Performance of Differentially Private Classifiers

Figure 2 shows the results on the MNIST dataset. The sigma corresponds to the standard deviation of the Gaussian that was sampled from (higher standard deviation means more privacy but less accuracy). The privacy loss ϵ is accumulated on each epoch, and large values of sigma correspond to less accumulation. The baseline algorithm with no privacy was able to attain 98.65% accuracy in fewer epochs than any of the differentially private methods. However, models with a modest amount of privacy guarantee were able to perform within a reasonable bound of this benchmark. A relatively large privacy bound of $\epsilon = 30$ attained 96.8% accuracy, just under a 2% difference. Perhaps more



(a) Stochasticity injected into the gradient update affects the learning rate required to converge



(b) The effect of a marginal increase to sigma can lead to large differences in the amount of cumulative privacy loss

Figure 2: Notice (left) that the model trained with no privacy achieved better performance in fewer epochs. Notice also (right) that the models which performed worst also reliably had the most privacy

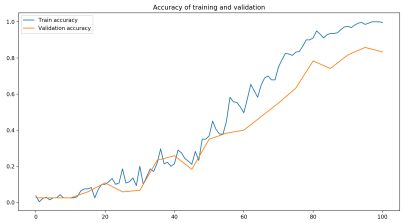
surprisingly, decreasing the ϵ loss from 30 down to 4.5 had negligible effect to accuracy (cost of 0.5% of accuracy) but the privacy guarantee improved by a factor of almost 7. The architecture was `conv -> relu -> maxpool -> conv -> relu -> maxpool -> fc -> relu -> cross entropy`.

4.2 Model Inversion Attacks

We present the *model inversion attack* in Appendix A originally described in [4]. This attack is supplied with a network trained on a facial recognition task, a label for an individual, and hyperparameters. It starts by initializing a blank tensor of the same dimension as the target image to reconstruct. It then computes the *cost* of this image, computed as the difference between the network's confidence that the image is of the desired individual, and the desired confidence (a value of 1 indicates perfect confidence). The gradient of the *cost* is computed with respect to the input image, and a gradient update is performed on *the image itself* (as opposed to the network, which stays fixed during this entire process). The original implementation[4] also supplies additional hyperparameters which allow the attack to end early when a local optimum is reached, but we neglect this for brevity. Figure 3 in the appendix displays results of *model inversion* on four different architectures.

4.3 Model Inversion Attacks with Differential Privacy

We now apply differential privacy to the training process to observe the effect it has on *model inversion attacks*. Figure 4 in the appendix shows the effect of varying epsilon ϵ on the resulting images. One may speculate that the image becomes obscured through increasing blurriness or distortion. However, it appears that the effect of differential privacy is merely increased white noise. We tested the *model inversion attack* with values of ϵ at roughly logarithmic intervals between 18 and 55,000,000. It appears that the noise makes the image incomprehensible at about $\epsilon = 430$. This means we can interpret $\epsilon = 430$ as sufficiently private for this dataset. However, we must also take into account the performance of the classifier at this level of ϵ. Fortunately, the performance is comparable to a classifier trained with with no differential privacy (i.e. a classifier with infinite ϵ). Figure **??** shows the accuracy of a softmax classifier trained with and without differential privacy. As expected, the differentially private classifier performed marginally worse (79% on validation set compared to baseline of 83%), but within a reasonable bound of performance accuracy. One desirable consequence of training with differential privacy noted by [6] is that it prevents overfitting. The concept of overfitting involves a model fitting too close to the idiosyncrasies of a dataset instead of a general pattern, therefore a differentially private classifier will by definition be more resistant because it cannot be effected by single examples 2.



(a) Accuracy of softmax classifier with *no* differential privacy ($\epsilon = \infty$)

(b) Accuracy of softmax classifier *with* differential privacy ($\epsilon = 430$)

5 Conclusion

We utilized differentially private techniques to evaluate the effect of ϵ on classifier performance. We performed *model inversion attacks* on several simple facial recognition classifiers to get a sense of how personally-identifiable information can be cleverly extracted from a model. We then qualified our choice of ϵ by judging the recognizability of extracted images, and we fortunately concluded that it is possible with current techniques to train a classifier that can obscure individual training examples to preserve privacy, while also maintaining high classifier performance.

References

[1] Latanya Sweeney. Simple demographics often identify people uniquely. *Health*, 671, 01 2000.

[2] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[3] Corinna Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 2004.

[4] T. Ristenpart M. Fredrikson, S. Jha. Model inversion attacks that exploit confidence information and basic countermeasures. *2015 ACM Conference on Computer and Communications Security (CCS)*.

[5] Cynthia Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer Verlag, July 2006.

[6] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, 2016.

[7] Facebook Research. Pytorch differential privacy. `https://github.com/facebookresearch/pytorch-dp`.

Appendix A Model Inversion Attack Code

```
def model_invert(label, max_steps, learning_rate, net):
  torch.set_grad_enabled(True)
  net.eval()
  x = torch.autograd.Variable(torch.zeros(size=(1, 1, 112, 92), dtype=torch.float,
      requires_grad=True), requires_grad=True)
  x_min = x
  c_min = float('inf')
  print('Model inversion')
  for step in trange(max_steps):
    net.zero_grad()
```

```
    cost = 1 - net(x)[0, label]
    cost.backward()
    x = torch.autograd.Variable(x - learning_rate * x.grad, requires_grad=True)
    if c_min > cost:
       c_min = cost
       x_min = x
plt.imshow(x_min.detach().numpy()[0][0], plt.cm.gray)
plt.show()
```

Appendix B Extracted Images from Model Inversion Attack

Appendix C Model Inversion on Classifiers Trained with Differential Privacy

(a) Image extracted from a single layer softmax network.

(b) Image extracted from a multi-layer perceptron

(c) Image extracted from a convolutional network with one output channel

(d) Image extracted from a convolutional network with three output channels

Figure 3: Model inversion is performed on four different network architectures. Notice the similarity with the results in figure 1 obtained by Fredrikson et al. They used additional methods to focus and sharpen their images; we neglect these methods in our implementation.

(a) σ of 0.001, ϵ of 55,000,000

(b) σ of 0.01, ϵ of 550,000

(c) σ of 0.1, ϵ of 5,600

(d) σ of 0.4, ϵ of 430

(e) σ of 1.1, ϵ of 85

(f) σ of 1.5, ϵ of 54

Figure 4: The identifiability of an extracted image from a softmax classifier changes with ϵ. The resulting image is almost completely unidentifiable at ϵ=430