CS 860: Algorithms for Private Data Analysis Fall 2020

Lecture 14 — Private ML and Stats: Modern ML

Prof. Gautam Kamath *Scribe: Gautam Kamath*

In today's lecture, we will discuss methods for differential privacy in modern machine learning settings. We will mainly cover two approaches, differentially private stochastic gradient descent (building off the coverage in the last lecture), and a method based on private aggregation of non-private learning algorithms. First, we give a quick primer on neural networks, and why they don't necessarily fit into our framework from the last lecture. The details will here will not be incredibly important (we will not depend on them afterwards), but it might serve as a useful background if not familiar.

Neural Networks

We will only describe the most fundamental class of neural networks, known as multilayer perceptrons. Other classes are similar, but swapping in different layers in place of what we describe here. As before, we have a dataset D of (x, y) pairs, a loss function ℓ, and the goal is to find a parameter vector θ which minimizes the following quantity:

$$\mathcal{L}(\theta, D) = \sum_{i=1}^{n} \ell(\theta, x_i, y_i).$$

Suppose that the data domain has $x \in \mathcal{R}^d$, and the label space is $y \in [k]$.

A neural network consists of a number of "layers," where the input to the first layer $v^{(0)}$ is equal to the input x, and the output of the ith layer is

$$v^{(i)} = f^{(i)}(W^{(i)} v^{(i-1)} + b^{(i)}),$$

where $v^{(i)} \in \mathbb{R}^{m_i}$, and the parameters $W \in \mathbb{R}^{m_i \times m_{i-1}}$, $b^{(i)} \in \mathbb{R}^{m_i}$. $f^{(i)} : \mathbb{R} \to \mathbb{R}$ is an "activation function." In a slight abuse of notation, we apply this function to a vector to indicate it is applied element-wise to the vector. These activation functions are typically non-linear in nature, which seems to add significant expressive power to neural networks. Some common non-linearities include the rectified linear unit (ReLU) $f(x) = \max\{0, x\}$, sigmoid $f(x) = \frac{1}{1+e^{-x}}$, and tanh $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. ReLU is generally the most popular activation function nowadays.

The output of the last layer o will be of dimension k, matching the number of possible labels. To convert from these values to a prediction for the label, we have the prediction vector $\hat{y} = \text{softmax}(o)$, where $\hat{y}_i = \frac{\exp(o_i)}{\sum_{i \in [k]} \exp(o_i)}$. Note that this normalizes $\hat{y}$, leading to the interpretation of this vector as a probability distribution. With all of these components in place, we have the loss function

$$\ell(\theta, x, y) = -\sum_{j=1}^{k} y_j \log \hat{y}_j,$$

where we take y to be the one-hot vector with a 1 in the index of the true label, and 0 elsewhere. Note that the dependence on the parameter vector θ is implicit in the definition of $\hat{y}$, and corresponds to the parameters W and b for each layer.

As mentioned before, this description corresponds to a very simple neural network. While neural networks have led to improved performance in complex machine learning tasks, they are not particularly "nice" in many ways, and we do not understand them well. For instance, while our discussion from last lecture showed that (stochastic) gradient descent could minimize the empirical risk, it depended on the loss function being convex, which is not the case here. Nonetheless, stochastic gradient descent has been seen to be empirically effective at optimizing neural networks. Furthermore, our privacy analysis required the loss function to be Lipschitz, in order to get an absolute bound on sensitivity of our gradients. We will simply clip the gradients to enforce a limit.

Differentially Private Stochastic Gradient Descent

In the previous lecture, we saw an approach for private SGD [BST14]. A work by Abadi, Chu, Goodfellow, McMahan, Mironov, Talwar, and Zhang [ACG+16] develops this method, making it more practical, and applying it to neural networks.

Recall that the work of [BST14] proposed iterating the following procedure:

1. Select a random point (x_i, y_i) from the dataset,
2. Compute the gradient of $\ell(\theta_t, x_i, y_i)$ and add Gaussian noise,
3. Take a step in the negative direction of the noised gradient.

Abadi et al. slightly modify this, resulting in the following procedure.

1. Sample a "lot" of points of (expected) size L by selecting each point to be in the lot independently with probability L/n,
2. For each point (x_i, y_i) in the lot, compute the gradient of $\ell(\theta_t, x_i, y_i)$ and "clip" it to have ℓ_2 norm at most C.
3. Average these clipped gradients and add Gaussian noise.
4. Take a step in the negative direction of the resulting vector.

As we can see, the second step deals with the fact that in these settings, we do not have a Lipschitz loss function, and thus the gradient may be unboundedly large. It deals with this in a rather crude way: simply clip each gradient so that its ℓ_2-norm is at most some pre-set bound C. Recall in the last lecture, for convergence of stochastic gradient descent, we required the estimate of the gradient to be unbiased. Since we clip the gradients, this loses information and biases the estimate, and this is no longer the case. However, since we are in a non-convex setting, we already had lost rigorous convergence guarantees. Nonetheless, this clipped SGD still seems to be effective in practice.

The other difference is the slightly different sampling scheme. Before, we chose a single sample from the training data and computed the gradient on this point. Recall this method of selection was important in our privacy calculation, allowing us to reduce the value of ε by a factor of n. Now, we choose a lot by selecting each point independently with probability L/n, where L is a hyperparameter determining the lot size. Note that even if we chose $L = 1$, this would not be

equivalent to the previous scheme, as we might choose more or fewer than 1 point. There are actually a number of different methods of randomly choosing a subset of the training data of size (roughly) L, including choosing each point independently with probability L/n (known as Poisson sampling), choosing L points without replacement, and choosing L points with replacement (where a point may be selected multiple times). While these methods of choosing minibatches may seem similar and arbitrary, they would have implications for the privacy analysis. In practice, people use the same method as in the non-private setting for choosing mini-batches: randomly permute the dataset and iterate over them in order. This is not equivalent to any of the aforementioned methods, and in particular correlation between mini-batches affects the privacy analysis. Nonetheless, it is common (and incorrect) to train like this and use the same privacy analysis as for the Poisson sampling case.

The last theoretical contribution of this work is a tighter privacy analysis. Let $q = L/n$ be the subsampling probability. Suppose we performed the analysis as done in the previous lecture. Recall this was a combination of the Gaussian mechanism's privacy guarantees, with subsampling amplification and advanced composition. If each step without subsampling would achieve (ε, δ)-DP, then the overall process would be (roughly) $(O(q\varepsilon\sqrt{T\log(1/\delta)}), T\delta)$. In constrast, Abadi et al. give a better analysis that achieves $(O(q\varepsilon\sqrt{T}), \delta)$. As shown in Figure 1, the savings can be substanial. Note that these are two ways of analyzing the exact same algorithm, one of which gives better guarantees. It is possible that a further analysis may lead to even better privacy guarantees for an algorithm which has already been run. Indeed, there have been improved analyses of differentially private SGD [WBK19, MTZ19], giving better guarantees than what was known before. This naturally raises the question of whether the analyses we have are "close" to optimal, or if there is room for significant improvement. A complementary recent work of Jagielski, Ullman, and Oprea [JUO20] develops privacy attacks, thus showing that we're not really that far off from the true privacy guarantees, as we can cook up instances in cases which roughly match the analysis.

To get an intuition for how we might surpass prior analyses, we recall the privacy loss random variable between random variables X and Y, which is distributed by drawing $t \sim Y$ and then outputting $L_{Y||Z} = \ln\left(\frac{\Pr[Y=t]}{\Pr[Z=t]}\right)$. Pure ε-differential privacy of a mechanism M says that $L_{M(X)||M(X')} \leq \varepsilon$ for all neighbouring datasets X and X'. Approximate (ε, δ)-differential privacy says that $L_{M(X)||M(X')} \leq \varepsilon$ for all neighbouring datasets X and X' with probability $1-\delta$. Another relevant quantity to consider are the *moments* of this random variable. Specifically, we can imagine bounds of the form

$$\ln \mathbf{E}_{t\sim M(X)}\left[\left(\frac{\Pr[M(X)=t]}{\Pr[M(X')=t]}\right)^{\lambda}\right] = \ln \mathbf{E}_{t\sim M(X)}\left[\exp\left(\lambda \ln\left(\frac{\Pr[M(X)=t]}{\Pr[M(X')=t]}\right)\right)\right] \leq \gamma$$

It is easy to see this implies a (ε, δ)-DP bound, by Markov's inequality:

$$\begin{aligned}\Pr_{t\sim M(X)}\left[\ln\left(\frac{\Pr[M(X)=t]}{\Pr[M(X')=t]}\right) \geq \varepsilon\right] &= \Pr_{t\sim M(X)}\left[\exp\left(\lambda \ln\left(\frac{\Pr[M(X)=t]}{\Pr[M(X')=t]}\right)\right) \geq \exp(\lambda\varepsilon)\right] \\ &\leq \frac{\mathbf{E}_{t\sim M(X)}\left[\exp\left(\lambda \ln\left(\frac{\Pr[M(X)=t]}{\Pr[M(X')=t]}\right)\right)\right]}{\exp\lambda\varepsilon} \\ &\leq \exp(\gamma - \lambda\varepsilon).\end{aligned}$$

Recall that the Gaussian mechanism offers a whole spectrum of approximate differential privacy bounds. Similarly, for each exponent λ, it is possible to compute a corresponding γ. It turns out

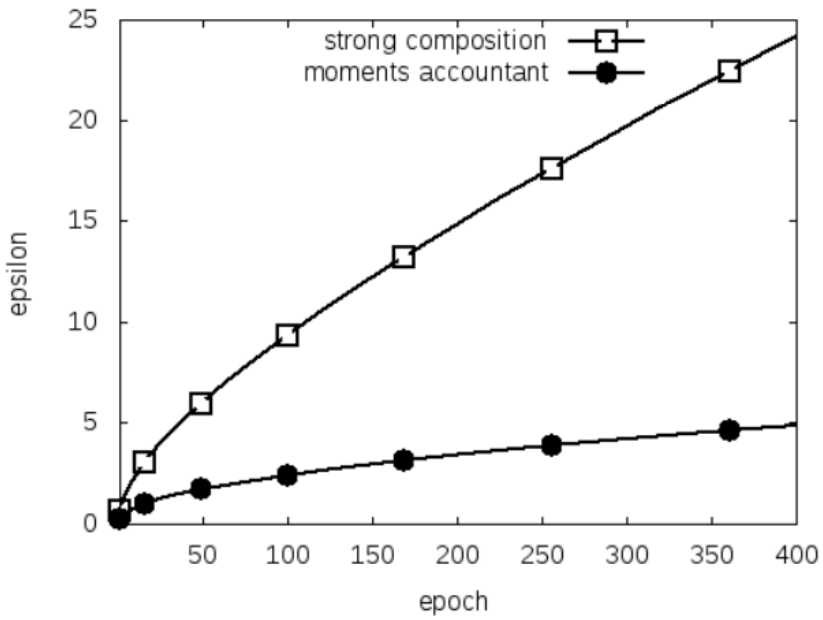

Figure 1: Figure from [ACG+16], showing the difference between using the improved analysis provided in their paper (called moments accountant) versus advanced composition.

that this latter representation will allow us to achieve a slightly tighter understanding of what happens when we apply composition of multiple queries.

It is possible to define an entire notion of differential privacy, known as Rényi Differential Privacy (RDP) [Mir17], based off of bounding moments of the odds ratio. RDP enjoys essentially all of the properties of differential privacy, including composition, being closed under post-processing, group privacy, and importantly, subsampling amplification. With these tools in place, the analysis of DPSGD can (abstractly) be outlined as follows:

1. For every λ simultaneously, obtain a corresponding γ in the expression above for the privacy guarantees of the subsampled Gaussian mechanism. That is, we perform the Poisson subsampling as described earlier, and apply the Gaussian mechanism with appropriate noise.
2. Apply composition of the privacy guarantees for the T iterations of the algorithm.
3. For a desired ε, compute the optimal associated δ by minimizing the expression $\exp(\gamma - \lambda\varepsilon)$ over all λ and their corresponding γ.

This approach is called *moments accountant*, and there exists code for computing it efficiently in most differential privacy libraries. Note that this is completely data independent, and given a set of parameters (namely, the number of datapoints n, the lot size L, the variance of the noise σ^2, the number of epochs T, and the target δ), it can compute the privacy guarantee in advance (i.e., the corresponding ε).

The results are passable, but not phenomenal. For instance, one common benchmark is image classification on the MNIST dataset. One state-of-the-art result (based on DPSGD, but with additional

tuning) gets 98.1% accuracy with $(2.93, 10^{-5})$-DP [PTS+20], whereas the best non-private methods get closer to 99.8% accuracy. A more challenging task is image classification on the CIFAR-10 dataset. Non-privately, methods are able to achieve 99.7% accuracy, while with $(7.53, 10^{-5})$-DP, we are able to achieve only 66.2% accuracy [PTS+20]. Neither the privacy guarantee or the accuracy guarantee are very compelling at the moment. There is clearly a lot of work to be done to improve the state of differentially private machine learning.

While DPSGD acts as a drop in replacement for SGD, there are many qualitative differences in the differentially private setting. We discuss some of them here.

One common complaint is that DPSGD is much slower than traditional SGD. The reason is that DPSGD requires one to clip each individual gradient in order to limit the sensitivity. Most modern machine learning frameworks are not built for this procedure, having methods which are optimized to use the GPU to compute the gradient over an entire mini-batch at once in parallel. The requirement of per-example gradients diverges from this standard, and the naive method of computing them would necessitate processing all points sequentially, thus losing all speedup granted by parallel processing on GPUs. As such, alternative algorithms for obtaining per-example gradients have been proposed [Goo15, RMT19], and sometimes alternative frameworks may be more efficient due to their low-level features, notably the recently-introduced framework JAX [SVK20].

As mentioned before, the ReLU is the most popular activation function in non-private machine learning, due to several convenient properties such as the ability to avoid "vanishing gradients" (an issue we will not get into here). However, in the differentially private setting, it appears that the tanh function (considered obsolete in the non-private setting) yields significant performance improvements [PTS+20]. This is one example showing that there can be benefits associated with modifying a network's architecture for the differentially private setting. On a similar note, non-private neural networks have grown exceptionally large, with the number of parameters growing into the hundreds of billions. This is because the size of a model is associated with higher "capacity," meaning that it can learn more functions. However, DPSGD requires the addition of Gaussian noise of magnitude proportional to the square root of the number of parameters. Thus, if we tried to run DPSGD on such a large model, our noise would drown out all signal. One must carefully choose the network architecture with this in mind – too small and the network wouldn't be able to represent the function, and too large and the noise would be overwhelming.

Hyperparameter tuning is a common challenge in machine learning tasks, and even more are introduced in the differentially private setting. For instance, how does one choose the learning rate, the lot size, the clipping norm, or the number of epochs? The canonical way to do this (non-privately) is to run a number of analyses on the training data with various hyperparameter settings, and choose the one which performs best on a validation set. Doing this in the differentially private setting would incur a cost in our privacy budget with every run, a cost which is currently omitted in most DP machine learning papers. This can be seen as pushing the methods to their limits, though they do not correspond to true privacy guarantees. Some methods have been proposed for hyperparameter optimization in a differentially private manner [LT19]. Another approach is to use public (non-sensitive) data that may have come from the same distribution as the private dataset. One can thus perform hyperparameter tuning on the public data, which will hopefully be suitable for the private data. An example of this is presented in [ACG+16], in which they treat the large CIFAR-100 dataset as public, and use this to train a neural network. They then freeze the majority of parameters in the model, and train the remainder privately on the sensitive CIFAR-10 dataset, achieving much better accuracy on the test set than without the CIFAR-100 training.

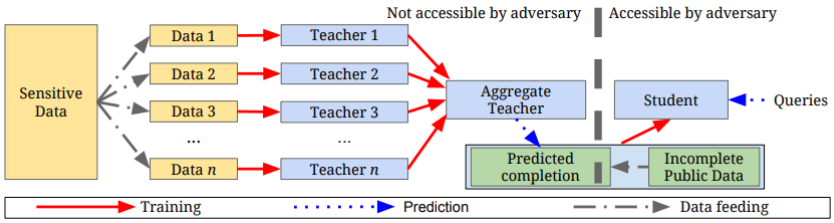

Figure 2: Figure from [PAE+17]. An illustration of the PATE pipeline.

Private Aggregation of Teacher Ensembles

Another approach involves private aggregation of teacher ensembles, or PATE for short. This approach was introduced in [PAE+17], refined in [PSM+18], and a theoretical version of this method was analyzed in [BTGT18] (presentation in this lecture will conflate all these methods but mostly focus on the latter, specific details are available in individual papers). In contrast to DPSGD, PATE can be used on top of an arbitrary non-private learning algorithm, making it more flexible – essentially, it only relies upon the non-private method being as accurate as possible. On the other hand, it requires a supply of public (i.e., non-sensitive) unlabeled data during the training process, which comes from the same distribution as the sensitive data. From a technical standpoint, PATE uses ideas of stability, as we discussed in lecture 10.

PATE works in the sample and aggregate framework, as introduced by Nissim, Raskhodnikova, and Smith [NRS07]. In this framework, a non-private algorithm is run on disjoint sets of data in order to obtain an answer to some query on each dataset – this is the "sample" step. The answers are combined (or as the name suggests, aggregated) in a differentially private manner, and we can output the result. The idea is, if the query has the same answer on most of the datasets (as perhaps it should, if they came from the same underlying population), then we can output this value at a low privacy cost, using the stability-based histogram mode as discussed in lecture 10 (some of these stability-based methods are formalized in [TS13]).

An illustration of the method is in Figure 2. The idea is the following. We split the data into n datasets, and use each dataset to train a separate non-private classifier. These will be known as the "teachers." Ideally, each of these teachers will then be highly accurate on the dataset.

Using this, it is not hard to make a single prediction under the constraint of differential privacy. If we want to label some point x, we run all n teachers on x to obtain non-private labels $\hat{y}_1, \ldots, \hat{y}_n$. If these teachers are highly accurate, (say) $0.99n$ of them will agree on the correct label. Since this leads to a large gap between the mode and the second most frequent label, we can release this value exactly using the stability-based mode from lecture 10. Remember how this works: we measure the gap between the two most frequently occurring elements and check if this is large (privatizing this quantity using the Laplace mechanism). If so, we can output the mode *exactly*, otherwise, we output randomly.

Instead of just one classification, suppose we wanted to answer T such classification tasks. Naively, the privacy cost would grow proportional to $\sqrt{T}$, using advanced composition. Instead, it's possible to use the sparse vector technique to ensure that we only pay $\sqrt{c}$, where c is the number of unstable predictions. Recall that sparse vector can answer a series of queries, only paying a cost based on the number of queries which are above the threshold. Our queries will be of the form "If we tried

to classify this image using these n classifiers, is the gap between the most and second most likely prediction they provide significantly large?" Setting things up the right way, we only have to pay when the answer is no.

So now we can answer a series of many classification tasks privately, but our privacy budget will still eventually be exhausted. However, the private predictions we have produced themselves have value. While the feature vectors are public, all the predictions are appropriately private, thus giving us a privatized synthetic dataset. This can be fed into a non-private learning algorithm to train a new model.

References

[ACG+16] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM Conference on Computer and Communications Security*, CCS '16, pages 308–318, New York, NY, USA, 2016. ACM.

[BST14] Raef Bassily, Adam Smith, and Abhradeep Thakurta. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '14, pages 464–473, Washington, DC, USA, 2014. IEEE Computer Society.

[BTGT18] Raef Bassily, Om Thakkar, and Abhradeep Guha Thakurta. Model-agnostic private learning. In *Advances in Neural Information Processing Systems 31*, NeurIPS '18, pages 7102–7112. Curran Associates, Inc., 2018.

[Goo15] Ian Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[JUO20] Matthew Jagielski, Jonathan Ullman, and Alina Oprea. Auditing differentially private machine learning: How private is private sgd? In *Advances in Neural Information Processing Systems 33*, NeurIPS '20. Curran Associates, Inc., 2020.

[LT19] Jingcheng Liu and Kunal Talwar. Private selection from private candidates. In *Proceedings of the 51st Annual ACM Symposium on the Theory of Computing*, STOC '19, pages 298–309, New York, NY, USA, 2019. ACM.

[Mir17] Ilya Mironov. Rényi differential privacy. In *Proceedings of the 30th IEEE Computer Security Foundations Symposium*, CSF '17, pages 263–275, Washington, DC, USA, 2017. IEEE Computer Society.

[MTZ19] Ilya Mironov, Kunal Talwar, and Li Zhang. Rényi differential privacy of the sampled gaussian mechanism. *arXiv preprint arXiv:1908.10530*, 2019.

[NRS07] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the 39th Annual ACM Symposium on the Theory of Computing*, STOC '07, pages 75–84, New York, NY, USA, 2007. ACM.

[PAE+17] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. In

Proceedings of the 5th International Conference on Learning Representations, ICLR '17, 2017.

[PSM+18] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable private learning with PATE. In *Proceedings of the 6th International Conference on Learning Representations*, ICLR '18, 2018.

[PTS+20] Nicolas Papernot, Abhradeep Thakurta, Shuang Song, Steve Chien, and Úlfar Erlingsson. Tempered sigmoid activations for deep learning with differential privacy. *arXiv preprint arXiv:2007.14191*, 2020.

[RMT19] Gaspar Rochette, Andre Manoel, and Eric W Tramel. Efficient per-example gradient computations in convolutional neural networks. *arXiv preprint arXiv:1912.06015*, 2019.

[SVK20] Pranav Subramani, Nicholas Vadivelu, and Gautam Kamath. Enabling fast differentially private sgd via just-in-time compilation and vectorization. *arXiv preprint arXiv:2010.09063*, 2020.

[TS13] Abhradeep Guha Thakurta and Adam Smith. Differentially private feature selection via stability arguments, and the robustness of the lasso. In *Proceedings of the 26th Annual Conference on Learning Theory*, COLT '13, pages 819–850, 2013.

[WBK19] Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. Subsampled rényi differential privacy and analytical moments accountant. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, AISTATS '19, pages 1226–1235. JMLR, Inc., 2019.