Private Global Generator Aggregation from Different Types of Local Models

Chunling Han[1,2(✉)] and Rui Xue[1]

[1] SKLOIS, Institute of Information Engineering, CAS; School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{hanchunling,xuerui}@iie.ac.cn
[2] Indiana University Bloomington, Bloomington, IN 47401, USA

Abstract. Generative Adversary Network (GAN) is a promising field with many practical applications. By using GANs, generated data can replace real sensitive data to be released for outside productive research. However, sometimes sensitive data is distributed among multiple parties, in which global generators are needed. Additionally, generated samples could remember or reflect sensitive features of real data. In this paper, we propose a scheme to aggregate a global generator from distributed local parties without access to local parties' sensitive datasets, and the global generator will not reveal sensitive information of local parties' training data. In our scheme, we separate GAN into two parts: discriminators played by local parties, a global generator played by the global party. Our scheme allows local parties to train different types of discriminators. To prevent generators from stealing sensitive information of real training datasets, we propose noised discriminator loss aggregation, add Gaussian noise to discriminators' loss, then use the average of noised loss to compute global generator's gradients and update its parameters. Our scheme is easy to implement by modifying plain GAN structures. We test our scheme on real-world MNIST and Fashion MNIST datasets, experimental results show that our scheme can achieve high-quality global generators without breaching local parties' training data privacy.

Keywords: GAN · Generator aggregation · Discriminator loss

1 Introduction

Generative Adversary Network (GAN) [9] is a thriving research topic, which can be used to generate fake (synthetic) data to replace sensitive data to be released for outside research [2]. Sometimes, data is distributed among different local parties, developing a global generator can help represent local parties to generate and release fake (synthetic) data.

We illustrate an example to explain why aggregating global generators are useful and will have many applications. We take Covid-19 as an example to explain how global generators can facilitate the understanding of this disease.

N. Park et al. (Eds.): SecureComm 2020, LNICST 336, pp. 324–335, 2020.
https://doi.org/10.1007/978-3-030-63095-9_21

A world organization wishes to use data from some countries to help other countries lack of data and research resources. Some medical research institutes from different countries are willing to contribute. However, due to the concern of patients' privacy, these research institutes are not willing to disclose their data. For this kind of situation, global generators can provide a solution. By developing a global generator, the world organization can generate synthetic data according to real data from those medical research institutes, then use the generated synthetic data to analyze and help other countries.

Here,"Global" can be interpreted from two aspects: firstly, the global generator can be aggregated from only **one** local party, represent this very local party to generate synthetic data; or it can be aggregated from **a group** of local parties, represent them to generate synthetic data reflecting the distribution of data from that group.

You may ask why not let those medical research institutes generate synthetic data themselves, and send the synthetic data to the world organization? The obstacles of this method are: these medical research institutes probably use different models and generate different quality of synthetic data. In addition, the actual amount of generated data needed is unknown at this moment, and every time when new synthetic data is needed, these medical research institutes need to be involved again. Therefore, a centralized global generator will be much easier to organize and manage.

Why Traditional Parameter Aggregation Fails. To aggregate a global generator from different local parties, there are some methods from federated learning can be referred to. Most of those traditional parameter based aggregation methods in federated learning are designed to average local models' parameters to get a global model [1,3,14,16]. They usually assume local parties and the global party develop exactly the same type of model and structure. To prevent private information leaking from local parameters, some works [1,16] release parameters under differential privacy [4–8] by adding noise to gradients. However, in fact, averaging local models' parameters is not always a good choice for model aggregation. Just simply taking the average of local parameters might not directly result in an accurate global model, let alone parameters with noise to achieve differential privacy. **Most importantly**, local parties might use different types of models, in which all different types and structures of parameters can not be averaged. These obstacles in traditional parameter based aggregation methods motivate our work.

Since we can not simply borrow traditional aggregation methods into global generator aggregation, in this paper, we propose a new global generator aggregation method.

To conventionally train a GAN, we usually train the discriminator and the generator together in one party, then use the generator to generate synthetic data. While, in our scheme, we **separate** discriminator and generator among local parties and the global party, allow **multiple** discriminators to one generator. Local parties train discriminators, which can have different types and structures. Meantime, the global party trains the generator. Because the gen-

erator might use gradients from discriminators to steal or extract local parties' sensitive information, we let local discriminators add Gaussian noise to their loss. To update global generator's parameters, the discriminators randomly select one discriminator as a representative, this discriminator will collect all discriminators' noised loss, use the average to calculate gradients and help the global generator to update its parameters. After training the global generator, the global party can represent local parties to generate synthetic (fake) data.

We test our scheme on real-world MNIST and Fashion MNIST datasets, experimental results show that our method can achieve high-quality global generators. To test the quality of generated samples output by the global generator, we use generated data to train deep learning models, we can achieve 98.02% and 88.54% accuracies for MNIST and Fashion MNIST test datasets respectively.

The contributions of our work are as follows:

- We solve a problem that aggregating global generators from different types of local discriminators. We achieve two main goals: global generators for local parties, suitable for different types of local discriminators and privacy protection for local parties' sensitive training data.
- We separate GAN into two parts: discriminators in local parties and a global generator in the global party. In this way, we can achieve global generators without access to local parties' private datasets. Since discriminators are probably in different types, parameters can not be used, therefore, we choose discriminator loss as vehicle to aggregate the global generators.
- We add noise to discriminators' loss computed on generated samples, by adding noise to discriminators' loss, we can prevent private information leakage from discriminators.

2 Preliminary

In this section, we briefly describe the basics of generative adversary networks.

Generative Adversarial Network (GAN) [9] consists of two models: generator G and discriminator D. Generator G takes random noise $z \sim p_z(z)$ as input, tries to output fake samples of data with distribution approximates real data's distribution $x \sim p_{data}(x)$. The discriminator D will estimate the probability that a sample is a real data comes from the training dataset rather than a fake data generated from G. These two models are simultaneously trained in a competitive way, the goal of GAN is training G and D playing a two-player minmax game with the value function V(G, D):

$$\min_G \max_D V(G,D) = E_{x\sim p_{\text{data}}(x)}[\log(D(x))] + E_{z\sim p_z(z)}[1-\log(D(G(z)))]$$

3 Our Approach

In this section, we illustrate our approach to global generator aggregation. We design our generator aggregation method for local parties, even they develop different types of discriminators and different structures of parameters.

3.1 Role of Models

There are two roles in our scheme, local parties and the global party. Local parties posses sensitive datasets, the global party is in charge of generator aggregation.

We consider an honest but curious global party, who participates by rules but always wants to steal privacy information from local parties.

We also consider honest but curious local parties, they participate in the system honestly but also want to steal sensitive information from other local parties. They might collude with others but will not destroy their collaboration of aggregation.

3.2 Global Generator Aggregation

In our scheme, we allow local parties to develop different types of discriminators. The global party will develop a global generator generating fake data for discriminators. The global generator and local discriminators form a GAN.

Our scheme is described in Algorithm 1. To train the GAN, **in every training epoch *t***, the generator generates a batch of fake data fake_data_t from random noise $z \sim p_z(z)$, feeds the generated data to discriminators. Every discriminator is trained on a batch of its real data and fake data fake_data_t. Then discriminators are set to be untrainable. Next, the generator generates a new batch of fake data fake_t from noise $z \sim p_z(z)$, feeds the generated data fake_t to the untrainable discriminators. Every discriminator $\{D_i\}_n$ computes the loss function on fake_t as g_loss_i. The output of discriminators are binary class classification (real or fake), we empirically assume all discriminators use binary cross entropy loss function.

To prevent privacy leakage from discriminators, discriminators will add noise to loss g_loss_i:

$$g\_loss_i \leftarrow g\_loss_i + \mathcal{N}(0, \sigma) \tag{1}$$

Where Gaussian noise has distribution with mean 0 and standard deviation σ (we will discuss the noise level σ later).

To compute gradients, those discriminators randomly select one discriminator as a representative to collect other discriminators' noised loss and average all loss values:

$$g\_loss_t = \frac{1}{n} \sum_{1}^{n} g\_loss_i \tag{2}$$

Next, the representative discriminator will use the averaged loss as GAN loss to do backpropagation and compute the gradients. Then the global generator will update its parameters according to the gradients.

Because in every epoch, we randomly choose one discriminator as a representative to collect other discriminators' noised loss and compute gradients for the combination of GAN, it can be seen as this discriminator transfers some knowledge about its sensitive training dataset to the global generator. Because every discriminator is selected by random, after several epochs, every local discriminator can have the same chance to be selected and transfer its knowledge

Algorithm 1. Global Generator Aggregation

Input: n discriminators $\{D_i\}_n$, a generator G, fake dataset fake_datat$_t$ for epoch $t \in (0, T)$, real datasets $\{\text{real}_1, \text{real}_2, ..., \text{real}_n\}$ for epoch $t \in (0, T)$
Parameter: Binary cross entropy loss for discriminators.

for epoch t in range $(0, T)$ **do**
  Generate fake data
  The generator G generates a batch of fake data: fake_data$_t \leftarrow G(z), z \sim p(z)$
  Train n discriminators
  Every discriminator D_i calculates the loss: d_i_real $\leftarrow D_i(\text{real}_i)$ and d_i_fake $\leftarrow D_i(\text{fake\_data}_t)$
  $d_i\_loss_t = \frac{1}{2}(d_i\text{\_real} + d_i\text{\_fake})$
  Every discriminator D_i computes gradients according to $d_i\_loss_t$ and updates its parameters.
  Set discriminators untrainable.
  Generate fake data
  The generator G generates a new batch of fake data from noise z. fake$_t \leftarrow G(z), z \sim p(z)$
  Compute loss
  Every discriminator D_i predicts on that batch of generated fake data and calculates the loss.
  $g\_loss_i \leftarrow D_i(\text{fake}_t)$
  Add noise
  $g\_loss_i \leftarrow g\_loss_i + \mathcal{N}(0, \sigma)$
  Average loss
  Randomly select one discriminator D_s to collect other discriminators' noised loss and average the loss.
  $g\_loss_t = \frac{1}{n}\sum_1^n g\_loss_i$
  Compute gradients
  The selected discriminator D_s and G calculate the gradients according to g_loss_t for the GAN.
  Generator updates parameters
  The generator G updates its parameters according to the gradients.
end for
Output: The global generator G.

to the global generator. Also, the GAN loss is the average of all discriminators' noised loss. Therefore, the global generator can capture the whole distribution features of all local parties' sensitive datasets.

3.3 How We Choose the Noise Level?

Because the scale of loss g_loss_i will change in every epoch of training, here we utilize an adaptive method to set the noise level σ.

As we can see, in every training epoch, the generator actually submits two batches of generated fake samples to discriminators. The first batch is used to train discriminators, while the second batch is the batch used to compute discriminators' loss g_loss_i.

In our scheme, we use the discriminator loss on the **first batch** of generated samples d_i_fake as the scale of noise added to loss g_loss_i, which is computed on the second batch of generated samples. We set $\sigma = 0.5 * d_i\_fake$.

As we can see, these two batches of generated fake samples come from the same state of the generator, the loss on the first batch is computed by discriminators before updating, while the loss on the second batch is computed by discriminators after updating. These two loss will be close to each other. Therefore we can use the first batch discriminator loss as a reference to add noise. To avoid adding too much noise, we add a ratio as 0.5 to the noise level. That is because the discriminator loss on the first batch is supposed to be a little bit higher than the loss on the second batch, because the loss for the second batch is computed by the updated discriminator, which is supposed to be better at classifying samples than the discriminator before updating. So we add a ratio (less than 1) to reduce the noise scale. Of course, this ratio can be adjusted in different discriminators.

3.4 Why Adding Noise Can Secure Our Scheme?

Firstly, let's explain why we should provide privacy protection for discriminators. Because discriminators are trained based on local parties' private training datasets, sensitive information of training data could be encoded or reflected into discriminators' parameters. On the other hand, the global generator always tries to steal sensitive information from discriminators and their private training datasets.

What the global generator can obtain are the gradients computed based on discriminators' parameters. Let's see how noised discriminators' noise can prevent the global generator from stealing sensitive information from discriminators' parameters.

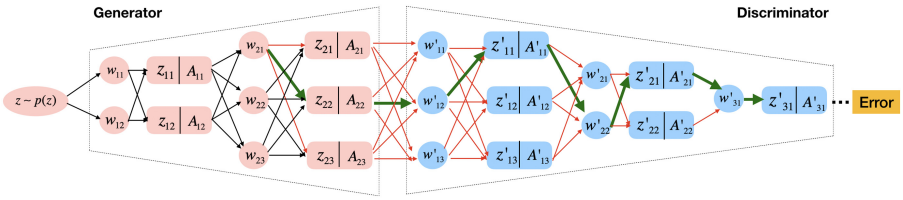

Fig. 1. An example of detailed GAN structure. (Color figure online)

We use a simple example shown in Fig. 1 to demonstrate how noised discriminators' loss will affect gradients. We only draw one discriminator here to represent the selected discriminator and the loss is averaged noised loss from all discriminators.

We take Generator's parameter W_{21} (shown in Fig. 1) as an example to compute gradient for W_{21}. We take one specific line (shown as the green line in Fig. 1) to demonstrate the computation. As we can see, the chain rule is:

$$\frac{\partial E}{\partial A'_{31}} \frac{\partial A'_{31}}{\partial Z'_{31}} \frac{\partial Z'_{31}}{\partial W'_{31}} \frac{\partial W'_{31}}{\partial A'_{21}} \frac{\partial A'_{21}}{\partial Z'_{21}} \frac{\partial Z'_{21}}{\partial W'_{22}} \cdots \frac{\partial Z_{22}}{\partial W_{21}} \tag{3}$$

As we can see, because E (discriminators' loss) in Eq. (3) is noised, $\frac{\partial E}{\partial A'_{31}}$ will be noised, so as $\frac{\partial A'_{31}}{\partial Z'_{31}}$, ... , so as $\frac{\partial Z_{22}}{\partial W_{21}}$, therefore the gradient for W_{21}: $\frac{\partial E}{\partial W_{21}}$ is noised. What the global generator obtained is the noised gradients.

Gradient for W_{11} will be a little bit different, because $Z_{11} = W_{11}X$, in which X are input samples, the gradient for W_{11}: ∂W_{11} will have X as coefficients. Notice that, those input samples X here are generated fake samples, not real samples. Even though the global generator might try to leverage well organized generated fake samples X to extract sensitive information from the gradient ∂W_{11}, because the gradient is noised, this intent will be handicapped due to the noise involved.

Therefore, adding noise to discriminators' loss can protect privacy of local parties' sensitive training datasets.

4 Evaluation

In this section, we evaluate the performance of our scheme on real-world datasets.

4.1 Implementation

We use deep convolutional generative adversarial network as our GAN structure. Deep convolutional generative adversarial network (DCGAN) [15] is an extension of GAN, in which generator and discriminator have deep convolutional network architectures.

We evaluate the performance of our scheme on MNIST and Fashion MNIST datasets. MNIST is a 10-class handwritten digit recognition dataset consisting of 60,000 training examples and 10,000 test examples [12], each example is a 28×28 size greyscale image. Similarly, Fashion MNIST is a 10-class dataset of fashion images, also consisting of 60,000 training examples and 10,000 testing examples [18], each example is a 28×28 size gray-level image. MNIST (produced in 1998) has been as a benchmark for machine learning and data science algorithms for years, and now Fashion MNIST (produced in 2017) serves as a replacement for the MNIST dataset for benchmarking machine learning algorithms.

We program our codes in Python, and execute them on Google Colab with free access to online GPU. We also use Tensorflow and Keras as backend. We develop different types of discriminators, they vary in numbers of layers and numbers of parameters. We equally split the training dataset among local parties, due to the limited number of training samples, we did not assume too many local parties, because more local parties will result in less training samples for every local discriminator and less accurate local discriminators can be achieved.

4.2 Experimental Results

We firstly aggregate a global generator from only one local party, in this case, the local discriminator will use the whole training datasets from MNIST (60,000 samples) and Fashion MNIST (60,000 samples). We train the global generator for 100 epochs with batch size as 256. After obtaining the global generator, we let it generate synthetic samples for MNIST and Fashion MNIST.

We also aggregate global generators from 5 and 10 different local discriminators, every local discriminator has 12,000 (5 local parties) and 6,000 (10 local parties) training samples from MNIST, 12,000 and 6,000 training samples from Fashion MNIST. We train the global generators for 200 epochs with batch size 128. After achieving the global generators, generators generate some fake samples.

In Fig. 2, we show some generated samples from two global generators. The left column are real samples from MNIST and Fashion MNIST. The middle column are generated samples from the global generator aggregated from one local party. The right column are from global generator aggregated from 5 local parties.

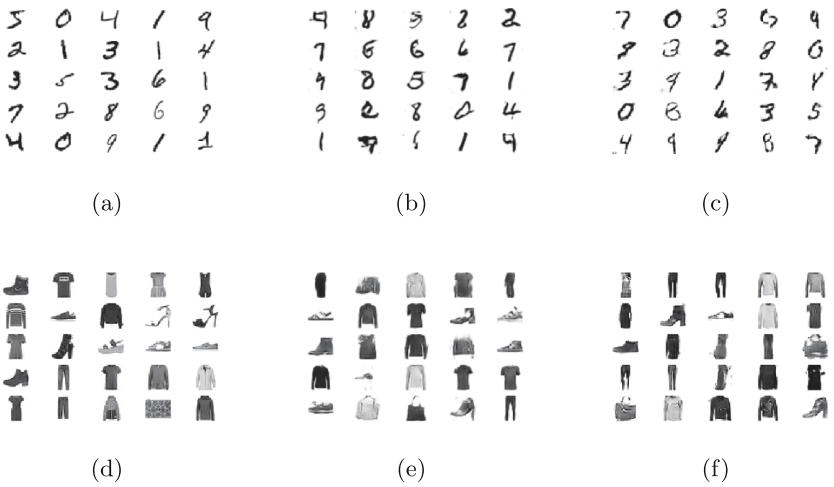

Fig. 2. Comparison between real samples and generated samples from global generators. The left column ((a), (d)) are real samples from MNIST and Fashion MNIST, the middle column ((b), (e)) are generated samples from the global generator aggregated from one local party, the right column ((c), (f)) are generated samples from the global generator aggregated from 5 local parties.

As shown in Fig. 2, generated samples look like real samples from MNIST and Fashion MNIST. Notice that, generated samples are a little bit blurry compared to real samples. The main reasons come from three aspects: firstly, GAN can not perfectly simulate real data, the quality of generated samples depend on the

GAN structure, the number of training epochs, and the training optimizer, etc. Secondly, because we split training datasets among local parties, the quality of generators will be affected by the amount of training data. Finally, the noise added to discriminators' loss will to some extent affect the ability of generators capturing features of real training data.

Compared with generated samples from generator aggregated from 5 local parties (right column of Fig. 2), with more training samples for local discriminators, generator aggregated from one local party produces higher-quality of generated samples (middle column of Fig. 2).

4.3 Performance Evaluation

After aggregating global generators, the global party can use those generators to generate synthetic data. To test the quality of generated synthetic data, we use generated samples to train deep learning models and test accuracies on real samples.

We let global generators generate the same amount of synthetic samples as training samples of MNIST and Fashion MNIST, so that, the global party can mimic the real training datasets of MNIST and Fashion MNIST.

We use generated samples to train deep learning models, then test these models on test datasets (10,000 test samples from MNIST and 10,000 test samples from Fashion MNIST).

To compare accuracies achieved by generated samples and local parties' real training data, we develop local models (CNNs) for local parties. Every local party trains its own CNN model on its dataset. We take 5 local parties as an example to show the accuracies of local models $(L_1, L_2, L_3, L_4, L_5)$ achieved, shown in Table 1. As we can see from Table 1, with 12,000 training samples, local models achieve average 97.81% and 88.26% accuracies for MNIST and Fashion MNIST.

Table 1. Local deep learning models' performance on their own datasets.

Accuracy	L_1	L_2	L_3	L_4	L_5	Average
MNIST	97.65%	97.40%	98.04%	98.21%	97.74%	**97.81%**
Fashion MNIST	88.10%	87.75%	87.59%	88.74%	89.10%	**88.26%**

We list test accuracies achieved by generated samples from global generators, local models' test accuracies and baselines in Table 2. The baselines are accuracies of machine learning models trained on the whole real MNIST and Fashion MNIST datasets and tested on real test datasets.

As seen in Table 2, generated synthetic samples from global generators can achieve accurate models. For MNIST and Fashion MNIST, when there are 5 and 10 local parties, models trained on generated synthetic samples achieve higher accuracies than local models trained solely on local parties' local training datasets. For example, when there are 5 local parties, local models trained on

Table 2. Accuracies achieved by generated samples from global generators.

Dataset	Local party	Local	Global	Baseline	Cmp to local	Cmp to Baseline
MNIST	1	99.20%	98.35%	99.20%	-0.85%↓	-0.85%↓
	5	97.81%	98.02%		+0.21%↑	-1.18%↓
	10	97.48%	97.65%		+0.17%↑	-1.55%↓
Fashion MNIST	1	92.40%	90.16%	92.40%	-2.24%↓	-2.24%↓
	5	88.26%	88.54%		+0.28%↑	-3.86%↓
	10	86.63%	86.87%		+0.24%↑	-5.53%↓

12,000 real samples achieve average 97.81% and 88.26% accuracies for MNIST and Fashion MNIST, while models trained on 60,000 generated samples from aggregated global generators can achieve 98.02% and 88.54% for MNIST and Fashion MNIST, increase 0.21% and 0.28% accuracies respectively. Similar results are shown for 10 local parties as well. These results indicate that aggregated global generators can generate high-quality synthetic samples.

On the other hand, when the global generator is aggregated from only one local party, compared with the local models with accuracies 99.20% and 92.40% trained on the whole training datasets of MNIST and Fashion MNIST, generated samples from global generators achieve less accurate models with 98.35% and 90.16% accuracies for MNIST and Fashion MNIST correspondingly. Generated synthetic samples bring 0.85% and 2.24% declines for MNIST and Fashion MNIST.

Compared with baselines for MNIST and fashion MNIST, which are achieved by whole real training datasets, generated samples from global generators are not as precise as real samples and achieve lower accuracies than baselines.

As a conclusion, from the experimental results, our scheme can achieve global generators with satisfying performance.

5 Related Works and Comparison

In this section, we illustrate some state-of-the-art works related to our study and compare our scheme with some of these related works.

There are some proposed methods for aggregating generators. Work [10] lets discriminators return intermediate feedback results of backpropagation for the generator to update. Work [13,17,19] use differentially private gradient descent (DP-SGD) to achieve differentially private GANs. Work [11] uses differential privacy on majority voting labelling plus a simple classifier to achieve a differentially private GAN.

Notice that, works [10,13] mentioned above only consider aggregating a global generator from same type and structure of discriminators, **while our scheme can generalize their methods and is also suitable for discriminators with different types and structures.** Moreover, in work [10], simply returning discriminators' intermediate feedback is not privacy-preserving for sensitive training datasets.

Works [11,13,17,19] use differential privacy during training, which need to use modified Tensorflow library (Tensorflow Privacy). Using this modified Tensorflow library can be extremely inefficient, according to open codes of work [17], training a differentially private GAN on MNIST can take over two hours on TPU.

We compare the accuracies achieved by generated samples from generators in our work and other three works [13,17,19] mentioned above. The comparison is presented in Table 3. Because these three related works actually achieve differentially private GANs (considered as the global generator aggregated from one local party), we only compare global generators aggregated from one local party. Due to lack of experimental results on Fashion MNIST dataset from these related works, we only list accuracies tested on MNIST dataset.

Table 3. Comparison of accuracies among three related works and ours.

Dataset	Scheme	Accuracy
MNIST	DP-GAN [19]	99.00%
	DP-CGAN [17]	88.16%
	Scalable DP-GAN [13]	80.92%
	Our scheme	**98.35%**

As shown from Table 3, our scheme can achieve higher accuracies compared with works [13,17]. With high privacy loss, work [19] can achieve slightly higher accuracy than our scheme.

6 Conclusion

Motivated by providing methods for global generator aggregation from different types of discriminators. We split GAN into two parts: discriminators in local parties and the global generator in the global party. Since parameters based aggregation fails, we use discriminator loss as vehicle to aggregate the global generator. We achieve two goals, we aggregate a global generator from different types of discriminators, and we achieve high-quality generators, from which generated samples would not reflect private features of local parties' sensitive training data. We test our scheme on two real-world datasets, experiments show that our scheme can achieve high-quality global generators.

References

1. Abadi, M., et al.: Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 308–318. ACM (2016)

2. Beaulieu-Jones, B.K., Wu, Z.S., Williams, C., Greene, C.S.: Privacy-preserving generative deep neural networks support clinical data sharing. Biorxiv **10**, 159756 (2017)
3. Bonawitz, K., et al.: Practical secure aggregation for privacy-preserving machine learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 1175–1191. ACM (2017)
4. Dwork, C., Lei, J.: Differential privacy and robust statistics. Stoc **9**, 371–380 (2009)
5. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_14
6. Dwork, C., Roth, A., et al.: The algorithmic foundations of differential privacy. Found. Trends® Theor. Comput. Sci. **9**(3–4), 211–407 (2014)
7. Dwork, C., Rothblum, G.N.: Concentrated differential privacy (2016). arXiv preprint arXiv:1603.01887
8. Dwork, C., Rothblum, G.N., Vadhan, S.: Boosting and differential privacy. In: 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, pp. 51–60. IEEE (2010)
9. Goodfellow, I., et al.: Generative adversarial nets. In: Advances in neural information processing systems, pp. 2672–2680 (2014)
10. Hardy, C., Le Merrer, E., Sericola, B.: Md-gan: Multi-discriminator generative adversarial networks for distributed datasets. In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 866–877. IEEE (2019)
11. Jordon, J., Yoon, J., van der Schaar, M.: Pate-gan: Generating synthetic data with differential privacy guarantees (2018)
12. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
13. Long, Y., Lin, S., Yang, Z., Gunter, C.A., Liu, H., Li, B.: Scalable differentially private data generation via private aggregation of teacher ensembles (2020). https://openreview.net/forum?id=Hkl6i0EFPH
14. Pathak, M., Rane, S., Raj, B.: Multiparty differential privacy via aggregation of locally trained classifiers. In: Advances in Neural Information Processing Systems, pp. 1876–1884 (2010)
15. Radford, A., Metz, L., Chintala, S.: Unsupervised representation learning with deep convolutional generative adversarial networks (2015). arXiv preprint arXiv:1511.06434
16. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, pp. 1310–1321. ACM (2015)
17. Torkzadehmahani, R., Kairouz, P., Paten, B.: Dp-cgan: Differentially private synthetic data and label generation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (2019)
18. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms (2017). arXiv preprint arXiv:1708.07747
19. Xie, L., Lin, K., Wang, S., Wang, F., Zhou, J.: Differentially private generative adversarial network (2018). arXiv preprint arXiv:1802.06739