Final Report

Final Year Project - CM3203 - 40 Credits

Generating Differentially Private Datasets Using Deep Learning (in collaboration with: Office for National Statistics)

Author:
Sam Wincott

Supervisor:
Dr. George Theodorakopoulos

May 15, 2020

Contents

1 Abstract

This project aims to implement and evaluate one method of creating synthetic datasets with differential privacy guarantees, by training a generative adversarial network in a differentially private manner. This project shows that high utility datasets can be generated with tight privacy bounds. The results from this work also demonstrate the relationship between utility and privacy in synthetic datasets.

2 Introduction

Following recent advances in deep learning, an increasing number of businesses, government organisations, and individuals are making use of vast amounts of data. The key ingredient to deep learning is data, and models learn from the statistical properties of these datasets. Datasets often cannot be shared freely between businesses and organisations however, as the privacy of the individuals in the dataset must be preserved. An enormous number of datasets held privately by institutions may have statistical insight to another institution though they have no reasonable way of accessing them.

One method of making use of this private data would be for the owner of the dataset to train their own deep learning models on the data privately, then release these models for anyone to use, without releasing the data. Two problems arise with this method however. Firstly, this approach is very restrictive on what people outside of the organisation can do, they can only use the model provided by the organisation, and not gain their own insights from the dataset. Secondly Fredrikson et al. (2015) and Hitaj et al. (2017) have shown that it is possible to reconstruct original examples from trained models and so the privacy of the dataset would not be protected.

Another method of making use of this private data is to de-identify, then release it. This allows people to gain their own insights from the data, however there are issues with this method also. Again, this method can be quite restrictive on what individuals outside of the organisation can do with it. When de-identifying data, fields may be randomised, aggregated or removed entirely from the dataset, this would make it difficult for a deep learning

model to learn from the data properly. This method is also liable to the reconstruction of individuals in the original dataset, as shown in Rocher et al. (2019).

Producing a synthetic dataset is another method of de-identifying and releasing data from an organisation. A synthetic dataset is one that is generated to mimic the statistical properties of an original dataset, and contains no 'real world' values. Though the synthetic datasets contain no values from the original, it may be possible to reconstruct them. This project will seek to generate a synthetic dataset while being able to quantify and minimise the privacy loss of the original dataset.

Many techniques have been proposed to quantify data privacy including Sweeney (2002), Li et al. (2007), and Machanavajjhala et al. (2007). For this project I will be using differential privacy (Dwork & Roth 2014), as it is widely accepted as a strong and rigorous standard of measuring privacy protection.

To generate synthetic datasets in this project, I plan to use generative adversarial networks (GANs) introduced in Goodfellow et al. (2014), which have been shown to be capable of producing high quality synthetic datasets. To train a GAN with differential privacy guarantees, I will build upon the work in Abadi et al. (2016) which presents a method of training neural networks in a differentially private manner and provides a method of measuring the privacy loss during training of the network.

The aim for this project is to be able to create a method of producing synthetic datasets, using GANs, while also being able to measure and fine tune the privacy protection of the original dataset.

3 Background and Related Work

3.1 Generative Adversarial Network

GANs are a recent development in machine learning, which aim to generate data sets by learning from an original (training) dataset. They comprise of two neural networks, the discriminator and the generator, which train and learn in an adversarial fashion. The generator takes random noise as input, and tries to output data that resembles the training data. The discriminator takes the training data and the generated examples as input, and tries to output whether it believes that this example is a 'real' example (i.e. from the training dataset), or a 'fake' example that has been created by the generator. It is adversarial in the sense that the two networks are competing against each other. The generator is trying to fool the discriminator, and the discriminator is trying to not be fooled.

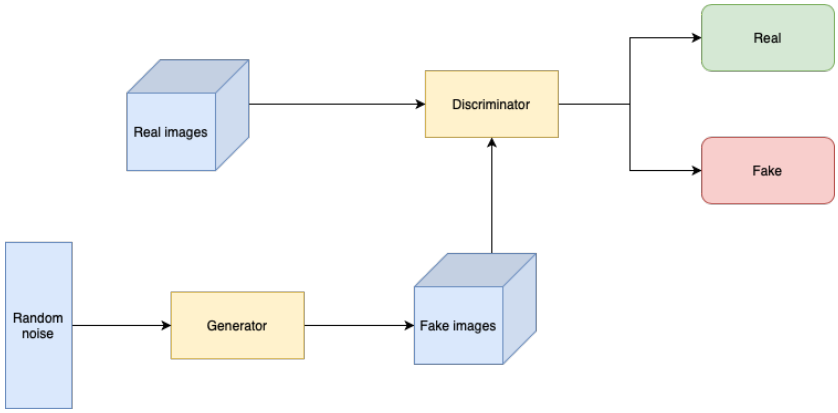

Figure 3.1: GAN framework

GANs have proven to be very successful at generating images, and here are some examples of generated faces by NVIDIA.

Figure 3.2: Generated images of faces from Karras et al. (2017)

3.2 Differential Privacy

Differential privacy, introduced in Dwork et al. (2006) and expanded upon in Dwork & Roth (2014) is a strong standard for privacy guarantees for algorithms that query databases. It works by adding noise to the data. More noise makes the data more anonymous but less useful. Adding a small amount of noise to the data makes it more useful, but does not protect the privacy of the data as well. This trade off between privacy and utility is controlled by the parameter epsilon, ϵ. Often referred to as the privacy budget, epsilon is used to determine how strict the privacy guarantees are for this specific application. Smaller epsilon values mean the privacy is protected well and an attacker is less likely to be able to identify an individual in the original dataset; bigger epsilon values provide weaker guarantees of the privacy protection.

The definition for differential privacy is given in terms of two adjacent databases, where adjacent databases are that which differ by a single entry. For this project the data will be a set of image and label pairs.

Definition 1. A randomized mechanism $\mathcal{M} : D \to R$ with domain $\mathcal{D}$ and range $\mathcal{R}$ satisfies (ϵ, δ)-differential privacy if for any two adjacent inputs $d, d' \in \mathcal{D}$ and for any subset of outputs $\mathcal{S} \subseteq \mathcal{R}$ it holds that

$$\Pr[\mathcal{M}(d) \in \mathcal{S}] \leq e^{\epsilon}\Pr[\mathcal{M}(d') \in \mathcal{S}] + \delta$$

3.3 Differentially Private Machine Learning

There are many ways of producing differentially private machine learning models such as the approach of Triastcyn & Faltings (2018). Specifically they propose the activations, $a(x)$, on the second to last layer of the discriminator become $\tilde{a}(x) = a(x)/max(||a(x)||_2, 1) + \mathcal{N}(0; \sigma^2)$ where $\mathcal{N}$ is Gaussian noise. Abadi et al. (2016) forms the basis of this project and proposes a different method, in which noise is added to the weights of the neural network. When the weights are calculated and updated in the optimiser of network, the Gaussian noise mechanism is applied to them. The amount of noise added to the weights can be controlled by the user and can be fine tuned to achieve different values of epsilon.

A neural network optimiser typically works as follows:

1. Sample a batch of training points (x, y) where x is the input data and y is the output data
2. Calculate the loss $L(\theta, x, y)$ between the model prediction $f_\theta(x)$ and y where θ is the model parameters
3. Calculate the gradient of the loss $L(\theta, x, y)$ with respect to the model parameters
4. Multiply the calculated gradients by the learning rate, and update the model parameters

The differentially private optimiser works slightly differently:

1. Sample a batch of training points (x, y) where x is the input data and y is the output data
2. Calculate the loss $L(\theta, x, y)$ between the model prediction $f_\theta(x)$ and y where θ is the model parameters
3. Calculate the gradient of the loss $L(\theta, x, y)$ with respect to the model parameters
4. Clip the gradients, per training example included in the batch, to ensure each gradient has a known maximum L^2 norm
5. Add Gaussian noise to the clipped gradients
6. Multiply the clipped and noised gradients by the learning rate, and update the model parameters

Abadi et al. (2016) also provides a method of calculating the epsilon that will be achieved, given the number of times the dataset will be accessed and the amount of noise that will be added to the weights. If the dataset is accessed only a few times, i.e. a small number of epochs are used in training, and the noise value added to weights is high then a very small value of epsilon will be achieved. The privacy of the dataset will be well protected, and an attacker would be less likely to reconstruct examples from the original dataset. In the paper, Abadi et al. create a differentially private classifier on the MNIST handwritten digit dataset (LeCun et al. 2010) and achieve an accuracy of 90%, 95%, and 97% for epsilon values of 0.5, 2, and 8 respectively.

3.4 Fréchet Inception Distance

This is a metric for measuring the utility of a generated dataset from a GAN. It is developed as an extension of the inception score (Salimans et al. 2016) which proposed applying the Inception-v3 network (Szegedy et al. 2015) to generated examples and then comparing the conditional label distribution with the marginal label distribution. The score represents the quality of the generated images and their diversity. A higher score is better. The IS is calculated:

$$\text{IS} = \exp(\mathbb{E}_{x \sim p_g} D_{KL}(p(y|x)||p(y)))$$

Where x is an image, y is the class label and $p(y|x) \in [0, 1]^{1000}$ is the probability that the network assigns to the image x being class y. $x \sim p_g$ indicates that x is an image sampled from the generator p_g. $D_{KL}(p||q)$ is the KL-divergence between the distributions p and q.

The FID (Heusel et al. 2017) is an improvement upon the IS by comparing the generated samples to real samples, instead of calculating the score on generated samples alone. It is also calculated using the Inception-v3 model, however the output layer is removed and the output is taken as the activations from the last pooling layer in the network. This layer has 2,048 activations and so each image input to the network is predicted as a 2,048 feature vector. The FID score is then calculated:

$$\text{FID} = ||\mu_1 - \mu_2||^2 + Tr(C_1 + C_2 - 2(C_1 * C_2)^{1/2})$$

Where μ_1 and μ_2 are the feature-wise mean of the activation feature vectors for real and generated images. C_1 and C_2 are the covariance matrices for the activation feature vectors of the real and generated images, and $Tr()$ is the trace linear operation where the elements along the main diagonal of the square matrix are summed. The FID corresponds to the similarity between the activation distributions of the real images and the generated images, a lower FID score is better as it represents more similar distributions.

4 Approach

Expanding on Abadi et al. (2016) I will be using their differentially private optimiser to create a differentially private generative adversarial network. To gain an understanding of how to use their work, I will first recreate the differentially private classifier that is implemented in the paper, on the MNIST handwritten digits dataset. I will experiment with varying the noise levels added to weights in the network, to achieve different values of epsilon and understand the effect of that on the accuracy of the classifier.

I will then implement the GAN that will be used for this project, using the deep convolutional generative adversarial network (DCGAN) framework in Radford et al. (2015). It is an extension to the original GAN framework (Goodfellow et al. 2014), which uses multilayer perceptron architectures for the discriminator and generator. DCGAN uses convolutional layers in the discriminator and generator. The dataset I will use for this project will be the MNIST handwritten digits dataset, as they are a widely used standard for testing GANs.

Having implemented the DCGAN, I will then use a differentially private optimiser in order to achieve differential privacy guarantees for the training dataset. The differentially private optimiser will only need to be implemented for the discriminator, as the generator will not receive any input from the dataset with privacy protection. The generator will use a normal optimiser without differential privacy guarantees. Any output by the generator will be differentially private due to the post-processing property of differential privacy (Dwork & Roth 2014), which states that if $\mathcal{M}$ is (ϵ, δ)-DP then $\mathcal{F} \circ \mathcal{M}$ is also (ϵ, δ)-DP. The framework for the differentially private GAN will be as such:

$$\begin{gathered}\text{differentially private discriminator + generator} \\ \rightarrow \text{differentially private generator}\end{gathered}$$

The output from the generator will have the same privacy guarantees with respect to the original dataset as the discriminator.

I will then train multiple models using this framework, where each model has a different value of epsilon. To create each model with a different value of

epsilon I will vary the amount of noise added to the weights of the discriminator. The generated images from each model should be of varying qualities depending on how tight their privacy guarantees are. I expect models with low epsilon values and strong privacy guarantees for the input dataset to be of a relatively low quality, and models with large epsilon values to be of a better quality. The trade off between privacy of the training dataset and utility of the generated dataset will be the subject of this project.

We have established a method of measuring the privacy protection of the original dataset, with the value epsilon. I will need to implement a method of measuring the utility of each dataset of generated images. For this I will use a slightly modified version of the Fréchet Inception Distance (FID), where instead of using the Inception v3 classifier model I will create a smaller network and use the activations in the last layer of that to calculate the FID score. I will then generate samples at each level of epsilon from the models I have trained, and use this new network to calculate their FID score. Images from models with low epsilon values should produce high FID scores, and images from models with high epsilon values should produce low FID scores.

5 Implementation

For this project I used the Python programming language in a Jupyter notebook hosted on the Google Colab platform. Jupyter notebooks are an open source interactive web application that can be used to write code and rich text elements. They are useful for experimenting with code, and are used often in data science and machine learning projects. Google Colab is a website that hosts Jupyter notebooks and provides hardware to run the code. I have developed my project on Google Colab because they provide GPUs for free, which speed up the training of neural networks by many orders of magnitude.

The research in Abadi et al. (2016) has been implemented as an extension to the TensorFlow library, referred to as TensorFlow Privacy and found at Andrew et al. (2020). IBM have also implemented a Python library for differential privacy in machine learning (IBM 2020). I chose to use the TensorFlow version as it provides better documentation, and in a few small experiments I found their code easier to work with.

Creating a differentially private classifier for the MNIST handwritten digits dataset is the first step in my approach for this project. For this I implemented a convolutional neural network with two convolutional layers followed by max pooling layers, a layer to flatten the image, a dense layer with 32 units and finally a dense layer with 10 units to output the probability that the input image corresponds to a specific number. I used the differentially private optimiser as implemented by the TensorFlow privacy library, and trained this network over 15 epochs with a learning rate of 0.25. I used a noise multiplier of 1.3 to add to the gradients in the differentially private optimiser. This network achieved an accuracy of 95% on a hold-out test dataset, with an epsilon value of 1.18 for the original training dataset. This shows that it is possible to achieve high utility and accuracy, while also maintaining a good level of privacy for the training dataset.

The next step was to create a GAN, for this I used a TensorFlow tutorial (TensorFlow 2020) on Deep Convolutional GAN (DCGAN) as a guide to set up my network.

I then changed the optimizer of the discriminator network from stochastic gradient descent to differentially private stochastic gradient descent. Having changed the optimiser, I received the error message 'an operation outside the function building code was being passed a graph tensor'.

```
TypeError: An op outside of the function building code is being passed
a "Graph" tensor. It is possible to have Graph tensors
leak out of the function building context by including a
tf.init_scope in your function building code.
For example, the following function will fail:
  @tf.function
  def has_init_scope():
    my_constant = tf.constant(1.)
    with tf.init_scope():
      added = my_constant * 2
The graph tensor has name: strided_slice:0
```

Figure 5.1: Error message

I spent a long time debugging my code and picking through the library to determine the cause of this issue to no avail. I then created an issue on the GitHub page of the library. Shortly after creating the issue, I realised the problem came from not passing the 'num-microbatches' argument to the

optimizer. I didn't think it was necessary because the optimizer calculates the microbatch size from the size of the loss vector regardless of whether the argument is given. This code seems to be faulty and I have amended the issue on GitHub to provide more information for the TensorFlow team, but have not yet had a response.

Once the optimiser was working correctly, this caused the time per epoch to increase from roughly 20 seconds per epoch to 200 seconds per epoch. This is because the gradients are clipped on each training point individually, to limit how much one training point can impact the model parameters. It also created noisy images, as shown below in figure 5.2.

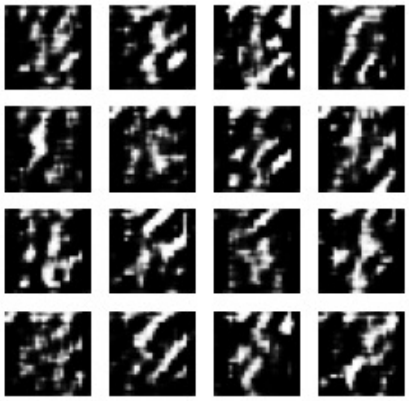

Figure 5.2: Noisy image

The implementation that I initially used to calculate the gradients for the discriminator used a method in TensorFlow to calculate the gradients which is very computationally expensive. I saw this as a potential route to speeding up the time per epoch, and then implemented it without using the computationally expensive method. This resulted in much quicker processing times, however it gave much more noisy images than before such as figure 5.3. The images were also not improving throughout training.

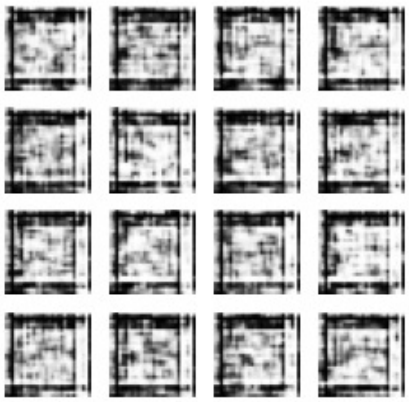

Figure 5.3: Very noisy image

Varying the noise levels showed no change in the resulting images, and it was clear the new method that I had implemented to calculate gradients was not working. I then reverted back to the old method which showed a variety of results when changing the noise levels, though all of the images were still far from acceptable, and the time per epoch again increased.

I realised the optimiser code as implemented by the TensorFlow team needed updating. Some of the gradients it was calculating were from fake images, which did not need noise adding to them. The fake images, produced by the generator did not need noise added to the gradients calculated from them, because they did not come from the training dataset which has the differential privacy guarantees. I modified the optimiser code to ensure no noise was added to the gradients calculated from generated images. This helped greatly, but the hyperparameters and network architectures still needed tuning.

To reduce the time it took to train the models, I implemented two dense layers in the discriminator network rather than convolutional layers. This resulted in images similar to figure 5.2, with a significant reduction in processing time. Experimenting with larger dense networks produced slightly better images while still keeping the time per epoch low. Despite the time improvements, the results were still very noisy, and it was clear that the

discriminator was still weak. I ruled out doubts about the generator, as the architecture I was using for that was still similar to the DCGAN I used as a starting point. To try to strengthen the dense discriminator network, I increased the number of times that it was trained per the number of times the generator was trained. Initially the discriminator was trained, then the generator was trained, at a 1:1 ratio. I then increased this ratio to 5:1. This produced better images where some numbers were legible, confirming my suspicion that the discriminator was previously too weak. Figure 5.4 below shows some images from this stage, a big improvement over figure 5.2.

Figure 5.4: Images from GAN with dense layer discriminator

However, the resulting images were still far from acceptable. At this point I wrote a function that would, at the end of every epoch, create a graph which tracked the loss values of the generator, the discriminator for real images, and the discriminator on fake images. It was a great insight when varying the hyperparameters of my network, and showed that my loss values were often increasing endlessly with every epoch.

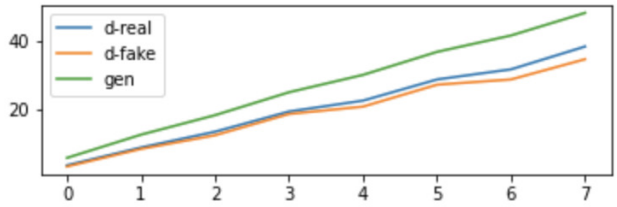

Figure 5.5: Rising loss values

At this stage I decided to re-use a convolutional architecture for my discriminator, with a much small number of parameters than I had previously. This was very promising, and I started to produce reasonably acceptable images. After tuning the hyperparamters again I settled on the architecture for my network. The generator is composed of two dense layers with RELU activations and batch normalisation, followed by a layer to reshape the data and an upsampling layer, then a convolutional layer with batch normalisation and a RELU activation, then another upsampling layer followed by a final convolutional layer with tanh activation. The discriminator is composed of two convolutional layers followed by max pooling layers, then a layer to flatten the image, followed by a dense layer with 32 units and finally a dense layer with 1 unit. The hyperparameters I chose for training were a batch size of 120, 30 epochs for training, a 1:1 ratio of training of the discriminator to the generator, and a learning rate that starts at 0.15 but decreases over time to 0.052.

Having settled on this model, I then started varying the noise levels to create multiple networks, each with different epsilon values. I saved the weights of each of these networks, to be used in the evaluation stage.

6 Results and Evaluation

To evaluate the images that are produced at different privacy budgets is a problem that can be tackled in many ways. In this section I will propose three methods of evaluating the results and compare these evaluation techniques. It is very difficult to objectively evaluate GANs, and there is no widely agreed upon metric to use. The first approach I will take is a qualitative measure, comparing images manually, and then I will implement two quantitative measures. I expect models with low epsilon values to produce low quality images, as they will be preserving the privacy of the original images better. High epsilon value models are expected to yield higher quality images, although they will incur significant privacy loss for the original dataset.

Visual inspection of the generated images is a rudimentary method of measuring the quality of generated images. However, it is quick to implement and is an intuitive way of showing that one generator is better than another if there is a stark difference between the generated images. In figure 6.1 below, the differences in quality of generated images clearly increases with higher epsilon values. Images with low epsilon values have a higher amount of noise added to the gradients when training the discriminator, and this results in noisier images produced by the generator.

$\epsilon = 1$ $\epsilon = 4.6$ $\epsilon = 9.0$ $\epsilon = 99.1$

Figure 6.1: Image quality over a range of epsilon values

When training images with high epsilon values (9+), the images generated at each epoch of training show a clear increase in quality as shown in figure 6.2 below.

Epoch 5 Epoch 10 Epoch 20 Epoch 30

Figure 6.2: Image quality increasing over time with $\epsilon = 9$

This is not the case however with lower epsilon values. I discovered that when training the low epsilon networks the quality of the images would increase in the first few epochs, then either plateau or decrease in quality for the rest of the training. Below are some examples of a network with an epsilon of 1.5 at epochs 5, 10, 20 and 30.

Epoch 5 Epoch 10 Epoch 20 Epoch 30

Figure 6.3: Image quality decreasing over time with $\epsilon = 1.5$

Clearly the quality of the images is much better at epochs 5 and 10 than it is at epochs 20 and 30, which is a very counter-intuitive result. In my time on this project I was not able to fix this problem, though a simple fix may be to stop the training a lot earlier for low epsilon value models.

There are two problems with manually inspecting images produced by a GAN, first being that the rating is subjective. Two different people may give very different opinions of the same image. Secondly, it is very difficult to score the generated images at a large scale. If a very small set of generated images are rated for their quality, you may inadvertently be cherry-picking

good images. To overcome these problems I explored two quantitative measures of GAN performance.

To implement both evaluative measures, I created a second neural network to classify images as real or fake, I will refer to this as the secondary network. The secondary network is separate from the GAN, and is not trained using a differentially private optimiser. It is a single layer dense network with 10 neurons. The training set is 60,000 images from the original MNIST dataset and 60,000 images generated from the differentially private GAN, each respectively having the output label of 1, representing real images, or 0, representing generated images. This network is trained over 10 epochs to output '1' if it thinks the image is real or '0' if the image is fake. I then generate another 10,000 images and combine them with another 10,000 images from the MNIST dataset to create a set of images to test the network with images that it has not seen during training.

The first quantitative measure is the accuracy of the secondary network on the test images. Lower accuracy in the secondary network would show that it is less sure whether an image is fake or not and therefore the generated images are similar to the original. With a generated dataset that perfectly mimics the original, the secondary network would be guessing whether an image is generated or not and the accuracy would approach 50%. Below is a graph showing the relationship between the epsilon value used to train the GAN and the accuracy of the secondary network on the test set. It shows a clear correlation between privacy loss of the original dataset and the utility of the generated dataset, where low epsilon values result in a higher accuracy for the secondary network and high epsilon values result in slightly lower accuracy in the secondary network. This is as expected, because the high epsilon value images mimic the original dataset much better than the lower epsilon value images do.

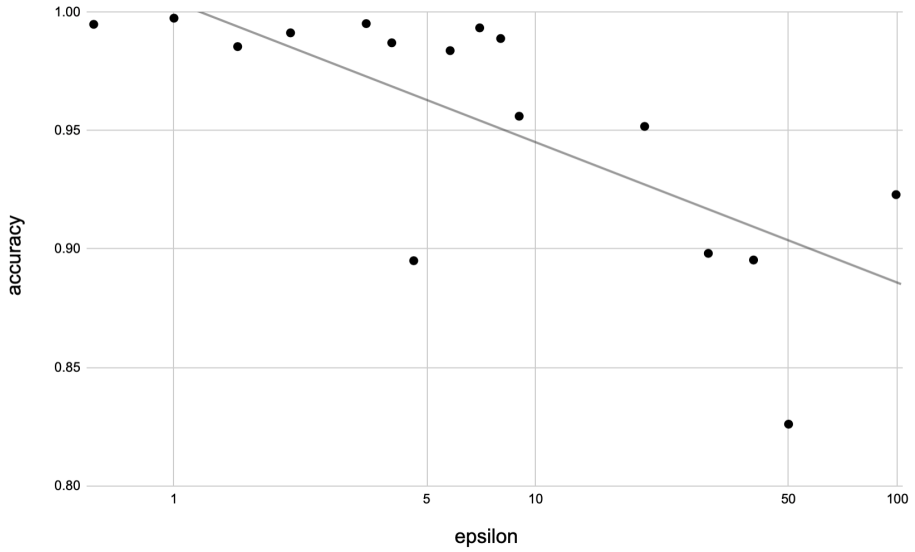

Figure 6.4: The relationship between accuracy of the secondary network and the epsilon of the generated images

The second measure is a slightly modified version of the Fréchet Inception Distance (FID). I have modified it so that instead of using the Inception-v3 model to give the activation feature vectors I will be using the activation layer from the secondary network used above. Figure 6.5 shows a plot of the the FID score versus the epsilon used when training the GAN. Like the first metric, it shows a clear trade off between the privacy loss of the original dataset and the utility of the generated dataset.

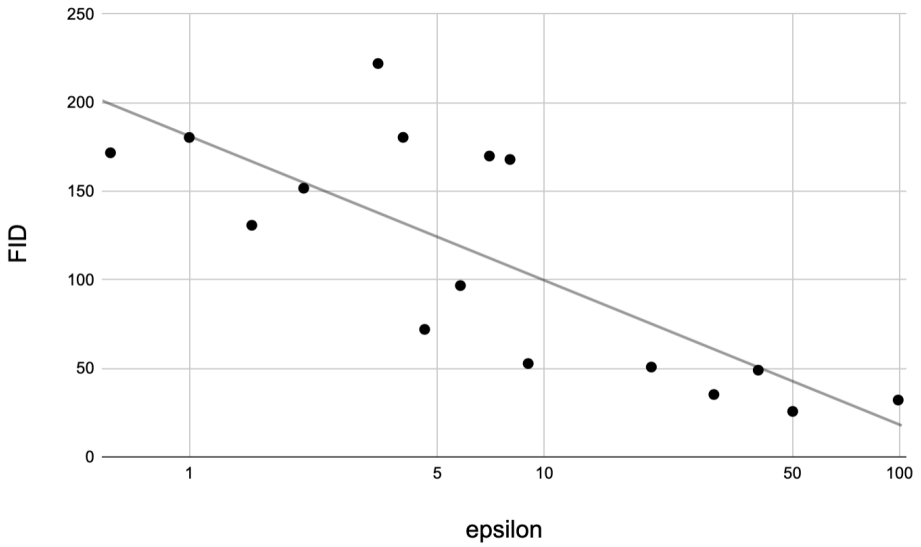

Figure 6.5: The relationship between FID and the epsilon of the generated images

By comparing figures 6.4 and 6.5, the FID score demonstrates the relationship between privacy and utility much more clearly. Using the FID score gives a much wider range of values, and seems to be more stable in the values it produces.

7 Future Work

The future work for this project can be split into two groups, improvement and expansion. There is room for improvement in the performance of the differentially private GAN, and tuning the implementation could lead to higher quality images being produced at very modest epsilon levels. The choice of loss function, architecture of the generator, architecture of the discriminator, and the hyperparameters could all be changed to achieve an improved result. There is also a promising set of use cases that this project could be expanded to. There are a large number of datasets with statistical insight that cannot be released due to privacy concerns.

The best loss function to be used when training a GAN is widely debated,

and it can vary for the specific data set which is trying to be generated. A loss function that is very promising for image data is the Wasserstein loss function proposed in Frogner et al. (2015). This loss function is based on the Wasserstein distance metric which is a measure of the distance between two probability distributions, for GANs this would be the probability distribution of the generated dataset and the training dataset. It is shown in Arjovsky et al. (2017) that when using the Wasserstein loss function, the training of GANs is less sensitive to hyperparameter changes and is more stable than previous loss functions such as binary cross entropy. An improvement could also be made in the choice of hyperparameters by making use of the techniques in Bergstra et al. (2013).

Expansion of this framework to other datasets would be the next step of work I would take for this project. The ability to generate synthetic datasets with differential privacy guarantees has a wide number of applications, as almost every data curator has data with statistical insight that cannot be released due to privacy concerns. This project shows it is possible to generate datasets that would be useful to other businesses, organisations or individuals. Minor changes would need to be made to the implementation to handle different datasets and data types such as tabular data or time series data, and the utility function would need to be tailored to each dataset. Choi et al. (2017) discusses healthcare organisations generating synthetic versions of electronic health records, in order to provide insight to medical researchers. This application could lead to great advances in biomedical research and patient care techniques.

8 Conclusion

This project set out to create a method of producing a differentially private synthetic dataset, given an original. It also set out to measure the privacy protection of the original dataset, and measure the utility of the data produced. Finally, this project aimed to study the relationship between utility and privacy loss when producing a synthetic dataset.

Having met all of these aims, the project was a success. This work has shown it is possible to create a high quality synthetic dataset while also giving reasonable bounds on the privacy loss of the original dataset. The results show

a clear relationship between privacy loss and utility, and the project has implemented a method of producing synthetic datasets at any value of epsilon.

There is still room for improvement, and there are a number of ways this project can be expanded from here. The results shown so far are very promising of the potential of this framework.

9 Reflection

My research skills have increased significantly while undertaking this project. For example comprehending the latest research in the field, and being able to apply and distil this knowledge into a workable project. I believe my writing skills have also improved, in being able to express my point of view and explain a problem and solution clearly.

While I deem this project a success, there are many points where, looking back, I could've worked significantly better. The first improvement I would make is to write down everything that I have worked on each day, noting what worked and what didn't. This would prevent me from repeating mistakes that I had already learned from. One example of this mistake is when I was tuning the hyperparameters and the architecture of the GAN in this project. I spent a few weeks trying to choose the correct layers for the discriminator, and the parameters to go with those choices, and I eventually settled on very similar choices to the ones I started with. I think this could've been avoided if I had made a better record of my work throughout this project.

I significantly developed my technical skills in this project, understanding the intricacies of implementing neural networks, and generative adversarial networks. I will take what I have learned, such as adapting the learning rate throughout the course of training, into future projects.

Differential privacy and the ways of measuring privacy loss were also topics that I learned significant amounts about during this project. Before I started this project I was not aware that there are ways to quantify privacy loss. The research that I have undertaken during this project has greatly improved my understanding of privacy, and furthered my mathematics skills by compre-

hending how many of the mechanisms that I have implemented work.

I feel that this project has great potential for future applications. I am very pleased with the results and achievements of this project, and I am proud of what I have learned along the way.

References

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K. & Zhang, L. (2016), 'Deep learning with differential privacy', *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16* .
URL: *http://dx.doi.org/10.1145/2976749.2978318*

Andrew, G., Chien, S. & Papernot, N. (2020), 'Tensorflow privacy', `https://github.com/tensorflow/privacy`.

Arjovsky, M., Chintala, S. & Bottou, L. (2017), 'Wasserstein gan'.

Bergstra, J., Yamins, D. & Cox, D. D. (2013), Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, *in* 'Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28', ICML'13, JMLR.org, p. I–115–I–123.

Choi, E., Biswal, S., Malin, B., Duke, J., Stewart, W. F. & Sun, J. (2017), 'Generating multi-label discrete patient records using generative adversarial networks'.

Dwork, C., McSherry, F., Nissim, K. & Smith, A. (2006), Calibrating noise to sensitivity in private data analysis, *in* S. Halevi & T. Rabin, eds, 'Theory of Cryptography', Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 265–284.

Dwork, C. & Roth, A. (2014), 'The algorithmic foundations of differential privacy', *Foundations and Trends® in Theoretical Computer Science* **9**(3–4), 211–407.
URL: *http://dx.doi.org/10.1561/0400000042*

Fredrikson, M., Jha, S. & Ristenpart, T. (2015), Model inversion attacks that exploit confidence information and basic countermeasures, *in* 'Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security', CCS '15, Association for Computing Machinery, New York, NY, USA, p. 1322–1333.
URL: *https://doi.org/10.1145/2810103.2813677*

Frogner, C., Zhang, C., Mobahi, H., Araya-Polo, M. & Poggio, T. A. (2015), 'Learning with a wasserstein loss', *CoRR* **abs/1506.05439**.
URL: *http://arxiv.org/abs/1506.05439*

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014), Generative adversarial nets, *in* Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence & K. Q. Weinberger, eds, 'Advances in Neural Information Processing Systems 27', Curran Associates, Inc., pp. 2672–2680.
URL: *http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf*

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G. & Hochreiter, S. (2017), 'Gans trained by a two time-scale update rule converge to a nash equilibrium', *CoRR* **abs/1706.08500**.
URL: *http://arxiv.org/abs/1706.08500*

Hitaj, B., Ateniese, G. & Pérez-Cruz, F. (2017), 'Deep models under the GAN: information leakage from collaborative deep learning', *CoRR* **abs/1702.07464**.
URL: *http://arxiv.org/abs/1702.07464*

IBM (2020), 'IBM Differential Privacy Library', `https://github.com/IBM/differential-privacy-library`.

Karras, T., Aila, T., Laine, S. & Lehtinen, J. (2017), 'Progressive growing of gans for improved quality, stability, and variation', *CoRR* **abs/1710.10196**.
URL: *http://arxiv.org/abs/1710.10196*

LeCun, Y., Cortes, C. & Burges, C. (2010), 'Mnist handwritten digit database', *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* **2**.

Li, N., Li, T. & Venkatasubramanian, S. (2007), 't-closeness: Privacy beyond k-anonymity and l-diversity', *2007 IEEE 23rd International Conference on Data Engineering* .

Machanavajjhala, A., Kifer, D., Gehrke, J. & Venkitasubramaniam, M. (2007), 'L-diversity: Privacy beyond k-anonymity', *ACM Trans. Knowl. Discov. Data* **1**(1), 3–es.
URL: *https://doi.org/10.1145/1217299.1217302*

Radford, A., Metz, L. & Chintala, S. (2015), 'Unsupervised representation learning with deep convolutional generative adversarial networks', *CoRR* **abs/1511.06434**.

Rocher, L., Hendrickx, J. & Montjoye, Y.-A. (2019), 'Estimating the success of re-identifications in incomplete datasets using generative models', *Nature Communications* **10**.

Salimans, T., Goodfellow, I. J., Zaremba, W., Cheung, V., Radford, A. & Chen, X. (2016), 'Improved techniques for training gans', *CoRR* **abs/1606.03498**.
URL: *http://arxiv.org/abs/1606.03498*

Sweeney, L. (2002), 'k-anonymity: A model for protecting privacy', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* **10**(05), 557–570.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2015), 'Rethinking the inception architecture for computer vision', *CoRR* **abs/1512.00567**.
URL: *http://arxiv.org/abs/1512.00567*

TensorFlow (2020), 'Deep Convolutional Generative Adversarial Network', `https://www.tensorflow.org/tutorials/generative/dcgan`.

Triastcyn, A. & Faltings, B. (2018), 'Generating differentially private datasets using gans', *CoRR* **abs/1803.03148**.
URL: *http://arxiv.org/abs/1803.03148*