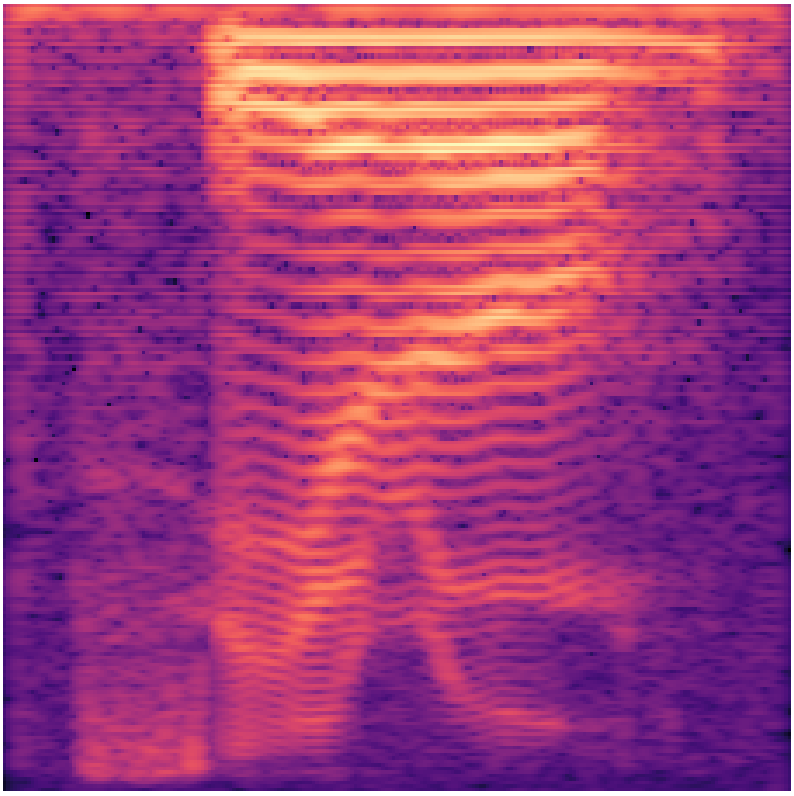


Adversarial Representation Learning For Private Speech Generation

Master's thesis in Engineering Mathematics and Computational Science

ADAM ÖSTBERG
DAVID ERICSSON

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:NN

Adversarial Representation Learning For Private Speech Generation

Adam Östberg
David Ericsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2020

Adversarial Representation Learning For Private Speech Generation
Adam Östberg
David Ericsson

Supervisor: Edvin Listo Zec, RISE Sweden
Examiner: Petter Mostad, Department of Applied Mathematics and Statistics

Master's Thesis 2020:NN
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Adversarial Representation Learning For Private Speech Generationt
Adam Östberg
David Ericsson
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

As more data is collected in various settings across organizations, companies, and countries, there has been an increase in the demand of user privacy. Developing privacy preserving methods for data analytics is thus an important area of research. In this work we present a model based on generative adversarial networks (GANs) that learns to obfuscate specific sensitive attributes in speech data. We train a model that learns to hide sensitive information in the data, while preserving the meaning in the utterance. The model is trained in two steps: first to filter sensitive information in the spectrogram domain, and then to generate new and private information independent of the filtered one. The model is based on a CNN that takes mel-spectrograms as input. A MelGAN is used to invert the spectrograms back to raw audio waveforms. We show that it is possible to hide sensitive information such as gender by generating new data, trained adversarially to maintain utility and realism.

Keywords: generative adversarial networks, adversarial representation learning, deep learning, privacy, speech generation

Acknowledgements

Looking back at the spring that passed, many things did not turn out as expected. Half the project had to be conducted from home, which was an additional challenge. We would like to express the deepest of gratitude to our supervisor Edvin Listo Zec for providing excellent guidance and support given the circumstances. He always pushed us to explore our own ideas, but also steered us in the right direction whenever we needed it. We would also like to thank John Martinsson and Olof Mogren for the fruitful discussions, the weekly seminars, and for letting us be a part of the research group at RISE. Finally, we would like to thank Petter Mostad for taking on the role as examiner on such short notice.

We are also thankful for all the good times and memories we have experienced during the project and the past two years with our co-students and friends Oskar Eklund, Kasper Bågmark, Simon Johansson, Emma Darebro, Tomas Forssmark, Simon Lundin and Hampus Larsson to name a few.

Finally, we would like to thank our partners Laura Ruiz Idänmaa and Lovisa Högberg for the constant support throughout the spring as well as putting up with us in our small apartments.

Adam Östberg & David Ericsson, Gothenburg August 2020

Contents

List of Figures

List of Tables

1
Introduction

With greater availability of computing power and large datasets, machine learning methods are increasingly being used to gain insights and make decisions based on data. While providing great utility, the methods may extract sensitive information which the provider of the data did not intend to disclose. An example of this is a digital voice assistant. The user provides it with a command by speaking, and the speech is recorded through the microphone. A speech processing algorithm infers the spoken contents and executes the command accordingly. However, it has been shown that such state-of-the-art algorithms may infer other attributes as well such as gender, emotional state and even identity [33]. This raises the question of how we can provide representations of data to such applications, which are useful for the intended use but still respects the privacy concerns of people.

One way of approaching this question is through a utility vs. privacy trade-off. There is a line of recent work in the image domain [2, 8, 16, 25] with this set up which lies in the intersection between deep learning and information theory. In this setting one wants to learn a utility preserving representation R, from some input data X, such that some protected sensitive attributes S, are obfuscated when the representation is disclosed. This is typically as an optimization problem of the form

$$\max_{p_\theta(X)} U(X,R) \quad \text{s.t.} \;\; D(R,S) \leq \epsilon \tag{1.1}$$

or

$$\min_{p_\theta(X)} D(R,S) \quad \text{s.t.} \;\; U(X,R) \geq \epsilon \tag{1.2}$$

where U and D are some measure of utility and privacy, respectively, such as mutual information. The objective is to find a stochastic mapping $p_\theta(X)$ which sends X to R, parameterized by θ, such that these criteria are fulfilled for some fixed level of utility or privacy ϵ. It is implemented as a game between deep neural networks. One network is trained to map data X to a representation R, in which the sensitive attribute S is obfuscated. Another network is trained to predict the sensitive attribute. The networks are pitted against each other in an adversarial fashion.
Huang et al. [16] use this framework to train a *filter* network for censoring sensitive attributes in images. The idea is extended in [25], where in addition to censoring a sensitive attribute, it is proposed to generate a synthetic version in its place using a *generator* network. In both works, the mapping is restricted to be domain-preserving and the data produced by this method can be used regardless of downstream task.

In this thesis we investigate if models based on a utility-privacy trade-off approach are suitable for audio. We review and build upon existing models to see if they can be extended to audio data. In particular, we study if the framework with generative adversarial networks (GANs) presented in [25] can be extended to the audio domain. The goal is to be able to transform a speech recording such that some specific sensitive information cannot be inferred.

1.1 Specification

The aim is to develop a model that given a speech recording, can generate a new recording with the same utility, but which is private with respect to some sensitive information. A recording with the same utility is one which has the same spoken content. A recording is private with respect to a sensitive attribute, if it is impossible to infer it. In this project, we consider the sensitive information to be the gender. We will mainly focus on extending the model in [25] from the image domain. A difference compared to images is that raw audio has a very high temporal resolution. Raw audio data usually has at least 16000 samples per second, hence modelling a mere single second of audio corresponds to an image with dimensions 128×128. In addition to that, there are short and long-term dependencies appearing at different timescales [21]. Taking these aspects into account, we consider speech recordings containing a single utterance.

1.2 Related work

Since Goodfellow et al. [11] introduced the adversarial framework, it has been applied to various types of problems, including problems regarding privacy. Huang et al. [16] developed a generative adversarial privacy (GAP) framework. The authors optimize a constrained minimax game between a filter network that censors the sensitive data subject to a utility constraint and an adversary network that classifies the sensitive information. Martinsson et al. [25] utilize this framework, but in addition to filtering the sensitive data, the authors propose to replace it with new synthetic data generated by an additional GAN. The results show that adding a synthetic instance of a sensitive attribute helps in fooling the adversary, thus improving the privacy.

While adversarial models have shown to be a successful approach for generating and transforming images [35, 18], the same success has not been achieved in the audio domain. One of the first attempts to model raw audio signals using the adversarial setup was WaveGAN [7]. WaveGAN was able to generate intelligible words when trained on single-word speech recordings, as well as synthesizing audio from other domains such as bird vocalizations, drums and piano. The raw audio-based model is compared with a corresponding model that generates spectrograms, called SpecGan. However, these spectrograms are not invertible and require approximations to reconstruct the phase information of the audio. Engel et al. [9] address this problem

by working on a spectral representation which is invertible. The phase information is accounted for by also modelling the instantaneous frequencies which aims to capture to what extent the phase of a signal changes between frames.

GANs have also shown promising results within the field of text-to-speech, for instance GAN-TTS [3] and MelGAN [21]. The former one is a conditional GAN, conditioned on linguistic features and pitch information to generate high fidelity speech. The latter, MelGAN, is a fully convolutional model generating raw audio via mel-spectrogram inversion. It is worth noting that both models use multiple discriminators operating at different scales and at random patches of what the generator produced. A model that also operates on mel-spectrograms, although in the field of voice conversion, is MelGAN-VC [29]. This model is trained to convert signals from a source voice to a target voice. The conversion is carried out on mel-spectrograms and then inverted to audio using the Griffin Lim algorithm. Additionally, its design allows conversion of non-fixed length audio signals and produces realistic and intelligible results on both on intra-gender and inter-gender translations.

1.3 Outline

The remainder of the thesis is structured as follows. In Chapter 2, we present the theory used in the thesis. It consists of three parts; first, we provide an overview of artificial neural networks. Then, we introduce the adversarial framework. Finally, we give a brief explanation of time-frequency analysis. Following the theory, in Chapter 3 we give a complete description of our model, and describe the data used as well as the experiments we have conducted. Finally we describe the evaluation methods. The obtained results are presented in Chapter 4, followed by a discussion and conclusion in Chapter 5.

2
Background

In this chapter we introduce the necessary theory to explain the model that is proposed in this thesis. The reader is not expected to be familiar with neural networks, and generative adversarial networks in particular, and we therefore provide a rather detailed exposition of the mathematical theory. We also give a brief overview of generative adversarial privacy and time-frequency analysis.

2.1 Artificial neural networks

An artificial neural network is a composition of $L \geq 1$ layers where each layer consists of a number of nodes, often called neurons. Let $k^{(\ell)} \in \mathbb{N}^+$ denote the number of nodes in layer ℓ, with $k^{(0)}$ being the size of the input data. Each layer $\ell \in \{1, \ldots, L\}$ constitutes a function $\mathcal{L}^{(\ell)} : \mathbb{R}^{k^{(\ell-1)}} \to \mathbb{R}^{k^{(\ell)}}$ on the form

$$\mathcal{L}^{(\ell)}(\boldsymbol{x}) = f^{(\ell)}(\boldsymbol{g}^{(\ell)}(\boldsymbol{x}; \boldsymbol{\theta}^{(\ell)})). \tag{2.1}$$

The function $\boldsymbol{g}^{(\ell)} : \mathbb{R}^{k^{(\ell-1)}} \to \mathbb{R}^{k^{(\ell)}}$ is parameterized by $\boldsymbol{\theta}^{(\ell)}$ and defines how the neurons in layer $\ell-1$ are combined to compute a new vector of neurons. The function $f^{(\ell)} : \mathbb{R}^{k^{(\ell+1)}} \to \mathbb{R}^{k^{(\ell+1)}}$ is then applied to this vector element-wise in order to get the neurons in layer ℓ. The network as a whole is the function $\mathcal{N}_{\boldsymbol{\theta}} : \mathbb{R}^{k^{(0)}} \to \mathbb{R}^{k^{(L)}}$ formed by the composition

$$\mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathcal{L}^{(L)} \circ \cdots \circ \mathcal{L}^{(1)}(\boldsymbol{x}) \tag{2.2}$$

parameterized by $\boldsymbol{\theta} = (\boldsymbol{\theta}^{(1)}, \ldots, \boldsymbol{\theta}^{(L)}) \in \boldsymbol{\Theta} \subseteq \mathbb{R}^D$, where $\boldsymbol{\Theta}$ is the set of feasible parameters. The networks are often visualized by a graph where the vertices correspond to the neurons, and the edges correspond to the combinations of the neurons described by the $\boldsymbol{g}^{(\ell)}$. An example of a network with three layers is shown in Figure 2.1.

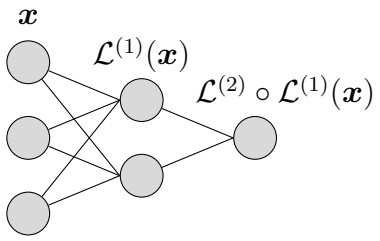


Figure 2.1: A neural network with two layers. The input consists of three neurons, the middle layer has two neurons and the output is a single neuron.

The number of layers L, the number of neurons in each layer $k^{(\ell)}$, as well as the activation and pre-activation functions $f^{(\ell)}$ and $\boldsymbol{g}^{(\ell)}$ are chosen when one sets up the structure of the network, and do not change during later stages. The parameters $\boldsymbol{\theta}$ may be adjusted in order to model a desired relationship between the input and the output. This process of tuning the parameters is called *training* the network, and is described in Section 2.1.2.1.

2.1.1 Building blocks of neural networks

In the following sections, the building blocks for artificial neural networks are described.

2.1.1.1 Fully connected layer

The most simple layer is a *fully connected layer*. As the name implies, there is a connection between every neuron in the input layer and output layer. In this case, the function $\boldsymbol{g}^{(\ell)}$ is on the form

$$\boldsymbol{g}^{(\ell)}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{2.3}$$

where $\boldsymbol{W} \in \mathbb{R}^{k^{(\ell)} \times k^{(\ell-1)}}$ and $\boldsymbol{b} \in \mathbb{R}^{k^{(\ell)}}$. The parameters are the weight matrix $\boldsymbol{W}$ and the bias vector $\boldsymbol{b}$, i.e. $\theta^{(\ell)} = (\boldsymbol{W}, \boldsymbol{b})$. Denoting the output $\boldsymbol{g}^{(\ell)}(\boldsymbol{x}) = \boldsymbol{y} = (y_1, \ldots, y_k)$, equation (2.3) can be written for each output i as

$$y_i = \sum_{j=1}^{k^{(\ell-1)}} W_{ij} x_j + b_i \tag{2.4}$$

An example of a fully connected layer is illustrated in Figure 2.2.

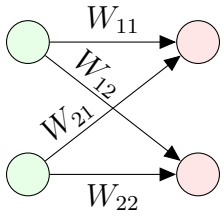


Figure 2.2: A fully connected layer with two input neurons and two output neurons.

2.1.1.2 Convolutional layer

A layer that is designed to take advantage of the spatial structure of the input is the *convolutional layer*. The idea is to utilize that nearby elements are more correlated than distant elements, and that similar features, e.g. lines or shapes in an image, may occur in different parts of the input [4].

As the name implies, a convolutional layer makes use of the mathematical operation of convolution. The discrete convolution between two functions $f, g : \mathbb{Z} \to \mathbb{R}$ at a

point $k \in \mathbb{Z}$ is defined as

$$(f * g)[k] = \sum_{n \in \mathbb{Z}} f[n]g[k-n] = \sum_{n \in \mathbb{Z}} f[k-n]g[n]. \tag{2.5}$$

The output can be interpreted as a g-weighted average of f. If we have a vector $\boldsymbol{x} \in \mathbb{R}^N$, we can see this as a function from the indices $D = \{1, \ldots, N\}$ to the real numbers, with $\boldsymbol{x}[n] = x_n$. We can also see the vector as a function $\boldsymbol{x} : \mathbb{Z} \to \mathbb{R}$ with $\boldsymbol{x}[n] = 0$ for $n \in \mathbb{Z} \setminus D$. Hence, for any two vectors $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{w} \in \mathbb{R}^M$, we can use the definition in (2.5) to define a discrete convolution for vectors

$$\boldsymbol{x} * \boldsymbol{w}[k] = \sum_{n \in \mathbb{Z}} \boldsymbol{x}[k-n]\boldsymbol{w}[n] \tag{2.6}$$

where the summation becomes finite due to the finite support of $\boldsymbol{x}$ and $\boldsymbol{w}$. This sum is hence well-defined and easily computed for any $k \in \mathbb{Z}$. In practice, the output of the convolution is restricted to a finite set of indices. If $N > M$, the index k is restricted to $k \in \{1, \ldots, N - M + 1\}$.

We may now proceed to define a convolutional layer. Let $\boldsymbol{x} \in \mathbb{R}^N$ be the input and let $\boldsymbol{w} = (w_1, \ldots, w_K) \in \mathbb{R}^K$ be a vector of weights. A layer is called *convolutional* when the function $\boldsymbol{g}^{(\ell)}$ is on the form

$$\boldsymbol{g}^{(\ell)}(\boldsymbol{x}) = \boldsymbol{x} * \boldsymbol{w}. \tag{2.7}$$

More explicitly, we have that the output is

$$\boldsymbol{g}^{(\ell)}(\boldsymbol{x})[k] = \boldsymbol{x} * \boldsymbol{w}[k] = \sum_{n=1}^{K} \boldsymbol{x}[k-n]\boldsymbol{w}[n]. \tag{2.8}$$

The parameters $\boldsymbol{\theta}$ for such a layer is the vector $\boldsymbol{w}$, which is called the *kernel*, and its length K is called the *kernel size*. The kernel size is also referred to as the *receptive field*. An example is illustrated in Figure 2.3.

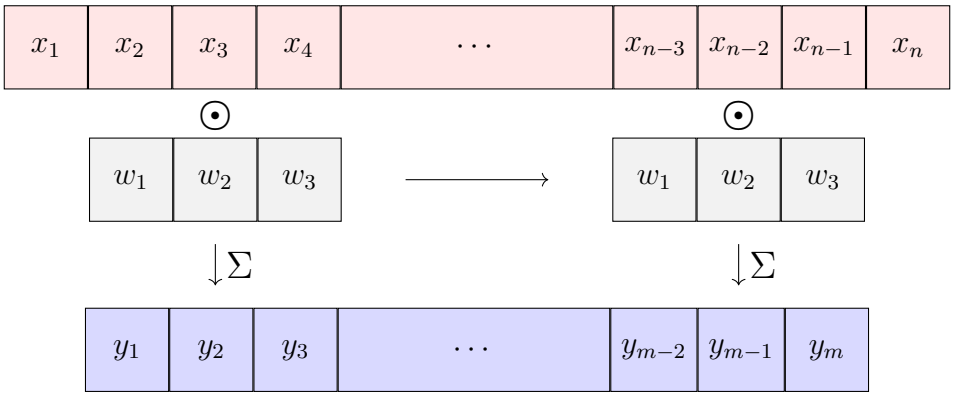


Figure 2.3: Illustration of the discrete convolution operation as a sliding window in one dimension. The input $\boldsymbol{x}$ is convolved with kernel $\boldsymbol{w}$, producing the output $\boldsymbol{y}$. The operation $\odot$ denotes the elementwise multiplication and the sum is taken over the elements in the resulting vector.

As can be seen, the convolution operation may be thought of sliding a window over the input. This sliding window is represented by the kernel $\boldsymbol{w}$. A result of this is that, in contrast to the fully connected layer, the convolutional layer can handle inputs of any size greater than the kernel size. Hence, a convolutional layer is not strictly speaking a function from $\mathbb{R}^{k^{(\ell-1)}}$ to $\mathbb{R}^{k^{(\ell)}}$. However, given a fixed input size N_{in}, the output size N_{out} of a convolutional layer with kernel size K is

$$N_{out} = \frac{N_{in} - K}{1} + 1 \tag{2.9}$$

and the layer can still be seen as a function from $\mathbb{R}^{N_{in}}$ to $\mathbb{R}^{N_{out}}$.

One can also let the sliding window move more than one step between each multiply-and-sum-operation. This can be achieved with a simple modification of (2.5):

$$\boldsymbol{x} * \boldsymbol{w}[k] = \sum_{n=1}^{K} \boldsymbol{x}[sk - n]\boldsymbol{w}[n]. \tag{2.10}$$

The number of steps s is called the *stride* and can be used to control the output size. Formula (2.9) then becomes

$$N_{out} = \frac{N_{in} - K}{s} + 1. \tag{2.11}$$

Another way to control the output is by using zero padding. In case one wants to preserve the dimensions of the input, then one can extend the input vector by adding zeros on both sides.

Sometimes, the input $\boldsymbol{x}$ will have multiple channels. An example with two channels would be a stereo recording. In general a single dimensional input with D channels would be a tensor in $\mathbb{R}^{N \times D}$. In order to feed such an input into a convolutional layer, we would like to perform the convolution operation on each channel. Since the different channels may highlight different aspects of the input, we do not necessarily want to use the same kernel $\boldsymbol{w}$ for each channel. Let $\boldsymbol{x}^i$ denote the ith channel of the input, with $\boldsymbol{x}^i = (x_1^i, \ldots, x_N^i)$ and let $\boldsymbol{w}^i$ denote a kernel with kernel size K. Each kernel $\boldsymbol{w}^i$ is convolved with the corresponding channel $\boldsymbol{x}^i$. The results of the convolutions are then added together to form the output, called a *feature map*.

The collection of kernel $\boldsymbol{w}^i$ can be stacked next to each other, as in Figure 2.4. For the multichannel case, performing the convolution operation can then be seen as sliding the box along the data dimension. This box is referred to as a *filter*. Hence, when we write $\boldsymbol{x} * \boldsymbol{w}$, we will mean convolving the filter box with the data box.

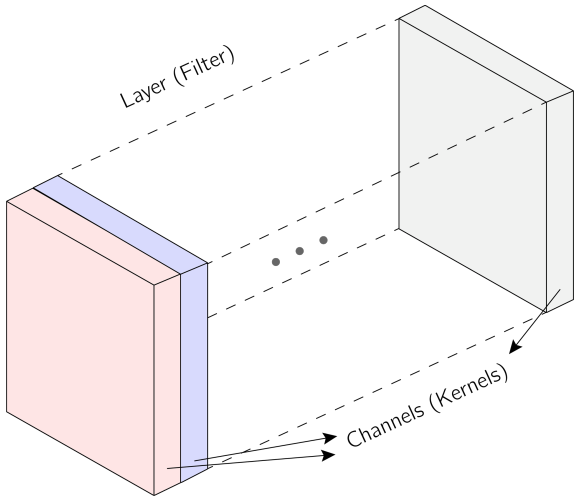


Figure 2.4: A collection of kernels $\boldsymbol{w}^i$ stacked next to each other to form a filter.

Finally, we want to be able to detect several different aspects in the input. Hence, a convolutional layer usually consists of multiple filters.

2.1.1.3 Activation functions

The function $\boldsymbol{g}$ in equation 2.3 is an affine function and in a neural network it is typically followed by a non-linear transformation, namely the activation function. The activation function adds a non-linear property to the network which enables it to model more complex relationships and patterns in the data [10]. A detailed description of the most commonly used activation functions, as well as their pros and cons, can be found in [28]. Most activation functions are defined elementwise, and three very common ones are defined below.

$$\text{ReLU:} \quad h(y) = \max\{0, y\} \tag{2.12}$$

$$\text{Sigmoid:} \quad \sigma(y) = \frac{1}{1 + e^{-y}} \tag{2.13}$$

$$\text{Hyperbolic tangent:} \tanh(y) = \frac{e^{2y} - 1}{e^{2y} + 1} \tag{2.14}$$

An activation function that differs from the ones above is the Softmax function, which is applied to the whole vector. For a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, the activation of the element x_j is calculated as

$$f(x_j) = \frac{e^{x_j}}{\sum_{i=1}^{n} e^{x_i}}. \tag{2.15}$$

This activation function produces values between 0 and 1, and normalizes the vector, thus it can be interpreted as a vector of probabilities. While the other listed activation functions are used in any layer of a neural network, the Softmax function is mostly used in the output layer.

2.1.1.4 Pooling

A convolutional layer is often followed by a *pooling* operation. In this operation, the output from the convolutional layer is replaced by a summary statistics of each neuron's neighbouring neurons. Two commonly used pooling operations are max pooling and the average pooling. In the former case, the neurons are replaced by the maximum neuron value within a rectangular neighbourhood of the neuron itself. In the latter case, the maximum is exchanged for an average. The pooling operation serves as an efficient way to reduce the spatial dimension, and thus reduce the number of parameters required in the subsequent layers. In addition to that, it introduces some local invariance to small translations of the input data [10]. Hence, if the input is translated by a small amount, most of the pooled output will not change.

2.1.2 Training a neural network

In this section we explain what it means to train a neural network and describe different aspects of the process. More specifically, we will work in the setting of *supervised learning*. In supervised learning, it is assumed one has access to data points $(\boldsymbol{x}, \boldsymbol{y})$ in a space $\mathcal{X} \times \mathcal{Y}$ which are sampled from a probability distribution $\boldsymbol{p}_{data}$ on $\mathcal{X} \times \mathcal{Y}$. The goal is to find a function $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ such that $\mathcal{F}(\boldsymbol{x}) = \boldsymbol{y}$. We can consider a neural network as a function $\mathcal{N}_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Z}$ parameterized by $\boldsymbol{\theta}$, where $\mathcal{Z}$ is some latent space, and then let $\mathcal{F} = \mathcal{P} \circ \mathcal{N}_{\boldsymbol{\theta}}$ where $\mathcal{P}$ is function that maps $\mathcal{Z}$ to $\mathcal{Y}$. The goal can then be reformulated as finding parameters $\boldsymbol{\theta}$ such that $\mathcal{P} \circ \mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{y}$. In the next section we will define a function $\mathcal{L}$ which measures how well the desired relationship is captured by the network, as a function of the parameters $\boldsymbol{\theta}$. Thus, finding the optimal parameters can be formulated as an optimization problem on the form

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}). \tag{2.16}$$

Training a neural network corresponds to approximately solving this optimization problem [10] in an iterative fashion that is explained in Section 2.1.2.2.

2.1.2.1 Loss function

In order to measure how well the network has captured the desired relationship between the input and the output, we will use a *loss function* ℓ. Recall that we have data $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$ from a distribution $\boldsymbol{p}_{data}$ and a neural network. The loss function $\ell : \mathcal{Z} \times \mathcal{Y} \to \mathbb{R}$ returns a real number indicating the quality of the mapping. Since we are interested in capturing the relationship for all points that follow $\boldsymbol{p}_{data}$, the function $\mathcal{L}$ will be defined as

$$\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{p}_{data}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim \boldsymbol{p}_{data}}[\ell(\mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})] \tag{2.17}$$

and will be referred to as the *expected loss*. In practice, we only have access to a dataset with a finite number of points $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^n \subseteq \mathcal{X} \times \mathcal{Y}$. Hence the expectation in

equation (2.17) cannot be computed. To get around this, we consider the empirical distribution

$$\boldsymbol{p}_{empirical}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} \delta_{(\boldsymbol{x}_i, \boldsymbol{y}_i)}(\boldsymbol{x}, \boldsymbol{y}) \tag{2.18}$$

and define the *empirical loss* as

$$\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{p}_{empirical}) = \frac{1}{n} \sum_{i=1}^{n} \ell(\mathcal{N}(\boldsymbol{x}_i), \boldsymbol{y}_i). \tag{2.19}$$

which can be computed from the dataset.

An example of a supervised learning problem is *classification*. The set $\mathcal{Y} = \{1, \ldots, K\}$ consists of classes, and each feature vector $\boldsymbol{x}$ belongs to exactly one class. In this case, $(\boldsymbol{x}, \boldsymbol{y}) \sim \boldsymbol{p}_{data}$ means that $\boldsymbol{x}$ belongs to the class $\boldsymbol{y} = k \in \mathcal{Y}$. The output space of the network is often set to $\mathcal{Z} = [0, 1]^K$ so that it can be interpreted as a probability distribution over $\mathcal{Y}$. The loss function is usually chosen as the cross-entropy loss, defined as

$$\ell(\mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}) = -\sum_{k=1}^{K} \mathbb{1}_{\{\boldsymbol{y}=k\}} \ln(\mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x})_k). \tag{2.20}$$

2.1.2.2 Optimization methods for training neural networks

With the loss function explained, we may now proceed to describe the different optimization techniques used for training neural networks. The most common way to solve the optimization problem (2.16) is by using *gradient descent* algorithms [31]. Such algorithms use the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ in order to update the value of $\boldsymbol{\theta}$. Hence, activation functions f, pre-activation functions $\boldsymbol{g}$, as well as the loss function must be differentiable.

The gradient of a function describes in which direction the rate of increase is the largest. By taking a step in the opposite direction, that is updating $\boldsymbol{\theta}$ by a factor of the negative of the gradient, the anticipation is to decrease the value of $\mathcal{L}(\boldsymbol{\theta})$. In other words, the update rule is

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}), \tag{2.21}$$

where $\alpha \in \mathbb{R}^+$ is the step size. The step size controls the distance we move in each iteration and is selected before the start of the algorithm. A too large step size may result in passing the optimal value, while a too small step size can lead to slow convergence. The step size is denoted η and referred to as the *learning rate*. In the context of training neural networks, there are three variants of gradient descent that are used. The difference between these variants is how much data they use in order to compute the gradient. When the full data set has been cycled through, an *epoch* has been completed.

The first variant is called *batch gradient descent*. In this case, the gradient of the loss function is computed for the entire dataset. The update rule (2.21) at epoch t

then becomes

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}(\boldsymbol{\theta}_t; \boldsymbol{p}_{empirical}). \tag{2.22}$$

A downside is that the variant can be very inefficient for large datasets; for each iteration we have to feed every sample through the network to compute the updated values of the parameters. Also, there may be multiple samples that are similar with similar gradients, resulting in redundant calculations.

At the other side of the spectrum, there is the second variant called *stochastic gradient descent*. This variant updates the parameters for each sample individually. Letting $(\boldsymbol{x}, \boldsymbol{y})$ denote a sample from the empirical distribution $\boldsymbol{p}_{empirical}$, the loss function for this sample is defined as

$$\mathcal{L}(\boldsymbol{\theta}; (\boldsymbol{x}, \boldsymbol{y})) = \ell(\mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}). \tag{2.23}$$

The parameters are updated according to

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}(\boldsymbol{\theta}_t; (\boldsymbol{x}, \boldsymbol{y})). \tag{2.24}$$

After the parameters have been updated for each sample, we let $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$.

Finally, there is the *mini-batch gradient descent* which is a combination of the first two methods. Here, the dataset $\mathcal{S}$ is partitioned into mini-batches $\mathcal{B}_1, \ldots, \mathcal{B}_m$ of equal size n, with $\mathcal{B}_j = \{(\boldsymbol{x}_{1,j}, \boldsymbol{y}_{1,j}), \ldots (\boldsymbol{x}_{n,j}, \boldsymbol{y}_{n,j})\}$ where $(\boldsymbol{x}_{i,j}, \boldsymbol{y}_{i,j})$ is the ith sample of the jth mini-batch. For each mini-batch $\mathcal{B}_j$, the gradient of the loss function is computed to perform the update of the parameters $\boldsymbol{\theta}$. In this case, the update rule reads

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} \mathcal{L}(\boldsymbol{\theta}_t; \mathcal{B}_j), \tag{2.25}$$

where $\mathcal{L}(\boldsymbol{\theta}; \mathcal{B}_j)$ denotes the loss over mini-batch, defined as

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{B}_j) = \frac{1}{n} \sum_{i=1}^{n} \ell(\mathcal{N}_{\boldsymbol{\theta}}(\boldsymbol{x}_{i,j}), \boldsymbol{y}_{i,j}). \tag{2.26}$$

Similarly to stochastic gradient descent, we let $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$ when the full data set has been cycled through.

While the methods above are simple and work quite well, a downside is that each component in the parameter vector $\boldsymbol{\theta}$ is updated using the same learning rate η. If the different components of $\boldsymbol{\theta}$ vary a lot in magnitude, then using the same learning may favour some components. One of the most popular optimization methods for training neural networks, which tackles this issue, is the *Adam* optimizer [19]. This optimization method uses *adaptive moment estimation*. For the gradient of the loss function at epoch t, we introduce the notation $\boldsymbol{l}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \cdot)$. We do not specify how much data is used to compute the gradient, the method is the same for all cases. The estimates of the first and second moments $\boldsymbol{m}_t$ (mean) and $\boldsymbol{v}_t$ (uncentered variance) at epoch t are given by

$$\begin{aligned} \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1) \boldsymbol{l}_t \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2) \boldsymbol{l}_t^2 \end{aligned}$$

where $\beta_1, \beta_2 \in [0, 1)$ are exponential decay rates for the respective estimates. Both $\boldsymbol{m}_0$ and $\boldsymbol{v}_0$ are initialized to $\mathbf{0}$. A result of this initialization is that in the beginning of the training and when the decay rates are small, i.e. β_i is close to 1, the moment estimates are biased towards zero. To correct this initialization bias, bias-corrected estimates of the moments are used instead [19]. The bias-corrected estimates are calculated as

$$\widehat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t}$$
$$\widehat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t}$$

and the *Adam* update rule for the parameters $\boldsymbol{\theta}$ is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\widehat{\boldsymbol{m}}_t}{\widehat{\boldsymbol{v}}_t + \boldsymbol{\epsilon}} \tag{2.27}$$

where the division is done elementwise. Here $\boldsymbol{\epsilon}$ is a constant vector which is added for numerical stability.

2.1.3 Regularization techniques

When training a neural network, one tries to approximately optimize a loss function on a finite set of samples, called the *training set*, with the hope that this training set is representative of the whole population. A common problem that arises when training neural networks is overfitting on this set. Overfitting means that the network adjusts its parameters to fit the training data too well and fail to perform on data it has not seen. The network's ability to react to new data is referred to as *generalization*. To measure how well a network is able to generalize, another finite set of samples called the *validation set* is used. There are several methods to address the problem of overfitting, referred to as regularization techniques.

2.1.3.1 Dropout

Dropout is a regularization technique in which, during training, the neurons are active based on a probability [34]. More technically, when dropout is applied to a layer, in each training iteration its neurons are either dropped out (put to zero) with probability $1 - p$ or kept with probability p. This may be seen as sampling a thinned network which only consists of the nodes that survived the dropout. Thus, in addition to preventing overfitting, dropout provides an efficient way of combining and utilizing exponentially many different network structures.

2.1.3.2 Batch normalization

Batch normalization is a technique that was introduced in order to accelerate the training of neural networks [17] and has been shown to improve generalization [24]. While training a neural network, the parameters of all preceding layers will affect the input to the current layer. Hence, in each iteration of the training, the layer has to adapt to the change in distribution of the input due to the changes in other

parameters. For an input $\boldsymbol{x} = (x_1, \ldots, x_n)$, the idea is to normalize each dimension according to

$$\widehat{x}_k = \frac{x_k - \mathbb{E}[x_k]}{\sqrt{\mathrm{Var}[x_k]}}$$

where the expectation and variance are substituted by the sample mean and sample variance computed over the training data set. To avoid changing what the layer represents, an additional pair of parameters γ_k, β_k are introduced that are allowed to scale and shift $\widehat{x}_k$ according to

$$\widehat{y}_k = \gamma_k \widehat{x}_k + \beta_k.$$

In practice, it is often combined with mini-batch training. For a given batch $\mathcal{B}$, the BN-transform is computed as

$$\mathcal{BN}_{\gamma,\beta}(x_i) = \gamma \frac{x_i - \mu_{\mathcal{B}}}{\sigma^2_{\mathcal{B}}} + \beta \tag{2.28}$$

where $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}}$ denote the mean and variance over the batch. This transform is differentiable [17], and can thus be included in a layer without having to change the training procedure. It is usually applied to the output of the pre-activation $\boldsymbol{g}$. In the convolutional case, it is applied to each spatial location of a kernel.

2.2 Generative adversarial networks

Generative adversarial networks were proposed as a new strategy to train generative models by Goodfellow in 2014 [11]. The strategy involves a generative and a discriminative model, which are pitted against each other. In this section, we give a brief introduction to generative models as well as explaining different variants of generative adversarial networks.

2.2.1 Generative models

The goal of generative models is to generate data from some population, by trying to model its distribution. Let $\boldsymbol{p_r}$ be the real distribution of the data. We want to find parameters $\boldsymbol{\theta^g}$ such that the distance between $\boldsymbol{p_r}$ and the approximated distribution $\boldsymbol{p_g}(\cdot; \boldsymbol{\theta^g})$ is minimal. When an optimal $\boldsymbol{\theta^g}$ is known, we can generate new examples by drawing samples from $p_g(\cdot; \boldsymbol{\theta^g})$. It can often be that the data lies in some high-dimensional space, such as images with dimensions $N \times N$ which can be seen as an N^2-dimensional random variable. In addition, the distribution can be a very complex one, which may be hard to generate. A way to tackle this is to generate a simple random variable $\boldsymbol{z}$ with density $\boldsymbol{p_z}$, and then transform it using a complex function $\mathcal{G} : \boldsymbol{z} \mapsto \boldsymbol{x_g}$. What the function $\mathcal{G}$ should be is of course non-trivial to find out. However, we can model it using a neural network. If $\mathcal{G}(\boldsymbol{z}; \boldsymbol{\theta^g})$ is a neural network with parameters $\boldsymbol{\theta^g}$, then the density $\boldsymbol{p_z}$ induces a density $\boldsymbol{p_g}$ when it is sent through $\mathcal{G}$. Instead of adapting the parameters of $\boldsymbol{p_z}$, we can train $\mathcal{G}$ to map it to a density $\boldsymbol{p_g}$ which is close to $\boldsymbol{p_r}$.

2.2.2 The adversarial setup

In the adversarial setup, an additional discriminative model $\mathcal{D}$ is used to compare the distributions in the following way. It takes samples $\boldsymbol{x_g}$ generated by $\mathcal{G}$ or $\boldsymbol{x}$ from the data as input, and should classify whether the input is drawn from the true distribution $\boldsymbol{p_r}$ or from the generative model with distribution $\boldsymbol{p_g}$.

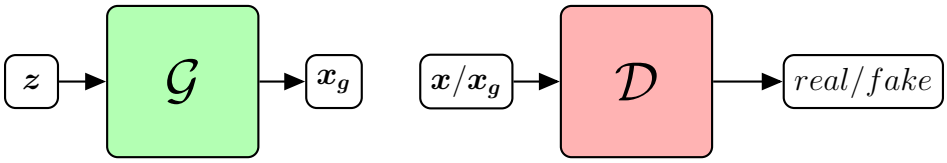


Figure 2.5: Schematic diagram of a GAN where the generator is denoted by $\mathcal{G}$ and the discriminator by $\mathcal{D}$. The generator takes a noise vector $\boldsymbol{z}$ as input and should output a generated sample $\boldsymbol{x}_g$. The discriminator takes as input either a real data point $\boldsymbol{x}$ or a generated sample $\boldsymbol{x}_g$, and should output $real$ or $fake$ depending on if the input is real or generated.

The two models $\mathcal{G}$ and $\mathcal{D}$ are trained simultaneously, but with opposite goals. Since the generator is supposed to learn to generate samples from $\boldsymbol{p_r}$, it wants to fool the discriminator that its output is a real data point. Hence, its goal is to maximize the classification error of $\mathcal{D}$. The discriminator on the other hand wants to detect fake samples generated by $\mathcal{G}$. Thus its goal is to minimize the classification error of $\mathcal{D}$. Letting $\mathcal{D}(\cdot)$ denote the probability of a sample being drawn from $\boldsymbol{p_r}$, the classification error $\mathcal{E}(\mathcal{D}, \mathcal{G})$ can be expressed as

$$\mathcal{E}(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p_r}}[\log \mathcal{D}(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim\boldsymbol{p_z}}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z})))]. \tag{2.29}$$

The strategy can be interpreted as the discriminator and generator playing a minimax game with payoff $\mathcal{E}(\mathcal{D}, \mathcal{G})$, where the objective is

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathcal{E}(\mathcal{D}, \mathcal{G}). \tag{2.30}$$

2.2.3 Conditional GAN

It is possible to turn GANs into a conditional model by passing extra information $\boldsymbol{c}$ to both the generator and discriminator [26]. As an example, consider generating images of handwritten digits. In this case, the conditional information may be the digit that should be generated. This means that the generator will learn to generate samples $\boldsymbol{x}$ conditioned on the information $\boldsymbol{c}$, i.e. samples from $\boldsymbol{p}_r(\boldsymbol{x}|\boldsymbol{c})$. The discriminator learns to distinguish between samples from $\boldsymbol{p}_r(\boldsymbol{x}|\boldsymbol{c})$ and $\boldsymbol{p}_z(\mathcal{G}(\boldsymbol{z}|\boldsymbol{c}))$. In this setup, the classification error in equation (2.29) becomes

$$\mathcal{E}(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{\boldsymbol{x}\sim\boldsymbol{p_r}}[\log \mathcal{D}(\boldsymbol{x}|\boldsymbol{c})] + \mathbb{E}_{\boldsymbol{z}\sim\boldsymbol{p_z}}[\log(1 - \mathcal{D}(\mathcal{G}(\boldsymbol{z}|\boldsymbol{c})))]. \tag{2.31}$$

The corresponding network structure is shown in Figure 2.6.

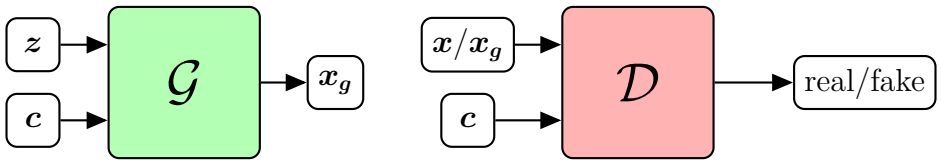


Figure 2.6: Schematic diagram of a conditional GAN. The $\mathcal{G}$ and $\mathcal{D}$ corresponds to the generator and the discriminator, respectively. The generator takes a noise vector $\boldsymbol{z}$ and conditional information $\boldsymbol{c}$ as input and should output a generated sample $\boldsymbol{x}_g$. The discriminator takes as input either a real data point $\boldsymbol{x}$ or a generated sample $\boldsymbol{x}_g$ as well as their conditional information $\boldsymbol{c}$, and should output *real* or $fake$ depending on if the input is real or generated.

2.3 Generative adversarial privacy

In this section, we describe the privacy framework called Generative adversarial privacy (GAP) [16]. In this framework, one considers a dataspace $\mathcal{X} \times \mathcal{S}$, where each data point $(\boldsymbol{x}, \boldsymbol{s})$ is a pair consisting of a public data point and a private attribute. We assume access to a dataset $X \subseteq \mathcal{X} \times \mathcal{S}$ with distribution $\boldsymbol{p}_{data}$ where each sample is associated with an individual. The public datapoint $\boldsymbol{x}$ is information that the individual wants to share. The private attribute $\boldsymbol{s}$ is information that the individual does not want to disclose. It is assumed that the private attribute can be inferred from the public data point.

A privacy mechanism is a mapping $f : \mathcal{X} \times \mathcal{Z} \to \mathcal{X}$ that transforms $\boldsymbol{x}$ into $\boldsymbol{x}' = f(\boldsymbol{x}, \boldsymbol{z})$, where $\boldsymbol{z}$ is noise, such that it is harder to infer $\boldsymbol{s}$ from $\boldsymbol{x}'$ than from $\boldsymbol{x}$. In order to measure the performance of the privacy mechanism, an adversary h is introduced, which is supposed to predict the sensitive attribute from $\boldsymbol{x}'$. How well the adversary predicts the sensitive attribute in the transformed image can be measured through a loss function $\ell(f, h)$, with expected loss

$$\mathcal{L}(f, h) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{s})\sim\boldsymbol{p}_{data}}[\ell(f, h)]. \tag{2.32}$$

On one hand, the objective of f is to fool the adversary h, meaning that the expected loss should have a high value. On the other hand, h wants to minimize the expected loss. Hence this can be formulated as a minimax game given by

$$\min_f \max_h -\mathcal{L}(f, h) \tag{2.33}$$

However, in order for the privacy mechanism to actually be useful, it might be necessary to impose a utility constraint. Otherwise one could just choose f to be a zero-mapping. Hence, a distortion constraint is introduced, by constraining the distance between $\boldsymbol{x}$ and $\boldsymbol{x}'$ with some $\varepsilon > 0$ for some distortion measure d.

$$\min_f \max_h -\mathcal{L}(f, h) \tag{2.34}$$

$$\text{s.t. } \mathbb{E}_{(\boldsymbol{x},\boldsymbol{s})\sim\boldsymbol{p}_{data}}[d(\boldsymbol{x}, \boldsymbol{x}')] \leq \varepsilon \tag{2.35}$$

As described in section 2.1.2.1, the distribution $\boldsymbol{p}_{data}$ is unknown but its possible to sample from it. To get around this, we may model the adversary and privacy mechanism using neural networks, and hence take a data-driven approach to approximate them. The privacy mechanism is modeled as a generative network $\mathcal{G}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}_{\mathcal{G}})$, parameterized by $\boldsymbol{\theta}_{\mathcal{G}}$. The adversary is a modeled as a classifier $\mathcal{D}(\boldsymbol{x}; \boldsymbol{\theta}_{\mathcal{D}})$ parameterized by $\boldsymbol{\theta}_{\mathcal{D}}$. The expected loss in equation (2.32) is replaced by

$$\mathcal{L}(\boldsymbol{\theta}_{\mathcal{G}}, \boldsymbol{\theta}_{\mathcal{D}}) = \mathbb{E}\Big[\ell\Big(\mathcal{D}(\mathcal{G}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}_{\mathcal{G}}); \boldsymbol{\theta}_{\mathcal{D}}), \boldsymbol{s}\Big)\Big] \tag{2.36}$$

resulting in the minimax game

$$\min_{\boldsymbol{\theta}_{\mathcal{G}}} \max_{\boldsymbol{\theta}_{\mathcal{D}}} - \mathcal{L}(\boldsymbol{\theta}_{\mathcal{G}}, \boldsymbol{\theta}_{\mathcal{D}}) \tag{2.37}$$

$$\text{s.t. } \mathbb{E}\left[d(\mathcal{G}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}_{\mathcal{G}}), \boldsymbol{x},)\right] \leq \varepsilon. \tag{2.38}$$

2.4 Time-frequency analysis

In this section we describe the time-frequency representations of audio used in this thesis. This representation is called a Mel spectrogram and builds upon the Fourier transform.

2.4.1 Short time Fourier transform

The Discrete Fourier Transform (DFT) $\boldsymbol{s} = \mathcal{DFT}(\boldsymbol{x}) = (s_0, \ldots, s_{N-1})$ for a signal $\boldsymbol{x} = (x_0, \ldots, x_{N-1})$ of length N is defined as

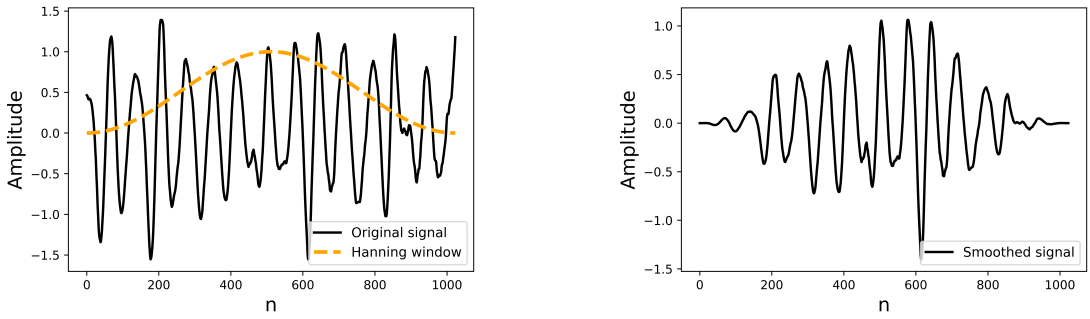
$$s_k = \sum_{n=0}^{N-1} x_n e^{\frac{-i2\pi kn}{N-1}}. \tag{2.39}$$

The DFT decomposes the signal into its frequency components averaged over the whole signal. This is useful when the signal is stationary, that is the frequency content does not vary with time. However, this is not a valid assumption for many real world signals, such as speech. To get around this, the signal can be split into different segments. If the segments are small enough, then the assumption of stationarity can be seen to be approximately valid, and hence applying the DFT to each such segment is reasonable. This procedure gives a time-frequency representation of the signal and is called the Short Time Fourier Transform (STFT).

The first step to compute the STFT of a signal $\boldsymbol{x}$ is to split into K frames $\boldsymbol{x}^{(i)}, i = 1, \ldots, K$ of length M. The frames may overlap, i.e. $x_m^{(i)} = x_n^{(j)}$ for some $i \neq j, m \neq n$. Each frame $\boldsymbol{x}^{(i)}$ is then multiplied by a window function $\boldsymbol{w}$ to smooth out the borders of the signal. This is done in order to avoid spectral leakage, which may occur due to the possible discontinuity stemming from the split [13]. The smoothed signal is $\tilde{\boldsymbol{x}}^{(i)} = \boldsymbol{w} \odot \boldsymbol{x}^{(i)}$ where $\odot$ denotes elementwise multiplication. One window function that is often used is the Hanning window $\boldsymbol{w} = (w_0, \ldots, w_{N-1})$, defined as

$$w_n = 0.5\left[1 - \cos\left(\frac{2\pi n}{N-1}\right)\right]. \tag{2.40}$$

An example of a signal multiplied with a Hanning window is shown in Figure 2.7a.



(a) **(b)**

Figure 2.7: In (a), the original signal and the Hanning window are plotted. In (b), the smoothed signal is plotted.

After this, $\boldsymbol{s}^{(i)} = \mathcal{DFT}(\tilde{\boldsymbol{x}}^{(i)})$ is computed for each frame. Computing the magnitude of each element $s_k^{(i)}$, letting $\tilde{s}_k^{(i)} = |s_k^{(i)}|^2$ and considering the $\tilde{\boldsymbol{s}}^{(i)}$ as column vectors, we can place them next to each other to form a matrix $\boldsymbol{S} = [\tilde{\boldsymbol{s}}^{(1)} \dots \tilde{\boldsymbol{s}}^{(K)}] \in \mathbb{R}^{M \times K}$ called a *spectrogram*. This real-valued matrix $\boldsymbol{S}$ can be interpreted as an image, as visualised in Figure 2.9.

2.4.2 Mel spectrogram

The Mel spectrogram is an extension of the spectrogram in the section above, modified to correlate better with human perception of sound. The human ear is better at distinguishing between lower frequencies. For example, the perceptual difference between 100 Hz and 200 Hz is greater than the difference between 4000 Hz and 4100 Hz, even though the frequency difference is the same. The Mel scale is a nonlinear frequency scale that was invented to deal with this by being linear in terms of human perception. The relationship between a frequency f and its Mel-frequency m is described by equation (2.41).

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right). \tag{2.41}$$

In order to transform the original spectrogram to a Mel spectrogram it is passed through a Mel-scale filter bank. A Mel-scale filter bank consists of triangular filters with centers uniformly spaced in Mel-scale. Given a frequency range $[f_{min}, f_{max}]$, the corresponding Mel-frequency range $[m_{min}, m_{max}]$ is computed using formula (2.41). Given that we want F filters, we uniformly partition the Mel-frequency range into nodes $m_{min} = m_0 < \ldots < m_{F+1} = m_{max}$. Each filter $\tilde{b}^{(i)} : \mathbb{R} \to [0, 1]$ is a triangular hat function with center m_i and support $[m_{i-1}, m_{i+1}]$. The corresponding filter $b^{(i)}$ in the normal frequency scale is shown in Figure 2.8, found by inverting the Mel-frequency range back.

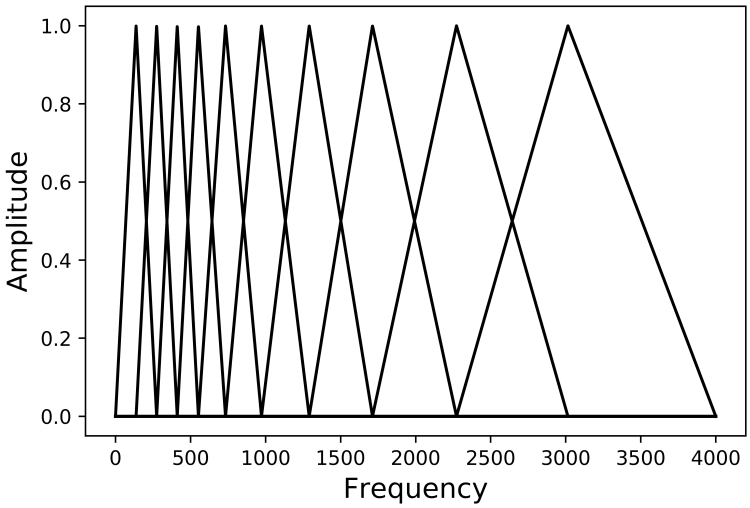
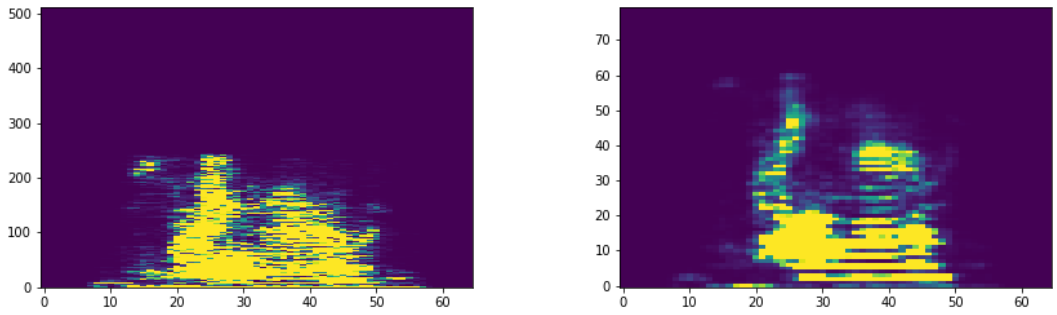


Figure 2.8: Mel-scale filter bank with 10 filters.

To pass a spectrogram $\boldsymbol{S} \in \mathbb{R}^{M \times K}$ through this filter bank, we construct a matrix $\boldsymbol{B} \in \mathbb{R}^{F \times M}$ where each row i is a discretization of the filter $b^{(i)}$; if $\boldsymbol{B}_{i,.}$ denotes the ith row, then $\boldsymbol{B}_{i,.} = [b^{(i)}(m_0) \ \cdots \ b^{(i)}(m_{F+1})]$. Multiplying the spectrogram $\boldsymbol{S}$ with the matrix $\boldsymbol{B}$ and taking the logarithm then produces the Mel spectrogram $\boldsymbol{S}_{mel} \in \mathbb{R}^{F \times K}$. An example of a spectrogram and its Mel spectrogram with 80 filters is shown in Figure 2.9.



(a) **(b)**

Figure 2.9: In (a), the original magnitude spectrogram is shown. In (b), its corresponding Mel spectrogram with 80 filters is shown.

3
Methods

We now proceed to describe the models and data used in this project, as well as the experiments we have conducted and the metrics we have used to evaluate the results.

3.1 MelGAN

MelGAN is a fully convolutional model designed to invert Mel spectrograms to raw waveform [21]. The MelGAN generator $\mathcal{G}$ consists of a stack of transposed convolutional layers. The model uses three different discriminators $\mathcal{D}_1, \mathcal{D}_2$ and $\mathcal{D}_3$, each operating at different resolutions. All discriminators operate on raw audio, the first one at the same resolution as the input, while the second and third on signals downsampled by a factor of two and four, respectively. The discriminators are trained using a hinge loss version [23] of GAN objective. The generator is trained using the original GAN objective, but also using a *feature matching loss* [22]. Instead of having a distortion loss such as in GAP, the output of the layers in the discriminators are seen as feature spaces in which the similarity between real and fake date can be measured. For each layer i, let $\mathcal{D}_k^{(i)}(\cdot)$ denote the output from the kth discriminator. The feature matching loss is computed as

$$\mathcal{L}_{\text{FM}}\left(\mathcal{G}, \mathcal{D}_k\right) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{m}\sim p_{\text{data}}}\left[\sum_{i=1}^{T} \frac{1}{N_i}\left\|\mathcal{D}_k^{(i)}(\boldsymbol{x}) - \mathcal{D}_k^{(i)}(\mathcal{G}(\boldsymbol{m}))\right\|_1\right] \tag{3.1}$$

where N_i is the number of output units in layer i, $\boldsymbol{x}$ is the raw audio signal and $\boldsymbol{m}$ is its corresponding Mel spectrogram. Intuitively, this loss can be seen as a learned similarity metric. The training objective for the discriminators and the generator are then

$$\min_{\mathcal{D}_k}\left(\mathbb{E}_{\boldsymbol{x}}\Big[\min\left(0, 1-\mathcal{D}_k(\boldsymbol{x})\right)\Big] + \mathbb{E}_{\boldsymbol{m},\boldsymbol{z}}\Big[\min\left(0, 1+\mathcal{D}_k(\mathcal{G}(\boldsymbol{m},\boldsymbol{z}))\right)\Big]\right), \forall k = 1,2,3 \tag{3.2}$$

$$\min_{\mathcal{G}}\left(\mathbb{E}_{\boldsymbol{m},\boldsymbol{z}}\left[\sum_{k=1}^{3} -\mathcal{D}_k(\mathcal{G}(\boldsymbol{m},\boldsymbol{z}))\right] + \gamma\sum_{k=1}^{3}\mathcal{L}_{\text{FM}}\left(\mathcal{G},\mathcal{D}_k\right)\right) \tag{3.3}$$

where γ is a hyperparameter controlling the balance between feature matching and fooling the discriminator. For further details as well as the official implementation of the model, we refer to the original article [21].

3.2 Private conditional GAN

Private conditional GAN (PCGAN) [25] is a model that builds upon the Generative Adversarial Privacy framework described in section 2.3. PCGAN consists of two GAP modules, a *filter* $\mathcal{F}$, and a *generator* $\mathcal{G}$, that are composed to form the model. In other words, the privacy mechanism is the composition $\mathcal{G} \circ \mathcal{F}(\boldsymbol{x})$. They examine the method on image data, to see if the addition of a generator is better than just using the filter as in the original paper [16]. The purpose of the filter is to censor the sensitive attribute s in the image. The objective of the generator is to take the filtered image $\boldsymbol{x}'$ and generate a new synthetic instance s' of the sensitive attribute in it, independent of the original value s. An overview of the model is shown in Figure 3.2, where $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are noise, $\mathcal{D}_\mathcal{F}$ and $\mathcal{D}_\mathcal{G}$ are discriminators for the filter and generator, respectively.

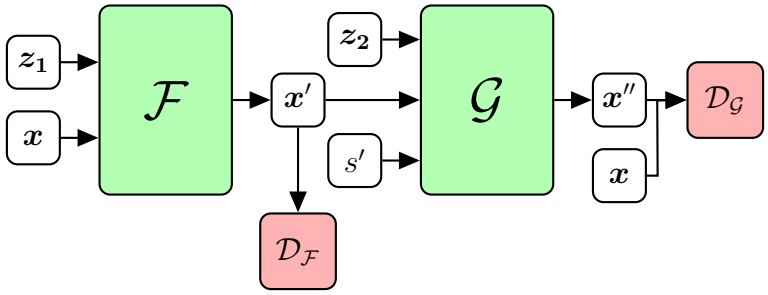


Figure 3.1: Schematic diagram of PCGAN. The original image $\boldsymbol{x}$ and a noise vector $\boldsymbol{z}_1$ is input into $\mathcal{F}$. The filtered image $\boldsymbol{x}'$ is input into $\mathcal{G}$ together with noise vector $\boldsymbol{z}_2$ and the value s' of the sensitive attribute.

As introduced in section 2.3, the filter module's training objective is the minimax game given by

$$\min_{\boldsymbol{\theta}_\mathcal{F}} \max_{\boldsymbol{\theta}_{\mathcal{D}_\mathcal{F}}} \mathbb{E}\Big[\ell_\mathcal{F}\Big(\mathcal{D}_\mathcal{F}(\mathcal{F}(\boldsymbol{x}, \boldsymbol{z}_1; \boldsymbol{\theta}_\mathcal{F}); \boldsymbol{\theta}_{\mathcal{D}_\mathcal{F}}), s\Big)\Big]$$
$$\text{s.t. } \mathbb{E}\Big[d\left(\mathcal{F}\left(\boldsymbol{x}, \boldsymbol{z}_1; \boldsymbol{\theta}_\mathcal{F}\right), \boldsymbol{x}\right)\Big] \leq \epsilon_1$$

The generator's objective is similar to the one in GAP, but uses a semisupervised objective instead. The loss is split into a supervised part, corresponding to classification of the sensitive attribute in real inputs, and an unsupervised part, corresponding to classification of the sensitive attribute in generated inputs. Hence, the objective is given by the minimax game

$$\min_{\boldsymbol{\theta}_\mathcal{G}} \max_{\boldsymbol{\theta}_{\mathcal{D}_\mathcal{G}}} \mathbb{E}\Big[\ell_\mathcal{G}\left(\mathcal{D}_\mathcal{G}\left(\mathcal{G}\left(\mathcal{F}\left(\boldsymbol{x}, \boldsymbol{z}_1; \boldsymbol{\theta}_\mathcal{F}\right), s', \boldsymbol{z}_2; \boldsymbol{\theta}_\mathcal{G}\right)\right), fake\right)\Big] + \mathbb{E}\Big[\ell_\mathcal{G}\left(\mathcal{D}_\mathcal{G}(\boldsymbol{x}; \boldsymbol{\theta}_{\mathcal{D}_\mathcal{G}}), s\right)\Big]$$
$$\text{s.t. } \mathbb{E}\Big[d\left(\mathcal{G}\left(\mathcal{F}\left(\boldsymbol{x}, \boldsymbol{z}_1; \boldsymbol{\theta}_\mathcal{F}\right), s', \boldsymbol{z}_2; \boldsymbol{\theta}_\mathcal{G}\right), \boldsymbol{x}\right)\Big] \leq \epsilon_2$$

The authors implement $\mathcal{F}$ and $\mathcal{G}$ using the UNet architecture [30]. The adversaries $\mathcal{D}_\mathcal{F}$ and $\mathcal{D}_\mathcal{G}$ are implemented as ResNet[14] networks, where the last fully connected layers were replaced with a two and three class output layer, respectively. The loss functions $\ell_\mathcal{G}$ and $\ell_\mathcal{F}$ are implemented as categorical cross entropy and the distortion measure d is the L^2-norm.

3.3 PCMelGAN

Now, we turn to the presentation of our set up which combines MelGAN and PCGAN. We consider raw waveform speech recordings $\boldsymbol{x}$ paired with a sensitive attribute s. The sensitive attribute is assumed to be binary, i.e. $s \in \{0, 1\}$. Instead of directly modelling the raw audio, we transform the recording to a Mel spectrogram according to the procedure presented in section 2.4.2 and work on spectrograms instead. Spectrograms have been shown to contain relevant features about audio data and been used extensively for classification tasks [1]. However, since we want to generate audio a downside is that such spectrograms are not invertible, as they only contain the magnitude and not the phase. The most well-known algorithm that approximate inversions of spectrograms is the Griffin-Lim algorithm [12]. A more modern approach is the MelGAN described in section 3.1 which has been shown to produce realistic inversions. We argue that those inversions are realistic enough to justify working with spectrograms instead of raw audio. Thus, we decide to use a privacy mechanism that operates on spectrograms.

The whole pipeline is shown in Figure 3.2. The speech recording $\boldsymbol{a}$ is transformed to a Mel spectrogram $\boldsymbol{m}$. The Mel spectrogram is then censored by the PCGAN privacy mechanism consisting of a filter $\mathcal{F}$ and a generator $\mathcal{G}$. The censored Mel spectrogram $\boldsymbol{m}''$ is inverted to audio $\boldsymbol{a}'$ by a pre-trained MelGAN network.

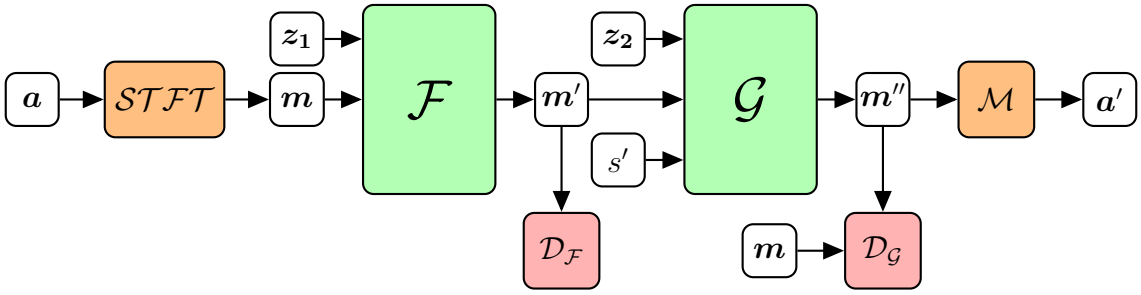


Figure 3.2: Schematic diagram of PCMelGAN. The audio recording $\boldsymbol{a}$ is transformed into a Mel spectrogram $\boldsymbol{m}$. The filtered Mel spectrogram is denoted $\boldsymbol{m}'$, and $\boldsymbol{m}''$ denotes the resulting Mel spectrogram from the generator. The sampled sensitive attribute is denoted s' and $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$ are noise vectors. The Mel spectrogram $\boldsymbol{m}''$ is inverted to audio $\boldsymbol{a}'$ with a pre-trained MelGAN $\mathcal{M}$.

Similarly to Martinsson et al. [25] we implement $\mathcal{F}$ and $\mathcal{G}$ using the UNet architecture. The adversaries $\mathcal{D}_\mathcal{F}$ and $\mathcal{D}_\mathcal{G}$ are, however, implemented as AlexNets[20] and not ResNets. The last fully connected layer are modified to output two and three classes. The choice of network architecture is motivated by the fact that AlexNet has achieved high accuracy on gender classification in spectrograms [1]. Additionally, initial experiments using ResNet did not yield promising results. The training procedure of the full model is shown in Algorithm 1. The loss functions $\ell_\mathcal{F}$ and $\ell_\mathcal{G}$ are categorical cross entropy and the distortion measure d is the L^1-norm. The constrained optimization problem is reformulated as an unconstrained one by relaxing it using the quadratic penalty method [27]. The distortion constraint is denoted by

ε and the penalty parameter by λ. The parameters are updated using the Adam update rule (2.27) denoted by $Adam(\boldsymbol{\theta}; \eta, \beta_1, \beta_2)$.

Algorithm 1: PCMelGAN

Input: $\mathcal{D}$, η, λ, ϵ

repeat

Draw n samples uniformly at random from the dataset

$(\boldsymbol{x}_1, s_1), \ldots, (\boldsymbol{x}_n, s_n) \sim \boldsymbol{p}_{empirical}$

Compute Mel spectrogram and normalize

$\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n = \mathcal{STFT}(\boldsymbol{x}_1), \ldots \mathcal{STFT}(\boldsymbol{x}_n)$

Draw n samples from the noise distribution

$\boldsymbol{z}_1^{(1)}, \ldots, \boldsymbol{z}_n^{(1)} \sim \mathcal{N}(0, 1)$

$\boldsymbol{z}_1^{(2)}, \ldots, \boldsymbol{z}_n^{(2)} \sim \mathcal{N}(0, 1)$

Draw n samples from the synthetic distribution

$s'_1, \ldots, s'_m \sim \mathcal{U}\{0, 1\}$

Compute the censored and synthetic data

$\boldsymbol{m}'_1, \ldots, \boldsymbol{m}'_n = \mathcal{F}(\boldsymbol{m}_1, \boldsymbol{z}_1^{(1)}; \boldsymbol{\theta}_{\mathcal{F}}), \ldots, \mathcal{F}(\boldsymbol{m}_n, \boldsymbol{z}_n^{(1)}; \boldsymbol{\theta}_{\mathcal{F}})$

$\boldsymbol{m}''_1, \ldots, \boldsymbol{m}''_n = \mathcal{G}(\boldsymbol{m}'_1, s'_1, \boldsymbol{z}_1^{(2)}; \boldsymbol{\theta}_{\mathcal{G}}), \ldots, \mathcal{G}(\boldsymbol{m}'_n, s'_n, \boldsymbol{z}_n^{(2)}; \boldsymbol{\theta}_{\mathcal{G}})$

Compute filter and generator loss

$$\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}_{\mathcal{F}}; \boldsymbol{p}_{empirical}) = -\frac{1}{n}\sum_{i=1}^{n} \ell(\mathcal{D}_{\mathcal{F}}(\boldsymbol{m}'_i; \boldsymbol{\theta}_{\mathcal{D}_{\mathcal{F}}}), s_i) + \lambda \max\left(\frac{1}{n}\sum_{i=1}^{n} d(\boldsymbol{m}'_i, \boldsymbol{m}_i) - \varepsilon, 0\right)^2$$

$$\mathcal{L}_{\mathcal{G}}(\boldsymbol{\theta}_{\mathcal{G}}; \boldsymbol{p}_{empirical}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathcal{D}_{\mathcal{G}}(\boldsymbol{m}''_i; \boldsymbol{\theta}_{\mathcal{D}_{\mathcal{G}}}), s_i) + \lambda \max\left(\frac{1}{n}\sum_{i=1}^{n} d(\boldsymbol{m}''_i, \boldsymbol{m}_i) - \varepsilon, 0\right)^2$$

Update filter and generator parameters

$\boldsymbol{\theta}_{\mathcal{F}} \leftarrow \text{Adam}(\boldsymbol{\theta}_{\mathcal{F}}; \eta_{\mathcal{F}}, \beta_1, \beta_2)$

$\boldsymbol{\theta}_{\mathcal{G}} \leftarrow \text{Adam}(\boldsymbol{\theta}_{\mathcal{G}}; \eta_{\mathcal{G}}, \beta_1, \beta_2)$

Compute discriminator losses

$$\mathcal{L}_{\mathcal{D}_{\mathcal{F}}}(\boldsymbol{\theta}_{\mathcal{D}_{\mathcal{F}}}; \boldsymbol{p}_{empirical}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathcal{D}_{\mathcal{F}}(\boldsymbol{m}'_i; \boldsymbol{\theta}_{\mathcal{D}_{\mathcal{F}}}), s_i)$$

$$\mathcal{L}_{\mathcal{D}_{\mathcal{G}}}(\boldsymbol{\theta}_{\mathcal{D}_{\mathcal{G}}}; \boldsymbol{p}_{empirical}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathcal{D}_{\mathcal{G}}(\boldsymbol{m}''_i; \boldsymbol{\theta}_{\mathcal{D}_{\mathcal{G}}}), fake) + \frac{1}{n}\sum_{i=1}^{n} \ell(\mathcal{D}_{\mathcal{G}}(\boldsymbol{m}_i; \boldsymbol{\theta}_{\mathcal{D}_{\mathcal{G}}}), s_i)$$

Update discriminator parameters

$\boldsymbol{\theta}_{\mathcal{D}} \leftarrow \text{Adam}(\boldsymbol{\theta}_{\mathcal{D}}; \eta_{\mathcal{D}}, \beta_1, \beta_2)$

until *termination criterion is met*;

Similarly to [25], we use a PCMelGAN where the generator module is excluded. In their case they were extending such a model and hence it served as a baseline. When it comes to audio, neither model has been investigated before and therefore allows us to compare the two against each other. The pipeline of this model is illustrated in Figure 3.3. The training procedure is similar to the one of PCMelGAN and is presented in Algorithm 2.

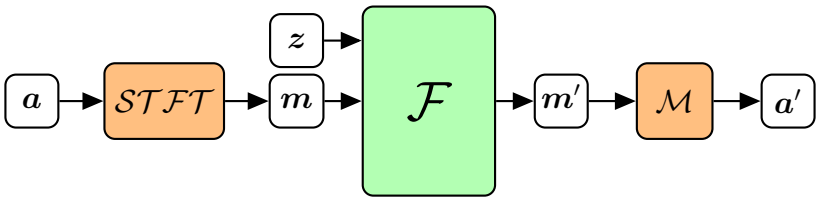


Figure 3.3: Schematic diagram of the filter model. The audio recording $\boldsymbol{a}$ is transformed into a Mel spectrogram $\boldsymbol{m}$. The filtered Mel spectrogram is denoted $\boldsymbol{m}'$ and the noise vector $\boldsymbol{z_1}$. The Mel spectrogram $\boldsymbol{m}'$ is inverted to audio $\boldsymbol{a}'$ with a pre-trained MelGAN $\mathcal{M}$.

Algorithm 2: Filter

Input: $\eta_{\mathcal{F}}, \eta_{\mathcal{D}}, \lambda, \epsilon$

repeat

  Draw n samples uniformly at random from the dataset

  $(\boldsymbol{x}_1, s_1), \ldots, (\boldsymbol{x}_n, s_n) \sim \boldsymbol{p}_{empirical}$

  Compute Mel spectrogram and normalize

  $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n = \mathcal{STFT}(\boldsymbol{x}_1), \ldots \mathcal{STFT}(\boldsymbol{x}_n)$

  Draw n samples from the noise distribution

  $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n \sim \mathcal{N}(0, 1)$

  Compute the transformed data

  $\boldsymbol{m}'_1, \ldots, \boldsymbol{m}'_n = \mathcal{F}(\boldsymbol{x}_1, \boldsymbol{z}_1; \boldsymbol{\theta}_{\mathcal{F}}), \ldots, \mathcal{F}(\boldsymbol{x}_n, \boldsymbol{z}_n; \boldsymbol{\theta}_{\mathcal{F}})$

  Compute filter loss

$$\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}_{\mathcal{F}}; \boldsymbol{p}_{empirical}) = -\frac{1}{n}\sum_{i=1}^{n} \ell(\mathcal{D}(\boldsymbol{m}'_i; \boldsymbol{\theta}_{\mathcal{D}}), s_i) + \lambda \max\left(\frac{1}{n}\sum_{i=1}^{n} d(\boldsymbol{m}'_i, \boldsymbol{m}_i) - \varepsilon, 0\right)^2$$

  Update filter parameters

  $\boldsymbol{\theta}_{\mathcal{F}} \leftarrow \text{Adam}(\boldsymbol{\theta}_{\mathcal{F}}; \eta_{\mathcal{F}}, \beta_1, \beta_2)$

  Compute discriminator losses

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\theta}_{\mathcal{D}}; \boldsymbol{p}_{empirical}) = \frac{1}{n}\sum_{i=1}^{n} \ell(\mathcal{D}(\boldsymbol{m}'_i; \boldsymbol{\theta}_{\mathcal{D}}), s)$$

  Update discriminator parameters

  $\boldsymbol{\theta}_{\mathcal{D}} \leftarrow \text{Adam}(\boldsymbol{\theta}_{\mathcal{D}}; \eta_{\mathcal{D}}, \beta_1, \beta_2)$

until *termination criterion is met*;

3.4 Data

The dataset used to evaluate our model was the AudioMNIST dataset [1], which consists of 30000 audio recordings of spoken digits $(0-9)$ in English. The dataset comprises of 60 different speakers with 50 repetitions for each digit. They were recorded with a sampling frequency of 48kHz. The meta data contains information about the speaker's age, gender, origin and accent. The dataset consists of 12 women and 48 men. In order to work with a balanced dataset with respect to gender, we randomly sample 12 men and limit the data used to speech recordings from the 12 women and the 12 randomly sampled men. This limited data is split into a training set and a test set, consisting of 10000 and 2000 samples, respectively. To achieve a similar balance in the splits, ten men and ten women are randomly sampled, and their recordings are chosen to be the training set. The test set consists of the recordings from the remaining two men and two women.

3.5 Experiments

There are two preprocessing procedures that are used. The first one is concerned with making the audio samples of the same length. The architecture used in the generator requires all inputs to be of the same size, and the original MelGAN works on segments of length 8192. Since the recordings in the data set are one second or less, we downsample them to 8 kHz and use zero padding to get a segment length of 8192. The zero padding is done by adding zeros on both sides until the desired length is attained. The second one is concerned with stabilizing the training of the GANs. It is common that the input to such networks is normalized to $[-1, 1]$. In our case, the spectrograms are preprocessed similarly to [7], where each spectrogram is normalized to zero mean and unit variance, followed by clipping the spectra to three standard deviations and rescaling it to $[-1, 1]$. It should be noted that MelGAN does not work on spectrograms in this range. Since the last activation function of the generators are tanh which produces values in $[-1, 1]$, the transformed spectrograms are denormalized using the same mean and standard deviation as in the preprocessing step above.

Next, we describe the hyperparameters for the models. Due to time constraints, it has not been possible to do an exhaustive search of the hyperparameter space. We therefore choose these according to what has worked well with different networks throughout course of the project. Hence, we have the quadratic penalty coefficient $\lambda = 10^2$, the learning rate for discriminators $\eta = 4 \times 10^{-4}$, the learning rate for the filter and generator $\eta = 10^{-4}$.

Recall that the aim is to censor the gender in a recording. Hence, the sensitive attribute s is gender with $male \leftrightarrow 1$ and $female \leftrightarrow 0$. Since the digit attribute contains all the information about the spoken words and can be inferred both by humans and classification models, it is chosen as utility attribute. To investigate the privacy vs utility trade-off, we consider different values of the hyperparameter ε. To

allow for transformations from male spectrograms to female spectrograms and vice versa, one must take into consideration how large this distortion tends to be. To this end, we sample six males and females, and compute the distortion measure pairwise between the corresponding spectrograms for each digit. Approximate density plots of these distortions, digitwise, are illustrated in Figure 3.4.

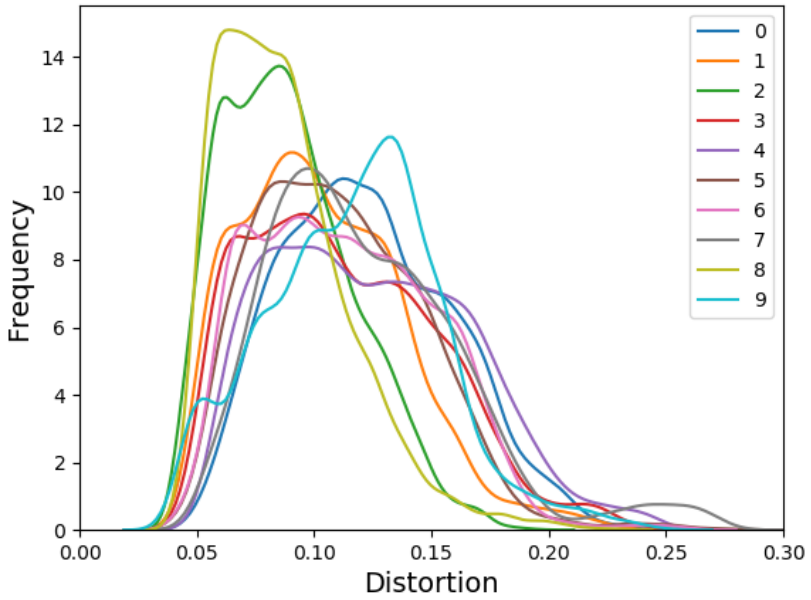


Figure 3.4: Density plot of inter gender distortion measures (L^1-norm pixelwise) of six men and six women. The distortion is computed digitwise.

The plots indicates that the distortions lies in the interval $[0.01, 0.2]$. An initial training session with $\varepsilon = 0.2$ resulted in larger distortions than desired. Hence, we choose to conduct our experiments with smaller distortion constraints, choosing $\varepsilon \in \{0.005, 0.01, 0.05, 0.1\}$.

We are now ready to describe the experimental setup. Before conducting the experiments, we train a MelGAN network on our dataset according to the official implementation[1]. Then, we train PCMelGAN and the filter model according to Algorithm 1 and Algorithm 2, respectively. This procedure is repeated fives times for each choice of ε to obtain more robust results. The hyperparameters are summarized in Table (3.1).

[1]https://github.com/descriptinc/melgan-neurips

Table 3.1: Table of hyperparameters

Name	Description	Value
λ	quadratic penalty	10^2
$\eta_{\mathcal{F}}$	learning rate for filter	10^{-4}
$\eta_{\mathcal{F}_{\mathcal{D}}}$	learning rate for filter discriminator	4×10^{-4}
$\eta_{\mathcal{G}}$	learning rate for generator	10^{-4}
$\eta_{\mathcal{G}_{\mathcal{D}}}$	learning rate for filter discriminator	4×10^{-4}
ε	distortion constraint	$\{0.005, 0.01, 0.05, 0.1\}$
β_1, β_2	exponential decay rates	0.5, 0.9

Each experiment is carried out on a NVIDIA V100 GPU. The training is restricted to 1000 epochs which takes about 10 hours.

3.6 Evaluation metrics

The absence of an objective measure in GANs makes it difficult to directly evaluate the performance of a given model. However, combining quantitative measures and qualitative assessment can serve as an adequate and robust substitute [6]. In our work, there are two performance aspects of the model that are of interest. Firstly, how well the sensitive attribute has been censored, and secondly, if the utility has been preserved. To evaluate the models quantitatively we use three different metrics. Firstly, two different fixed classifiers pre-trained to predict the ground truth labels gender and digit, in order to measure the model's ability to censor and preserve utility, respectively. The third metric is called Fréchet Inception Distance (FID) and is used to quantify the audio quality. In addition to this, we will do a qualitative assessment of the generated audio in terms of auditory inspection.

3.6.1 Fixed classifiers

In order to quantitatively evaluate how well the sensitive attribute has been censored as well as if the utility has been preserved, we employ fixed classifiers. We define the utility to be that one can determine the number uttered in the recording. By fixed classifiers we mean pre-trained classification networks, trained to predict the gender and the digit in a spectrogram as well as raw audio. The classifiers which works on Mel spectrograms are implemented as AlexNets, where the final layer is modified to have two output classes in the gender classifier and ten output classes in the digit classifier. The classifiers on raw audio waveform are implemented as AudioNets [1]. All networks were pre-trained, by training on 80 % of the training set, and using the remaining 20 % for validation. The training is done using an early stopping criterion, i.e. stopping the training when the validation loss has not been updated for 100 epochs. The classifiers on Mel spectrograms achieved their best accuracy at epoch 29, attaining an accuracy of 100 % for gender and 98 % for digit on the validation set. The classifiers on raw audio were trained for 36 and 26 epochs, achieving an accuracy of 99 % for gender and 94 % for digit on the validation set.

The fixed classifiers are used in the following way; the level of privacy is measured as how rarely the classifier predicts the original gender. The level of utility is measured as how often the classifier predicts the original digit.

3.6.2 Fréchet Inception Distance

The Fréchet Inception Distance (FID) was introduced by Heusel et al. [15] as a measure to quantify the quality of generated images. It is one of the most widely adopted evaluation metrics for GANs and has shown to correlate well with human evaluation [5, 32]. The idea of FID is to embed a set of real samples $\boldsymbol{x}$ and a set of generated samples $\boldsymbol{g}$ into a feature space. These embeddings are found by extracting the output from a specific layer in a CNN trained for classification. This network is typically an Inception Net, however, any convolutional architecture suffices [5]. The embedded vectors are assumed to be multivariate Gaussian distributed and their corresponding mean μ and covariance Σ are estimated. The FID score of the real samples $\boldsymbol{x}$ and the generated samples $\boldsymbol{g}$ is defined as

$$FID(\boldsymbol{x}, \boldsymbol{g}) = ||\mu_{\boldsymbol{x}} - \mu_{\boldsymbol{g}}||_2^2 + \mathrm{Tr}(\Sigma_{\boldsymbol{x}} + \Sigma_{\boldsymbol{g}} - 2(\Sigma_{\boldsymbol{x}}\Sigma_{\boldsymbol{g}})^{\frac{1}{2}}) \tag{3.4}$$

A lower FID score indicates that the two different sets of samples have more similar statistics and consequently that the generated samples have a higher quality. We implemented the FID-embedding network in the audio domain as an AudioNet. The network was trained to predict digits in the training set introduced in section 3.4 and achieved 94% on the validation set after 26 epochs. The features used to compute the FID-score was chosen to be the output from the final convolutional layer.

4
Results

In this chapter we present the results obtained from the experiments described in section 3.5. Recall that we trained a filter model and a PCMelGAN five times each, for four different values of the distortion constraint ε. The presentation of the results is laid out as follows: Firstly, we present the privacy vs utility trade-off when evaluated using the fixed classifiers. Secondly, we examine the quality, in terms of FID-score, of the audio obtained by inverting the transformed spectrograms using MelGAN. Finally, we provide a link to a collection of transformed audio samples.

We begin by presenting the privacy vs utility trade-off. Recall that privacy is measured by the accuracy of the fixed classifier predicting the original gender s_i, where an accuracy close to 50% corresponds to more privacy. Utility is measured by the accuracy of the fixed classifier predicting the digit u_i, where a higher accuracy corresponds to greater utility. In Table 4.1, the accuracy of the fixed classifiers on Mel spectrograms transformed by the Filter and PCMelGAN are shown. Each entry in the table corresponds to the mean and the standard deviation over five runs using different random seeds for a fixed value of ε. Table 4.2 shows the corresponding accuracies after transforming the spectograms back to raw waveform using the pretrained MelGAN.

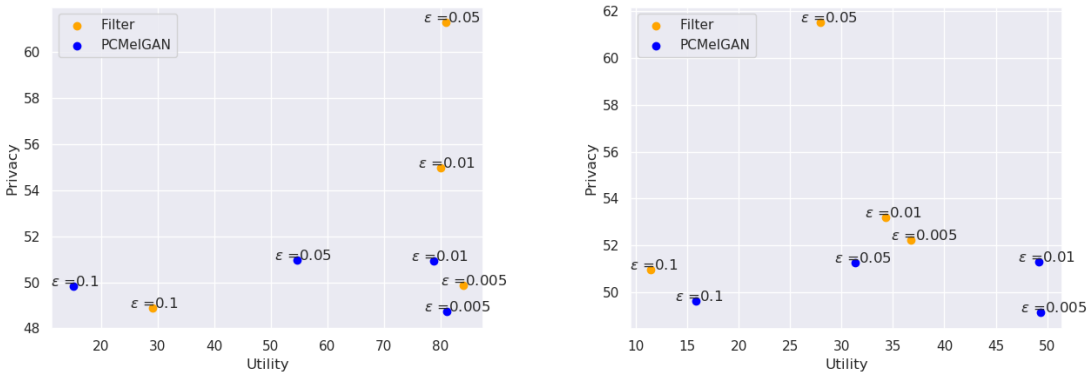
Table 4.1: The spectrogram classifiers' mean accuracy and standard deviation on the test set for varying values of ε. For privacy (gender) an accuracy close to 50% is better. For utility (digit), a higher accuracy is better.

Dist.	**Privacy**		**Utility**	
ε	Filter	PCMelGAN	Filter	PCMelGAN
0.005	49.9 ± 2.2	48.7 ± 2.4	84.1 ± 2.8	81.1 ± 3.7
0.01	55.0 ± 4.7	50.9 ± 1.4	79.9 ± 4.3	78.8 ± 7.8
0.05	61.3 ± 10.2	51.0 ± 0.7	80.9 ± 8.2	54.7 ± 23.8
0.1	48.9 ± 1.0	49.8 ± 0.5	29.1 ± 7.5	15.1 ± 5.4

Table 4.2: The raw waveform classifiers' mean accuracy and standard deviation on the test set for varying values of ε. For privacy (gender) an accuracy close to 50% is better. For utility (digit), a higher accuracy is better.

Dist.	**Privacy**		**Utility**	
ε	Filter	PCMelgan	Filter	PCMelgan
0.005	52.2 ± 3.6	49.1 ± 1.6	36.8 ± 4.0	49.4 ± 9.8
0.01	53.2 ± 3.2	51.3 ± 1.6	34.3 ± 8.5	49.2 ± 8.6
0.05	61.5 ± 8.1	51.2 ± 0.7	28.0 ± 15.8	31.3 ± 10.3
0.1	51.0 ± 1.3	49.6 ± 0.4	11.4 ± 1.7	15.8 ± 2.3

To illustrate the trade-off between privacy and utility, the mean accuracies in Table 4.1 and 4.2 are plotted against each other in Figure 4.1. Each point corresponds to the mean accuracy of each fixed classifier for a specific epsilon. The lower right corner corresponds to a higher utility and a greater privacy.



(a) **(b)**

Figure 4.1: Privacy vs utility trade-off on Mel spectrogram (a) and raw audio waveform (b). The x-axis corresponds to the percentage of times the fixed digit classifiers predicts the correct digit. The y-axis is the percentage of times the fixed gender classifier predict the original gender. The blue and orange points corresponds to the Filter and PCMelGAN, respectively.

Table 4.3: The FID scores of the two models on raw audio waveform for different distortion constraints ε. Each score corresponds to a mean and standard deviation of five identical experiments but initialized with different seeds, where a lower score is better.

Dist.	**FID Audio**	
ε	Filter	PCMelgan
0.005	20.17 ± 4.04	10.12 ± 3.15
0.01	27.27 ± 4.50	10.02 ± 2.27
0.05	29.59 ± 5.77	20.22 ± 4.87
0.1	41.50 ± 3.49	22.32 ± 5.20

In Table 4.3, FID scores are shown for our model working in the audio domain.

Finally, we provide samples from the AudioMNIST test set that were transformed by our model [1]. The shared folder contains original sound clips and their corresponding transformed versions. Figure 4.2 illustrates a Mel spectrogram transformation using PCMelGAN for a recording of a woman saying "zero".

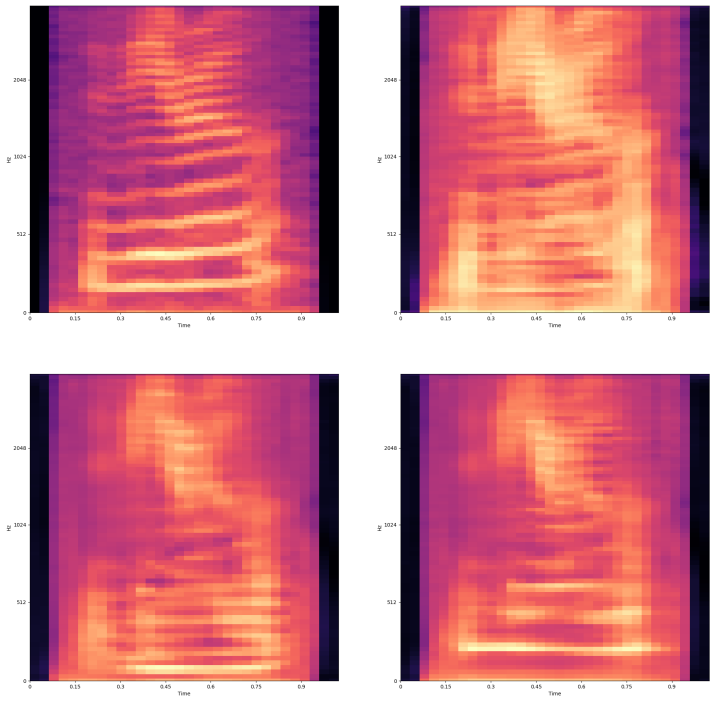


Figure 4.2: Spectrograms of a person saying 'zero'. The original recording of a female (top left), transformed ones from the baseline (top right), and our model of a sampled male (bottom left) and a sampled female (bottom right).

[1] https://www.dropbox.com/sh/oangx84ibhzodhs/AAAfG-PBW4Ne8KwdipAmKFy1a?dl=0

5
Discussion

In this chapter we discuss the results of this thesis, and consider possible improvements of the model and the experiments.

Concerning the results in the spectrogram domain, it can be seen in Table 4.1 that the Filter and the PCMelGAN achieve strong privacy for all values of ε. The difference between the two models is very small, indicating that the extension in PCMelGAN does not necessarily improve upon the privacy. We assumed initially that the privacy would suffer from having a stricter distortion constraint ε, but this was not observed in the experiments. However, more experiments need to be carried out to detect when privacy starts to deteriorate with lower ε. With regards to utility, the results are good for small values of ε, but for the largest value the utility decreases significantly. We believe the decrease in utility may come from the fact that a larger distortion constraint allows the model to make greater changes to spectrogram, possibly removing the information about the digit.

Continuing with the transformed audio, we see in Table 4.2 that both models are able to provide strong privacy in the audio domain. However, in this case it is to a greater loss of utility. For the two smaller values of epsilon, this loss of utility is greater for the Filter, indicating that the addition of a synthetic instance of the sensitive attribute improves the utility. Since Mel spectrograms are not invertible, it is reasonable that some information about the spoken content is not reconstructed, which may explain the loss of utility. While this raises the question if the spectrogram domain is reasonable to work in, it should be noted that generating coherent raw audio with GANs has had limited success so far. Although it would have been more ideal to work with raw audio, we argue that working on spectrograms is a good first step.

In Table 4.3 we notice that PCMelGAN obtains substantially better FID scores than the Filter in the audio domain. We also observed this from listening to the generated sounds. Our hypothesis is that adding the synthetic sample of the sensitive attribute results in a more realistic speech signal. This could perhaps be explained by the fact that the goal of the filter network is to hide the sensitive attribute, possibly removing it from the signal, and the only restriction on how realistic it sounds is the distortion constraint.

The choice of distortion measure may have a great impact on the transformation learnt by the network. We chose to use the L^1-norm, as it gave the most promising

results in comparison with mean square loss and L^2-norm loss during our test runs. However, it is not clear if measuring the distortion between spectrograms using any of these norms actually capture the distortion in the intelligibility of the speech. This is an aspect we believe needs to be further researched.

Another aspect that also requires further investigation is how well the models generalise to more complex speech signals as well as more diverse datasets. AudioMNIST is rather restricted in the sense that it only contains ten different utterances, which are not expressed in the same signal, and there are only 24 different speakers. It would be of interest to evaluate the model on a larger dataset with full sentences. Since the utility cannot be measured by a simple classifier such as in the case of AudioMNIST, one could instead use a transcription model and measure the utility via the word error rate.

Finally, there may be potential improvements in the evaluation of the realism and utility of the generated speech signals. Ideally, we would have liked to conduct listening tests with a large group of people to create a Mean Opinion Score (MOS), but this was out of scope for the project. Using FID as alternative measure was deemed to be a good middleground. If time had allowed, it would have been interesting to compare it with the Fréchet DeepSpeech distance [3].

5.1 Conclusion

In this work we have proposed an adversarially trained model that learns to make speech data private. We do this by first filtering a sensitive attribute, and then generating a new, independent sensitive attribute. We formulate this as an unconstrained optimization problem with a distortion budget. This is done in the spectrogram domain, and we use a pretrained MelGAN to invert the generated mel-spectrogram back to a raw waveform. We performed experiments on the AudioMNIST dataset where we try to make the gender private. The quantitative results as well as auditory inspection of the transformed samples show that we succeed with this, but to a loss of utility in the resulting recording. Further experiments should be conducted on more complex datasets with longer sentences and more speakers to evaluate possible applicability to real world data.

Bibliography

[1] Sören Becker, Marcel Ackermann, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Interpreting and explaining deep neural networks for classification of audio signals. *CoRR*, abs/1807.03418, 2018.

[2] Martin Bertran, Natalia Martinez, Afroditi Papadaki, Qiang Qiu, Miguel, Galen Reeves Rodrigues, and Guillermo Sapiro. Adversarially learned representations for information obfuscation and inference. *In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, Proceedings of the 36th International Conference on Machine Learning, volume 97 of Proceedings of Machine Learning Research, pages 614–623, Long Beach, California, USA*, June 2019. PMLR.

[3] Mikolaj Binkowski, Jeff Donahue, Sander Dieleman, Aidan Clark, Erich Elsen, Norman Casagrande, Luis C. Cobo, and Karen Simonyan. High fidelity speech synthesis with adversarial networks, 2019.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[5] Ali Borji. Pros and cons of gan evaluation measures, 2018.

[6] J. Brownlee. *Generative Adversarial Networks with Python: Deep Learning Generative Models for Image Synthesis and Image Translation*. Machine Learning Mastery, 2019.

[7] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. 2018.

[8] Harrison Edwards and Amos J. Storkey. Censoring representations with an adversary. *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[9] Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis, 2019.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[12] D. Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984.

[13] Fredric J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform, 1978.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[15] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2017.

[16] Chong Huang, Peter Kairouz, Xiao Chen, Lalitha Sankar, and Ram Rajagopal. Generative adversarial privacy. 2018.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[18] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[21] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, and Aaron Courville. Melgan: Generative adversarial networks for conditional waveform synthesis, 2019.

[22] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300, 2015.

[23] Jae Hyun Lim and Jong Chul Ye. Geometric gan, 2017.

[24] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization, 2018.

[25] John Martinsson, Edvin Listo Zec, Daniel Gillblad, and Olof Mogren. Adversarial representation learning for synthetic replacement of sensitive data. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (under review)*, June 2020.

[26] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[27] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[28] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.

[29] Marco Pasini. Melgan-vc: Voice conversion and audio style transfer on arbitrarily long samples using spectrograms, 2019.

[30] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[31] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[32] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[33] Brij Mohan Lal Srivastava, Aurélien Bellet, Marc Tommasi, and Emmanuel Vincent. Privacy-preserving adversarial representation learning in asr: Reality or illusion? *Interspeech 2019*, Sep 2019.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[35] Yan Wu, Jeff Donahue, David Balduzzi, Karen Simonyan, and Timothy Lillicrap. Logan: Latent optimisation for generative adversarial networks, 2019.