Master Thesis

University of Bergen

Department of Informatics

Neural Networks for Lossy Weakly-Private Information Retrieval

Author:
Christopher Hærem

Supervisors:
Hsuan-Yin Lin
Eirik Rosnes
Yauhen Yakimenka

June 1, 2021

Acknowledgements

I want to start by expressing my gratitude towards my supervisors, Hsuan-Yin Lin, Eirik Rosnes, and Yauhen Yakimenka, for the guidance, mentoring, and discussions. Thank you for the opportunity to write this thesis and for sharing your expertise with me. I would like to thank Chung-Wei Weng for answering all my questions and for sharing invaluable insight. I also want to thank my wonderful family and friends for all their support and encouragement.

Lastly, I would like to give a special thanks to my flatmates and study partners throughout my time in Bergen, Jonas Mossin Wagle, Henrik Libeck Hexeberg, Håkon Osland, Anders Mølster Hopland, and Sondre Eide Omland, for all the wonderful memories which I will cherish for the rest of my life.

Abstract

The availability of information through public accessible databases has never been greater, but do raise privacy concerns. Private information retrieval schemes guarantees full privacy regarding the servers ability to infer what the user retrieved, but are highly unpractical for single server purposes since the only scheme is downloading the entire database. By allowing some leakage regarding what file the user retrieved to the server, and some distortion between the original and received file, Lossy Weakly-Private Information Retrieval (LWPIR) schemes manage to improve the download rate while still preserving some degree of privacy. Optimal LWPIR schemes are found by numerically solving a constrained optimization problem given the distribution of the data. However, when the distribution is unknown it is interesting to consider a data-driven approach leveraging recent advancements for Generative Adverserial Nets (GANs). GANs have proven useful for similar applications such as in Generative Adversarial Privacy (GAP). This thesis explores this opportunity, first by implementing the GAP model and validating its results, and then secondly by implementing a new LWPIR model with the goal of finding LWPIR schemes. Achieved results for the GAP and LWPIR model are plotted against the theoretical optimal ones with the conclusion that, even though training of the models is challenging, there is great potential in using neural networks for LWPIR.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation

Publicly accessible databases have become an essential resource for accessing up-to-date information. The ever-increasing availability of information has however also raised concerns regarding the privacy of the user. It has evolved a growing financial incentive for businesses to collect and store information pertaining to the online activity of users. Information like user requests is used to extract intent and behavioral patterns about both the online and offline presence of the user, commonly regarded as private information. This private information may then be used in a wide range of applications, both virtuous (e.g. improving the user experience) and nefarious (e.g. unwanted user tracking of sensitive information like race, gender, political affiliation, etc.).

Several attempts have been made to better preserve user privacy, such as private information retrieval (PIR). PIR schemes have the goal of allowing a user to retrieve information from a database, such as a file, without disclosing what was retrieved to the server. For a single server database the only PIR scheme guaranteeing full privacy is the one where the user download the entire database. This is however highly inefficient when the database grows larger. Therefore, it has been proposed extensions of PIR relaxing the condition of full privacy to improve the rate, called weakly-private information retrieval (WPIR), and relaxing the condition of perfect retrievability, called lossy weakly-private information retrieval (LWPIR). By allowing for some leakage regarding what file the user retrieved to the server, and some distortion between the original and received file, LWPIR schemes manage to improve the download rate while still perceiving some degree of privacy. The optimal LWPIR scheme is then a trifold tradeoff between leakage, distortion, and download rate, which can be expressed as a constrained optimization problem.

Finding the optimal LWPIR scheme can be done by numerically solving the constrained optimization problem given the distribution of the data stored on the server. However, for the case where the data distribution is unknown, it is interesting to consider a data-driven approach, where the optimal LWPIR scheme is learned directly from the data. Recent advancements in the neural network architecture Generative Adversarial Nets(GAN) has the potential to solve such constrained optimization problem formulated as a mini-max game, with many promising applications in similar scenarios such as the Generative Adversarial Privacy (GAP).

1.2 Objective

The main objective of this thesis is to investigate the use of neural networks for Lossy Weakly-Private Information Retrieval. The thesis examines previous work done within the field. First, by researching the related privacy perceiving framework "Context-Aware Generative Adversarial Privacy", and validate its results by implementing and testing the described model. Additionally, the thesis will explore a new generative adversarial model named "LWPIR GAN", with ambitions of finding Lossy Weakly-Private Information Retrieval schemes for given constraints concerning leakage, distortion, and rate.

1.3 Thesis Organization

- **Chapter 2:** Overview of relevant key concepts from Information Theory and Neural Networks
- **Chapter 3:** Overview of relevant related work
- **Chapter 4:** Explanation of methodology and model architecture
- **Chapter 5:** Detailed information of the results and findings
- **Chapter 6:** Conclusion regarding results obtained, and suggested future work
- **Chapter 7:** Appendix

Nomenclature

X	Random variables denoted by capital italic letters
$\boldsymbol{x}$	Vectors denoted by bold italic font, thus $\boldsymbol{X}$ is a random vector
$\mathcal{X}$	Sets denoted by calligraphic capital letters
$\lvert\mathcal{X}\rvert$	Size of the set $\mathcal{X}$
$\Pr[X = x]$	Probability for the event "$X = x$"
$P_X(x)$	Probability distribution for random variable X
$\mathbb{E}_X[\cdot]$	Expectation with respect to X
$\mathbb{R}$	Real numbers
$\mathsf{H}(X)$	Entropy fro random variable X
$\mathsf{I}(X;Y)$	Mutual information between X, Y
θ_{p}	Neural network parameters denoted by theta with subscript for model name (p for privatizer)
PIR	Private Information Retrieval
WPIR	Weakly-Private Information Retrieval
LWPIR	Lossy Weakly-Private Information Retrieval
GANs	Generative Adverserial Nets
GAP	Generative Adverserial Privacy
CNN	Convolutional Neural Network
MAP	Maximum a Posteriori
$\nabla_x f$	Gradient of a function f with respect to x

Chapter 2

Background

This chapter introduces fundamental concepts within the field of information theory and neural networks, relevant for this thesis. The explanations are meant to give a concise introduction to relevant concepts in order to further explore the neural network models presented in Chapter 4, with their associated notions from information theory.

2.1 Information Theory

This section gives an introduction to fundamental concepts of information theory and lay the ground work to further explore the field of private information retrieval. If not otherwise mentioned, the definitions and expressions are taken from [3].

2.1.1 Entropy

In information theory, entropy (also referred to as Shannon entropy) measures "information content" or uncertainty of a random variable, which means a way of measuring the spread of a distribution for a random variable.

Definition 2.1.1. For a discrete random variable X, with possible outcomes in the alphabet $x_1, ..., x_n$, which occur with probability $P_X(x_1), ..., P_X(x_n)$, the entropy of X is defined as

$$\mathsf{H}(X) = \sum_{i=1}^{n} P_X(x_i) \log(1/P_X(x_i)). \tag{2.1}$$

It is also possible to interpreter entropy as the expected value of $\log(1/P_X(x))$ giving

$$\mathsf{H}(X) = \mathbb{E}_X[\log(1/P_X(x))]. \tag{2.2}$$

It is worth noting that for the case where $P_X(x_i) = 0$ it can then be assumed that $PX(x_i)\log(P_X(x_i)) = 0$, since $\lim_{x\to 0} x\log(x) = 0$. It follows that entropy never can be negative

$$\mathsf{H}(X) \geq 0, \tag{2.3}$$

and if the occurrence probabilities are uniformly distributed ($P_X(x_1) = 1/n, \forall x \in X$) the following property holds

$$\mathsf{H}(X) = \sum_{x}^{n} 1/n \log(n) = \log(n). \tag{2.4}$$

2.1.2 Joint Entropy

Extending on the definition of entropy for one variable, joint entropy describes the case for two variables by using the joint distribution.

Definition 2.1.2. For the discrete random variables X, Y with possible outcomes in the alphabet $\mathcal{X}, \mathcal{Y}$ following the joint probability mass function $P_{X,Y}(x, y)$, the joint entropy of X, Y is defined as

$$\mathsf{H}(X,Y) = -\sum_{x}\sum_{y} P_{X,Y}(x,y)\log(P_{X,Y}(x,y)) = \mathbb{E}_{X,Y}[\log(P_{X,Y}(x,y))]. \tag{2.5}$$

2.1.3 Conditional Entropy

Conditional entropy describes the amount of uncertainty remaining about a discrete random variable Y given X. It can also be interpreted as the number of bits needed to describe Y when X is known.

Definition 2.1.3.

$$\mathsf{H}(Y \mid X) = \sum_{x} P_X(x)\mathsf{H}(Y \mid X = x) = -\sum_{x}\sum_{y} P_X(x)P_{Y|X}(y \mid x)\log(P_{Y|X}(y \mid x)). \tag{2.6}$$

Conditional entropy can also be interpreted as the expected value of the entropy of the conditional distribution over the conditioned random variable

$$\begin{aligned} &-\sum_{x}\sum_{y} P_X(x)P_{Y|X}(y \mid x)\log(P_{Y|X}(y \mid x)) && (2.7)\\ &= -\sum_{x}\sum_{y} P_{X,Y}(x,y)\log(P_{Y|X}(y \mid x)) && (2.8)\\ &= \mathbb{E}_{X,Y}[\log(P_{Y|X}(y \mid x))]. && (2.9)\end{aligned}$$

Using the chain rule allows for the following

$$\begin{aligned} &\mathsf{H}(Y,X) = \mathsf{H}(Y) + \mathsf{H}(X \mid Y) = \mathsf{H}(X) + \mathsf{H}(Y \mid X) && (2.10)\\ &= -\mathbb{E}_{X,Y}[log(P_{X|Y}(x \mid y))] - \mathbb{E}_{X,Y}[log(P_Y(y))] = -\mathbb{E}_{X,Y}[log(P_{X,Y}(x,y))]. && (2.11)\end{aligned}$$

Further deduction gives the relation

$$\mathsf{H}(X_1, X_2, ..., X_n) = \mathsf{H}(X_1) + \mathsf{H}(X_2 \mid X_1) + ... + \mathsf{H}(X_n \mid X_1, X_2, ..., X_{n-1}). \quad (2.12)$$

It should be pointed out that conditional entropy is not necessarily symmetric

$$\mathsf{H}(Y \mid X) \neq \mathsf{H}(X \mid Y). \quad (2.13)$$

2.1.4 Mutual Information

Mutual information is, as the name implies, a measure of the mutual information between two random variables. Meaning a measure of the mutual dependency between two variables. This can be interpreted as the amount of information (in units such as bits) that can be obtained about a random variable from knowing another random variable.

Definition 2.1.4. For the discrete random variables X, Y, with the joint probability mass function $P_{X,Y}(x, y)$ and marginal probability mass functions $P_X(x)$ and $P_Y(y)$, the mutual information between X, Y is defines as

$$\mathsf{I}(X;Y) = \sum_x \sum_y P_{X,Y}(x,y) \log \frac{P_{X,Y}(x,y)}{P_X(x)P_Y(y)} = \mathbb{E}_{X,Y}\left[\log \frac{P_{X,Y}(X,Y)}{P_X(X)P_Y(Y)}\right]. \quad (2.14)$$

In terms of entropy mutual information has the following relation

$$I(X;Y) = \sum_x \sum_y P_{X,Y}(x,y) \log \frac{P_{X,Y}(x,y)}{P_X(x)P_Y(y)} = \sum_x \sum_y P_{X,Y}(x,y) \log \frac{P_{X|Y}(x \mid y)}{P_X(x)} \quad (2.15)$$

$$= -\sum_x \sum_y P_{X,Y}(x,y) \log(P_X(x)) + \sum_x \sum_y P_{X,Y}(x,y) \log(P_{X|Y}(x \mid y)) \quad (2.16)$$

$$= \mathsf{H}(X) - \mathsf{H}(X \mid Y) = \mathsf{H}(Y) - \mathsf{H}(Y \mid X) \quad (2.17)$$

$$= \mathsf{H}(X) + \mathsf{H}(Y) - \mathsf{H}(X,Y). \quad (2.18)$$

Other properties of mutual information are:

1. Mutual Information can never be negative, and is 0 only when X and Y are independent

$$\mathsf{I}(X;Y) \geq 0. \quad (2.19)$$

2. Conditioning can on average only reduce the uncertainty about one variable

$$\mathsf{H}(X \mid Y) \leq \mathsf{H}(X). \quad (2.20)$$

2.1.5 Hamming Distance

Hamming distance, or hamming distortion, is a measure used to define the distance between binary vectors. For two equal length binary vectors the hamming distance corresponds to the number of positions where the symbols are different. If the vectors are of length 1 the hamming distance is defined as

Definition 2.1.5. For the binary vectors $x, \hat{x} \in \{0, 1\}$, the hamming distance is

$$d(\hat{x}, x) = \begin{cases} 1, & \text{if } \hat{x} \neq x \\ 0, & \text{if } \hat{x} = x. \end{cases} \tag{2.21}$$

2.1.6 Quantitative Information Flow

Quantitative Information Flow [4] is a paper discussing a quantitative theory of information flow. This has become of great interest in a variety of scenarios such as secure information flow, anonymity protocols, and side-channel analysis. A general consensus for theories of quantitative information flow is that they should be based on concepts regarding Shannon entropy and mutual information. However, a useful theory of quantitative information flow must also provide security guarantees in an operational setting.

Classically approaches to the problem of protecting the confidentiality of sensitive information involve seeking to enforce *noninterference*, meaning low inputs should be independent of high inputs. This implies that an adversary would be unable to deduce high inputs from low inputs. In practice however, achieving *noninterference* is typically not possible, since in many contexts, it is necessary to reveal some information that is dependent on high inputs. Examples of this are election protocols, where individual votes should be confidential while naturally, the tally of votes should be public. Another example is password checkers, which must reject incorrect passwords and therefore indirectly reveal information regarding the actual password. The framework presented in [4] aims to solve this problem by tolerating a quantifiable amount of leakage while still providing guarantees regarding privacy.

The paper discusses a (probabilistic or deterministic) program $\mathtt{c}$ that receives some high input $\mathtt{H}$ and produces some low input $\mathtt{L}$. An observing adversary $\mathtt{A}$ may then deduce something about $\mathtt{H}$ from $\mathtt{L}$. The goal is to quantify the amount of information in $\mathtt{H}$ (the initial uncertainty of $\mathtt{A}$), how much information is leaked to $\mathtt{L}$, and the unleaked information about $\mathtt{H}$ (the remaining uncertainty of $\mathtt{A}$). This can intuitively be formulated as

$$\textit{initial uncertainty} = \textit{information leaked} + \textit{remaining uncertainty}. \tag{2.22}$$

In this context for probabilistic programs the amount of information leaked corresponds to the mutual information $\mathsf{I}(\mathtt{H};\mathtt{L})$, the initial uncertainty is the entropy $\mathsf{H}(\mathtt{H})$, and the remaining uncertainty is the conditional entropy $\mathsf{H}(\mathtt{H} \mid \mathtt{L})$. If the program is deterministic $\mathsf{H}(\mathtt{L} \mid \mathtt{H}) = 0$, the following relation emerges

$$\mathsf{I}(\mathtt{H};\mathtt{L}) = \mathsf{I}(\mathtt{L};\mathtt{H}) = \mathsf{H}(\mathtt{H}) - \mathsf{H}(\mathtt{L} \mid \mathtt{H}) = \mathsf{H}(\mathtt{L}). \tag{2.23}$$

One justification for using the conditional entropy $\mathsf{H}(\mathtt{H} \mid \mathtt{L})$ as the measure for remaining uncertainty, is the bound of the *guessing entropy* $G(\mathtt{H} \mid \mathtt{L})$. $G(\mathtt{H} \mid \mathtt{L})$

expresses the expected number of guesses required to guess $\mathtt{H}$ given $\mathtt{L}$

$$G(\mathtt{H} \mid \mathtt{L}) \geq 2^{\mathsf{H}(\mathtt{H}|\mathtt{L})-2} + 1, \tag{2.24}$$

for $\mathsf{H}(\mathtt{H} \mid \mathtt{L}) \geq 2$. However, the adversaries expected number of guesses is not necessarily an adequate measurement of the threat to $\mathtt{H}$. The paper argues that even for large $G(\mathtt{H} \mid \mathtt{L})$, the probability of an adversary guessing $\mathtt{H}$ in one try can be significant. Classically the *Fano's inequality* addresses this by giving lower bounds, in term of $\mathsf{H}(\mathtt{H} \mid \mathtt{L})$, on the probability of $\mathtt{A}$ failing to guess $\mathtt{H}$ given $\mathtt{L}$. Unfortunately the bound given by the *Fano's inequality* is very weak for many cases, understating the actual probability of error.

As a solution, the paper proposes to define a measure for the remaining uncertainty directly in terms of the desired security guarantees, rather than inventing a new measure and then try to prove good security guarantees. The proposed measure is *vulnerability*, defined as

Definition 2.1.6. Given the random variable $X \in \mathcal{X}$, the vulnerability $V(X)$ is

$$V(X) = \max_{x \in \mathcal{X}} \Pr[X = x]. \tag{2.25}$$

Mapping the vulnerability to an entropy measure by $\log \frac{1}{V(X)}$, gives the *min-entropy* as

Definition 2.1.7. The min-entropy of X, denoted $\mathsf{H}_\infty(X)$, is given by

$$\mathsf{H}_\infty(X) = \log \frac{1}{V(X)}. \tag{2.26}$$

It is proposed to use H_∞ as a measure of the initial uncertainty, and to measure the remaining uncertainty thought *conditional vulnerability*

Definition 2.1.8. Given jointly distributed variables X and Y, the conditional vulnerability $V(X \mid Y)$ is

$$V(X \mid Y) = \sum_{y \in \mathcal{Y}} \Pr[Y = y] V(X \mid Y = y) \tag{2.27}$$

where

$$V(X \mid Y = y) = \max_{x \in \mathcal{X}} \Pr[X = x \mid Y = y], \tag{2.28}$$

leading to the conditional min-entropy $\mathsf{H}_\infty(X \mid Y)$

$$\mathsf{H}_\infty(X \mid Y) = \log \frac{1}{V(X \mid Y)}. \tag{2.29}$$

This definition of vulnerability gives the immediate security guarantee $V(\mathtt{H} \mid \mathtt{L}) = 2^{-\mathsf{H}_\infty(\mathtt{H}|\mathtt{L})}$, meaning the expected probability of an adversary correctly guessing $\mathtt{H}$ given $\mathtt{L}$ decreases exponentially with $\mathsf{H}_\infty(\mathtt{H} \mid \mathtt{L})$. It is also demonstrated in [4] that for various deterministic and probabilistic programs, the new definition of vulnerability and min-entropy more precisely access an adversaries ability to infer $\mathtt{H}$ in one try, compared to using Shannon entropy or mutual information. The LWPIR schemes, on which the model presented in Section 4.2 is built, has privacy metrics concerning maximal leakage which was derived using the above mentioned definitions. This will further be explained in Section 3.2, introducing the concept of private information retrieval and extensions of it.

2.2 Neural Networks

This section explains fundamental concepts of machine learning and gives an introduction to neural networks, explores some of the most predominant architectures, and finally describes the neural network framework, "Generative Adversarial Nets", the framework used for building the models presented in Chapter 4. If not otherwise mentioned, the following expressions and examples are taken from [5].

2.2.1 Introduction to Machine Learning and Neural Networks

Machine learning algorithms have risen as a prominent tool used in a wide variety of applications. The core objective of machine learning is to create an algorithm capable of generalizing from experience to solve a task, meaning an algorithm that is able to learn from past experience to perform accurately on new unseen examples/tasks. Generally, this means an algorithm that takes in training examples from some, usually unknown, probability distribution and uses it to build a general model about the space in order to perform accurate predictions on new cases. The best performing machine learning algorithm in the context of generalization is one where the complexity of the model space matches the complexity of the function underlying the data. If the complexity is too low, the model will not be accurate since the model does not capture all the complexity of the underlying data, called "under fitting". While if the complexity is too high the model has "over fitted" the data and will have made bad generalizations also resulting in poor accuracy.

Machine learning is traditionally divided into three categories, according to the approach of the learning:

- Supervised learning: The model is trained on labeled data where input is paired with desired output, with the goal of finding general rules that map inputs to output.
- Unsupervised learning: The model is trained on unlabeled data, having to find structure in the data without predefined labeled input-output pairs.
- Reinforcement learning: The model is trained by interacting with a dynamic environment with a specified goal. Examples of reinforcement learning are driving a car or playing chess. The reinforcement is happening as the model navigates the problem space by providing feedback interpreted as rewards, which it tries to maximize (avoiding collisions/capturing opponent chess pieces).

Typical tasks within machine learning are regression and classification tasks. Regression are tasks of predicting a continuous quantity (age, weight, height etc.), while classification are tasks of predicting a discrete class label (color, country, animal etc.). The most basic models are based on linear and logistic regression, using linear combinations of fixed basis functions. These have proven useful for their analytical and computational properties, but have lacked the power to express more complex structures due to their inherent linearity. It has therefore been an ongoing challenge to develop more sophisticated models that adapt the basis functions to the data. One approach is support vector machines (SVMs), which first define a large set of basis functions centered around the data, and then selects a subset of those vectors (support vectors) during training. The goal of SVMs is to find a hyperplane in an N-dimensional space (where N is the number of labels) that distinctly differentiates the data points.

In more recent years another approach, neural networks, has experienced significant development. Neural networks, like SVMs, is built on the concept of basis functions, but uses parametric forms of the functions that allows the parameter values to evolve during training. The first and simplest form of neural network is the feedforward neural network. The feedforward neural network can be interpreted as stacked layers of logistic regression models, and is usually faster to train than SVMs with the same generalization performance. In contrast with SVM, the maximum likelihood function, that is the function describing the parameter values best estimating the real distribution, is for feedforward neural networks no longer convex. This means that it is not possible to guarantee an optimal solution, however in practice feedforward neural networks often finds a minimum that is acceptable. The next section will further explore the feedforward neural network by extending on the definition of linear and logistic regression to form a definition for the baseline feedforward network.

2.2.2 Feedforward Neural Network

Before defining feedforward neural networks, it is helpful to look at linear and logistic regression which use a fixed set of basis functions, denoted as $\phi_j(\boldsymbol{x})$, multiplied with corresponding weights w_j, to produce outputs

$$y(\boldsymbol{x}, \boldsymbol{w}) = h\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right). \tag{2.30}$$

Here $h(\cdot)$ is identity for regression tasks or a non-linear activation for classification tasks. The next step in order to use this definition to describe neural networks is to express the basis functions as parametric functions with parameter values that can adapt during training. This is done by picking basis functions on the form (Equation 2.30), where each basis is itself a non-linear function of the linear combination of inputs.

Starting from prior to applying the activation function $h(\cdot)$, it is then possible to group the inputs in M linear combinations and define the most basic neural network consisting of one hidden layer (denoted by superscript 1) as

$$s_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_{j0}^{(1)}, \qquad j = 1, \ldots, \mathrm{M}. \tag{2.31}$$

The resulting s_j is then passed to the activation function $h(\cdot)$,

$$z_j = h(s_j) \tag{2.32}$$

to obtain the so called "hidden units", z_j, corresponding to the output from the basis function (2.30). The hidden units are then applied with another linear combination

$$s_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_{k0}^{(2)}, \qquad k = 1, \ldots, K \tag{2.33}$$

resulting in the unit activation s_k, which is then finally transformed by an appropriate activation function to give the outputs y_k. The type of activation function depends on if it is a regression or a classification problem. If the problem is a regression problem the activation function can be the identity $y_k = s_k$. For binary classification problems the activation function transforms each unit output into a probability using a logistic *Sigmoid* function

$$y_k = \sigma(s_k) = \frac{1}{1 + e^{(-s_k)}}. \tag{2.34}$$

Multi-class classification problems uses a Softmax activation function which transforms the output into a probability vector

$$\boldsymbol{y}_k = \text{Softmax}(s_k) = \frac{e^{s_k}}{\sum_k e^{s_k}}. \tag{2.35}$$

Nesting the hidden units inside the activation function (assumed to be Sigmoid), results in the following output

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_{k0}^{(2)}\right) \tag{2.36}$$

$$= \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_j + b_{j0}^{(1)}\right) + b_{k0}^{(2)}\right). \tag{2.37}$$

The equation (Equation 2.36) expresses a forward pass through the feedforward neural network which can be used sequentially for each of the hidden layers in deeper networks (networks with more than 1 hidden layer). The model has now achieved the initial goal of transforming the inputs x_i to the outputs y_k through a non-linear function parameterized by weights and biases encapsulated in the vector $\boldsymbol{w}$. The notation can further be simplified by defining a constant $x_0 = 1$ as input, making the input sum

$$s_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i, \qquad j = 1, \ldots, M \tag{2.38}$$

such that the complete model simplifies to

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right). \tag{2.39}$$

Feedforward neural networks have received much attention due to their strong approximation properties. It has been shown that simple networks with as few as one hidden layer are capable of approximating any continuous function on a compact input domain as long as it is provided sufficiently many hidden units [6]. Hence, the primary challenge of feedforward networks is to determine fitting parameter values based on a set of training data. It is therefore essential to have good optimization algorithms. One important characteristic of feedforward neural networks is that they are differentiable with respect to their parameters. This is due to the fact that the activation functions used are smooth and differentiable, making the entire model differentiable. Using differentiable activation functions enables the use of the very efficient optimization method gradient descent, which is introduced in the next subsection about optimization in neural networks.

2.2.3 Optimization in Neural Networks

Optimization in neural networks is the operations done to update the parameter weights with the goal of determining fitting weights based on a set of training data. Given a neural network $y = f(\boldsymbol{x}, \boldsymbol{w})$, a set of training data and the target function $E(\boldsymbol{w})$, it is usually not possible to determine a global optimum. This is because of the fact that the models are non-convex, meaning it is not possible to find an analytic solution for $\nabla E(\boldsymbol{w}) = 0$. A optimization function able to optimize over continuous nonlinear functions is therefore needed, where the most prominent proposals revolve around the use of gradient decent. Starting with an initial value $\boldsymbol{w}_0$, gradient decent iteratively updates the preceding weight $\boldsymbol{w}_{t+1}$ as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \Delta \boldsymbol{w}_t \tag{2.40}$$

where $\Delta \boldsymbol{w}_t$ is the weight vector update, calculated using some optimization algorithm. The optimization algorithm involves finding the gradient of the target function $\nabla E(\boldsymbol{w})$, concerning specific input values to the function. Calculating the gradient of parameters in the hidden layers of the network requires error backpropagation, which passes the gradient from the last layer to the first using the chain rule. Gradient descent and error backpropagation is further explored in the following two subsections.

2.2.4 Gradient Descent and Extensions

The most popular and simplest optimization algorithm for neural network is gradient descent. Gradient descent works by repeatedly taking small steps in the opposite direction of the gradient (or approximated gradient), gradually moving toward the local minimum which, can be described as

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma \nabla E(\boldsymbol{w}_t), \qquad \gamma > 0 \tag{2.41}$$

where γ is the step size (also called learning rate). The evaluation of $\nabla E(\boldsymbol{w}_t$ is traditionally done for each time step on the entire dataset, called *batch gradient decent*. *Batch gradient decent* is however in general highly inefficient for larger datasets due to the high computational cost of calculating the full gradient. This motivated the development of alternate variants of gradient decent which differ in the amount of data used to calculate the gradient, resulting in a trade-off between the accuracy of the parameter update and the cost of calculating the gradient. One approach is *stochastic gradient decent*, or SGD, which in contrast to *batch gradient decent* evaluates the gradient of the loss function for one data point and performs an update to the parameters for each data point in the dataset

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma \nabla E_n(\boldsymbol{w}_t). \tag{2.42}$$

Here the error $E(\boldsymbol{w})$ function is comprised of the sum of errors for each data point

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}). \tag{2.43}$$

There also exists other variants of gradient decent, such as *mini-batch gradient decent*, which takes an intermediate approach where the data is split into smaller mini-batches. Here the gradient is calculated as an average for every mini-batch and then the update is performed to the parameter weights.

Using SGD or *mini-batch gradient decent* has shown to have many advantages over traditional *batch gradient decent.* Firstly, by not calculating the full gradient on the entire dataset, the training generally becomes much faster and more robust to redundant data. Secondly, SGD (and *mini-batch gradient decent* to some extent) can also experience higher variance during the training, which can have the positive ability of being able to escape local minima during training to possibly find new better local minima. Higher variance during training can however also introduce the challenge of overshooting and oscillation, where the model struggles to converge to an exact minimum. There have been numerous extensions made to address this problem such as using a varying step size, called *momentum*,

$$\nu_t = \eta \nu_t + \gamma E_n(\boldsymbol{w}_t), \tag{2.44}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \nu_t. \tag{2.45}$$

Momentum works by using a *momentum coefficient*, η, to add a fraction of the past update vector to current update. Here ν_t is usually initialized at 0, and η to a value slightly less than 1 like 0.9. The effect of momentum is analogous to the movement of a ball down a hill, where the ball accelerates down the hill until it reaches terminal velocity ($\eta < 1$). Using momentum reduces the oscillation and can make the training converge faster, however overshoot is still a possibility. It is therefore common practice to gradually lower the step size, such as initializing η to 0.5 and gradually anneal it to 0.9.

Another method addressing the problem of oscillation and overshooting is *Root Mean Squared Propagation* (RMSProp). RMSProp adapts the learning rate for each parameter by normalizing the gradient according to the magnitude of the recent gradients. The update for each parameter w_j in $\boldsymbol{w}$ can be expressed as

$$\nu_t = \rho \nu_{t-1} + (1-\rho) \cdot g_t^2, \tag{2.46}$$

$$\Delta \boldsymbol{w}_t = -\frac{\gamma}{\sqrt{\nu_t + \epsilon}} \cdot g_t, \tag{2.47}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta \boldsymbol{w}_t. \tag{2.48}$$

Here γ is the initial learning rate, ν_t is an exponential average of the past gradients squared and g_t is the gradient at time step t along w_j. The first equation calculates an exponential average of the square of the gradient. This is done by multiplying the exponential average gradient, calculated for the gradient up to the last update, with the hyperparameter ρ. Then the square of the current gradient is multiplied with $(1-\rho)$. Finally they are added together, resulting in the exponential average up to the current time step, ν_t. Using exponential average forces the update to give more weight to the recent gradient update, causing weights of the previous terms to fall exponentially. The second equation determines the step size, by normalizing the initial learning rate γ with the exponential average squared. The third equation is the actual weight update. Generally the hyperparameter ρ is set to 0.9, and the ϵ to a small number such as 1e-10 to ensure no zero division. A significant advantage of RMSProp over momentum is its inherent annealing property; Continuing the analogy of the ball down the hill, RMSProp will gradually slow down when approaching the bottom reducing the risk of overshooting. Since when moving towards a minima the exponential average of past gradients ν_t will become large, reducing the step size.

A final variant of gradient to be discussed in this thesis is the Adaptive Moment Estimation (Adam) [7]. Adam was introduced in 2014 as a compromise between the accelerated convergence towards the minima gained by momentum, and the oscillation reduction through normalization of the step size achieved by RMSProp. The update equations for Adam are

$$\nu_t = \beta_1 \cdot \nu_{t-1} + (1-\beta_1) \cdot g_t, \tag{2.49}$$

$$s_t = \beta_2 \cdot s_{t-1} + (1-\beta_2) \cdot g_t^2, \tag{2.50}$$

$$\Delta \boldsymbol{w}_t = -\gamma \frac{\nu_t}{\sqrt{s_t + \epsilon}} \cdot g_t, \tag{2.51}$$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta \boldsymbol{w}_t. \tag{2.52}$$

Here ν_t is an exponential average of the gradient and s_t is the exponential average of the past gradients squared for each parameter $\boldsymbol{w}^j$, with hyperparameters β_1, β_2 analogous to the ρ in RMSProp. The third equation determines the step size by multiplying the learning rate with the average of the gradient (similar to momentum) and then dividing it by the root mean square of the exponential average of the past gradients (similar to RMSProp). Then finally applying the update in the fourth equation. Generally the hyperparameters β_1 and β_2 are chosen to be around 0.9 and 0.99, respectively, while epsilon is a small value such as 1e-10 to ensure no zero division. The models presented in this thesis were trained using the Adam optimizer with these hyperparameter values.

2.2.5 Error Backpropagation

The previous section explored some of the various ways to update the weight parameters of a neural network with different optimizers using the gradient of each weight. This section will describe the most prominent algorithm used for calculating the gradients, called *error backpropagation.*

The error backpropagation was first introduced in [8] as an efficient method for computing the derivative of the loss function in regard to the parameter weights utilizing the chain rule. The "backward" part of the name derives from the fact that the gradient is iteratively calculated backwards through the network, starting with the gradient for the weights in the final layer and ending with the gradient for the weights in the first.

To help illustrate error backpropagation, the following equations will walk through each step of the algorithm given a feedforward neural network, such as the one described in Section 2.2.2. The evaluation of $\Delta E_n(\boldsymbol{w})$ is done focusing on one data point in the dataset. Starting with a linear model

$$y_k = \sum_i w_{ki} x_i, \qquad k = 1, \ldots, K \tag{2.53}$$

with mean squared error as the error function

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2, \tag{2.54}$$

where t_{nk} is the label value and y_{nk} is the estimate for one data-point.

The gradient for the weights w_{ji} is

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nk} - t_{nk}) x_{ni}. \tag{2.55}$$

As explained earlier in Section 2.2.2, a weighted sum of inputs for the activation function is computed for each unit in the feedforward network

$$s_j = \sum_i w_{ji} z_i \tag{2.56}$$

where z_i is the activation for the unit which is connected to the unit j with the weight w_{ij}. Here, as in Section 2.2.2, the weights and biases are encapsulated into the inputs and the hidden units, making the activation of the unit j

$$z_j = h(s_j). \tag{2.57}$$

Using these equations it is possible to calculate the activations of all the units through forward propagation, using the input values from the dataset and the weight values. Since the derivative of E_n with respect to the weight w_{ji} is only dependent on w_{ji}, through the sum s_j, it is possible to utilize the chain rule

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial s_j} \frac{\partial s_j}{\partial w_{ji}} = \partial_j \frac{\partial s_j}{\partial w_{ji}}, \tag{2.58}$$

where $\partial_j = {}^{\partial E_n}\!/\!_{\partial w_{ji}}$ is used as notation for the *errors*. Further, define

$$\frac{\partial s_j}{\partial w_{ji}} = z_i \tag{2.59}$$

results in the expression

$$\frac{\partial E_n}{\partial w_{ji}} = \partial_j z_i. \tag{2.60}$$

Finding the derivative of E_n with respect to the weight w_{ji} is then only dependent on the value of ∂_j and the activation z_i. Calculation of ∂_j is therefore of great importance, and is done for each output unit

$$\partial_k = y_k - t_k, \tag{2.61}$$

and for each hidden unit using the chain rule

$$\partial_j = \frac{\partial E_n}{\partial s_j} = \sum_k \frac{\partial E_n}{\partial s_k}\frac{\partial s_k}{\partial s_j}. \tag{2.62}$$

Here the sum is for every unit k which has a connection from the unit j. Given the above equations, the complete formulation of backpropagation becomes

$$\partial_j = \sum_k \frac{\partial E_n}{\partial s_k}\frac{\partial s_k}{\partial s_j} = \sum_k \partial_k \frac{\partial s_k}{\partial s_j} \tag{2.63}$$

$$= \sum_k \partial_k w_{kj} h'(s_j) = h'(s_j) \sum_k w_{kj} \partial_k. \tag{2.64}$$

These equations enable the calculation of the the unit derivatives ∂ throughout the network by propagating the ∂ backwards from the output layer to the input layer. The calculations can be done to find all the derivatives for the units in a feed forward network, regardless of the topology or what activation function is used.

For batch training, because the error is defined as the total errors given for each data point, the derivative is defined as the aggregated total of the error derivatives for each data point in the batch

$$\frac{\partial E}{\partial w_{jk}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}. \tag{2.65}$$

Error backpropagation has since it first introduction become the industry standard within the field of machine learning. Also the model described in this thesis was trained using an implementation of the error backpropagation. The next section will introduce a class of neural networks called Convolutional Neural Network, and then discuss two other architectures, Generative Adversarial Networks and Context-Aware Generative Adversarial Privacy, both important for the design of the model presented in this thesis.

2.2.6 One-hot Encoding

One-hot encoding is a process where categorical variables, or labels, are converted to numerical values. This is often necessary in machine learning to make sure that the data fit the model. Given the four different categorical variables red, blue, green, and yellow, a one-hot encoding for each variable is

- red = [1,0,0,0],
- blue = [0,1,0,0],
- green = [0,0,1,0], and
- yellow = [0,0,0,1].

2.2.7 Convolutional Neural Network

The earlier described feedforward neural networks (Section 2.2.2) are considered to be general, with no restrictions on the data. Feedforward neural networks are however also computationally expensive to train, since they typically grow large. Often there are structure in the data, such as in image data, where it is possible to leverage the knowledge of the structure to improve training. Many attempts have been made to improve the "quality" of the training parameters, one of which is through weight sharing used in Convolutional neural networks (CNNs). First introduced in [9], it was shown that training on hand-written digit data could be improved by sharing the weights. CNNs exploit the structure of data by integrating invariance to certain transformations. For image classification transformations such as scaling or rotation is not important for correct labeling, and the model should therefore have an invariance towards them. Also for images there usually is a closer correlation between pixel values that are close to each other. Several techniques have been developed to exploit relations like these by the use of filters, which extract useful local features and combine them in order to detect higher order features. Different types filters can then be used to extract different features.

CNNs integrate filters (also referred to as kernels) as part of what is known as a convolutional layer. The convolutional layer applies the filter to the input in order to create a feature map, summarizing the presence of the detected feature for the input. The filter size is smaller than the input data, resulting the same filter being applied several times in filter-sized patches across the full input space (called a convolution). The filter is applied to the input as a dot product, where each filter-sized patch of input is multiplied with the filter element-wise and then summed, resulting in a single value. An image illustrating a filter being applied to a two-dimensional input resulting in a feature map can be seen in Figure 2.1.

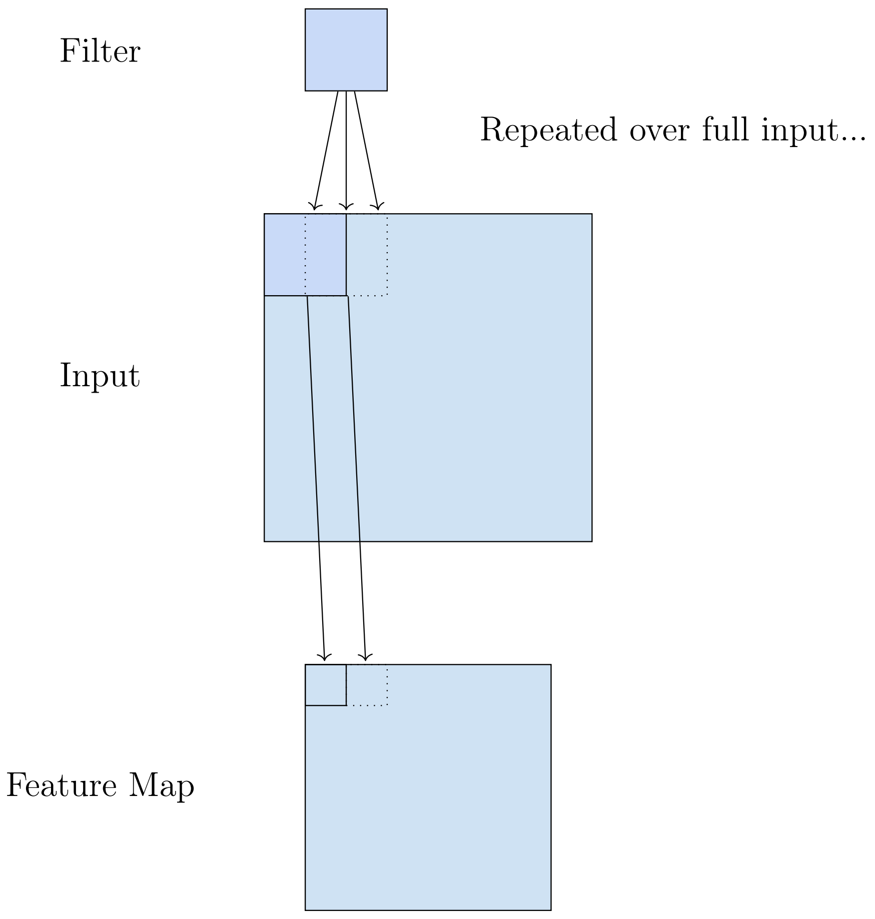


Figure 2.1: A filter being applied to a two-dimensional input to create the feature map

As seen from the Figure 2.1, the convolution usually results in a down-sampling, meaning the output dimensions are less than that of the input. The size of the feature map is determined by the input size i (according to the input dimension $i \times i$), the filter size f, and by two additional parameters called padding p and stride s. Here the padding is an optional number of zeros padded around the original input (set to zero for Figure 2.1), and the stride refers to the amount which the filter is shifted across the input. Given i, f, p, and s, the size of the generated feature map is

$$\frac{i + 2p - f}{s} + 1. \tag{2.66}$$

There is also a reversed form of the standard convolutional layer, called the transposed convolutional layer. While the standard convolutional layer performs a down-sampling, the transposed convolutional layer performs an up-sampling, meaning the output feature map has a greater dimension than the input. The transposed convolutional layer is, like the standard convolutional layer, defined by the padding and stride. Here the padding and stride corresponds to the values that were hypothetically used on the output to generate the input, meaning performing a standard convolution on the output with the defined stride and padding generates an output of the same dimension as the input. The output size for the transposed convolutional layer is given by

$$(i-1)\cdot s+f-2\cdot p. \tag{2.67}$$

The motivation behind the transposed convolutional layer is to, through learning, find a filter that maps the output to the initial input. For example given images data, a standard convolutional layer takes the image as an input and generates a vector, while the transposed convolutional layer takes the vector as input an tries to replicate the image. This type of transposed convolutional layer is used in the *generator* of Generative Adverserial Nets, which are introduced in the following subsection. The transposed convolutional layer was also used in the implementation of both the models presented in Chapter 4.

2.2.8 Generative Adversarial Networks

Generative Adversarial Networks (GANs) is class of machine learning that uses two neural networks competing against each other as the training process, first introduced by Goodfellow et al. [1]. The framework consists of a generative model (the generator), and a discriminating model (the discriminator). The goal of the generative model is to capture the data distribution from input data, and use it to generate new data points, while the goal of the discriminating model is to estimate the probability whether a sample came from the original data or was generated by the generator. The training consists of the discriminator trying to maximize the accuracy of predicting correct origin of the input (either generated or from the real data), while simultaneously the generator is trained to minimize the accuracy of the discriminator. The resulting framework can be described as a minimax two-player game, where the generator (f_{G}) and the discriminator (f_{D}) are competing against each other to respectively minimize and maximize a value function. The generator wants to learn the distribution P_{g} over the input data $\boldsymbol{x}$. The distribution over the real data $\boldsymbol{x}$ is denoted by $P_{\mathrm{data}}(\boldsymbol{x})$. Defining a prior on input noise variable $\boldsymbol{Z}$ with probability distribution $P_Z(\boldsymbol{z})$, the generator is modeled as a the mapping function $f_{\mathrm{G}}(\boldsymbol{z};\theta_{\mathrm{g}})$, where f_{G} represent a neural network with the parameters θ_{g}. The discriminator is also modeled as a neural network $f_{\mathrm{D}}(\boldsymbol{x};\theta_{\mathrm{d}})$, with the parameters θ_{d}. $f_{\mathrm{D}}(\boldsymbol{x};\theta_{\mathrm{d}})$ outputs a scalar $f_{\mathrm{D}}(\boldsymbol{x})$, representing the probability that $\boldsymbol{x}$ is from the the original data rather than P_{g}. The resulting value function $f_{\mathrm{V}}(f_{\mathrm{D}},f_{\mathrm{G}})$ is defined as

$$\min_{f_{\mathrm{G}}}\max_{f_{\mathrm{D}}} f_{\mathrm{V}}(f_{\mathrm{D}},f_{\mathrm{G}})=\mathbb{E}_{\boldsymbol{X}}[\log(f_{\mathrm{D}}(\boldsymbol{X}))]+\mathbb{E}_{\boldsymbol{Z}}[\log(1-f_{\mathrm{D}}(f_{\mathrm{G}}(\boldsymbol{Z})))], \tag{2.68}$$

where $\boldsymbol{X}\sim P_{\mathrm{data}}(\boldsymbol{x})$ and $\boldsymbol{Z}\sim P_{\boldsymbol{Z}}(\boldsymbol{z})$.

Examples of minimax games are chess or tic-tac-toe. The insight of describing the framework as a minimax game is that according to theory there exists an optimal unique solution where neither participant can improve their outcome (Nash equilibrium). It is shown in [1] that given infinite capacity in the discriminator and generator, the optimal discriminator for a given generator is

$$f_{\mathrm{D}}^{*}(\boldsymbol{x})=\frac{P_{\mathrm{data}}(\boldsymbol{x})}{P_{\mathrm{data}}(\boldsymbol{x})+P_{\mathrm{g}}(\boldsymbol{x})}. \tag{2.69}$$

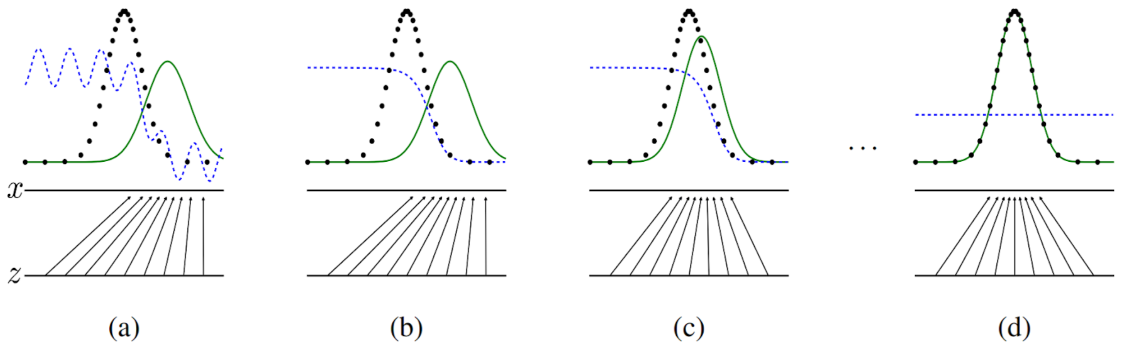
Here the global minimum is achieved at $P_{\mathrm{g}}=P_{\mathrm{data}}$, where the generated samples are indistinguishable from the data. To achieve this optimum it is necessary to alter between training the generator and the discriminator. The proposed training algorithm from [1] is as follows:

Algorithm 1: Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter.

```
for number of training iterations do
    for k steps do
        Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $P_Z(\boldsymbol{z})$.
        Sample minibatch of $m$ noise samples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $P_{\text{data}}(\boldsymbol{x})$.
        Update the discriminator by ascending its stochastic gradient:
            $\nabla_{\theta_\text{d}} \frac{1}{m} \sum_{i=1}^{M} [\log D(\boldsymbol{x}^{(i)}) + \log(1 - D(G(\boldsymbol{z}^{(i)})))]$
    Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, ..., \boldsymbol{z}^{(m)}\}$ from noise prior $P_Z(\boldsymbol{z})$.
    Update the generator by ascending its stochastic gradient:
        $\nabla_{\theta_\text{g}} \frac{1}{m} \sum_{i=1}^{\text{M}} \log(1 - D(G(\boldsymbol{z}^{(i)})))$
```

As seen from the algorithm above the discriminator is trained k times for each training iteration of the generator. Ideally the training will be similar to Figure 2.2, illustrating the general stages of training. The figure consists of the discriminative distribution $f_\text{D}(\boldsymbol{x})$ (blue dashed line), the distribution of the samples obtained from the data generating distribution, denoted by P_x (black dotted line) and the generative distribution P_g (green solid line). The lower horizontal lines represents the domain from where samples $\boldsymbol{z}$ are drawn from (here uniformly), while the arrow pointing to the upper horizontal line represents the mapping of samples $\boldsymbol{z}$ to the domain of $\boldsymbol{x}$, $\boldsymbol{x} = f_\text{G}(\boldsymbol{z})$. Starting from (a) the adversarial pair is near convergence, P_g is similar to P_data, while the discriminator f_D is a moderately accurate classifier. (b) illustrates the scheme after the discriminator f_D has been trained in the inner loop of the algorithm, converging to $f^*_\text{D}(\boldsymbol{x}) = \frac{P_\text{data}(\boldsymbol{x})}{P_\text{data}(\boldsymbol{x}) + P_\text{g}(\boldsymbol{x})}$. Then in (c) an update is made to the generator f_G, guiding $f_\text{G}(\boldsymbol{z})$ to generate samples more likely to be classified as data. Finally, (d) shows the scheme after several training iterations where, if f_G and f_D have had enough capacity, it will have converged to an equilibrium where neither f_G or f_D can improve because $P_\text{g} = P_\text{data}$, resulting in $f_\text{D}(\boldsymbol{x}) = \frac{1}{2}$, since the discriminator now longer is capable of differentiating the two distributions.

Figure 2.2: Illustration of GAN training steps from [1]



GANs have been shown to achieve excellent performance when generating samples from CIFAR-10 [10], MNIST [11] and the Toronto Face Database [12], from [1]. Advantages to GANs include the simplicity of the model, being only dependent on backpropagation during training and allowing for any differentiable function for both the generator and the discriminator increasing the flexibility of the framework. However there are also disadvantages to GANs such as not achieving an explicit representation of $P_{\mathrm{g}}(\boldsymbol{x})$ and the challenge of synchronizing the training of the generator and discriminator. As will be presented in Chapter 4, the model introduced in this thesis was created using a GAN framework due to the promising properties described. There has also been other work utilizing the GAN framework in the context of privacy such as generative adversarial privacy (GAP), which will be discussed in the next chapter on related work, Chapter 3.

2.2.9 Maximum a Posteriori

Maximum a posteriori (MAP) estimation is a method for estimating a distribution and model parameters given an observed dataset.

Definition 2.2.1. Given the observation of the random variable $Y = y$, which is dependent on the variable X, the MAP estimation of X is the value x that maximizes:

- $f_{X|Y}$ when X is a continuous random variable, or
- $P_{X|Y}$ when X is a discrete random variable.

Hence, the MAP estimation for the value x is given by

$$\arg\max_{x\in\mathcal{X}} f_{X|Y}(x \mid y) = \arg\max_{x\in\mathcal{X}} \frac{f_{Y|X}(y \mid x) f_X(x)}{f_Y(y)}. \tag{2.70}$$

Chapter 3

Related work

This chapter explains two concepts closely relating to the models that are introduced in Chapter 4. The first section discusses context-aware generative adversarial privacy, a privacy framework built using a GAN framework. The second section introduces the concept of private information retrieval and extensions on which the model presented in Section 4.2 is built.

3.1 Context-Aware Generative Adversarial Privacy

Context-aware generative adversarial privacy (GAP) is a data-driven privacy perceiving framework inspired by the advancement of GANs, first introduced in [2]. GAP uses a framework similar to the one described in Section 2.2.8, where two neural networks compete against each other in a mini-max game with the goal of learning a privacy mechanism. From the paper the setting is presented as a problem where a data provider wants to publish a dataset $\mathcal{D}$ consisting of (X, Y) pairs, where X is the public variable and Y the private variable. The goal is to find a privatized version of X, noted as $\hat{X}$, which contains many of the original properties of X without revealing information useful for an adversary to infer the private variable Y. The resulting framework consists of a privatizer competing against an adversary. The privatizer has the goal of finding the privacy mapping $\hat{X} = g(X, Y)$ under a distortion constraint. While the goal of the adversary is to infer the private variable Y from the privatised data, $\hat{Y} = h(g(X, Y))$. An illustration of the framework can be seen in Figure 3.1.

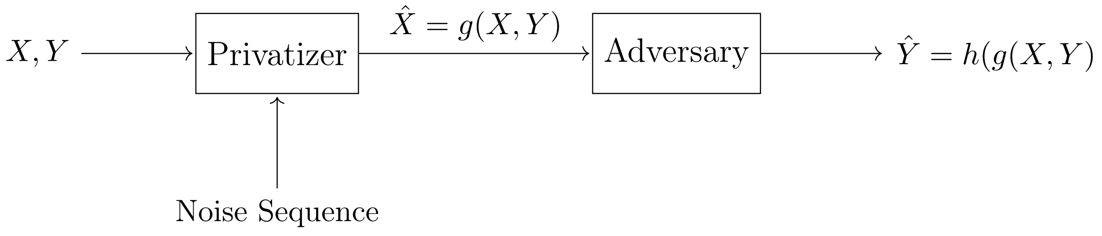


Figure 3.1: Illustration of GAP framework from [2].

The loss function for the framework is defined as the inference accuracy of the adversary, where two different types of problems are presented:

- 0-1 binary loss, resulting in a maximum a posteriori (MAP) adversary (Section 2.2.9),
- empirical log-loss, resulting in a minimum cross-entropy adversary.

The final goal of the framework is to guarantee privacy against a MAP adversary. However, the binary loss function is not differentiable and thus not suitable for data-driven approaches, since training of the adversary becomes infeasible. Therefore the paper instead uses an adversary with a log-loss function, and shows performance matching that of an theoretical optimal MAP adversary. The formal definition of the framework is as follows:

First as mentioned earlier, the privatizer it defined as the randomized mapping

$$g(X, Y) : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}. \tag{3.1}$$

Then given the output $\hat{X} = g(X, Y)$ the adversary becomes

$$\hat{Y} = h(g(X, Y)), \tag{3.2}$$

and the corresponding loss function is modeled as

$$\ell(\hat{Y} = \hat{y}, Y = y) = \ell(h(g(X = x, Y = y)), Y = y) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}. \tag{3.3}$$

The expected loss of the adversary with respect to X and Y is defined to be

$$f_{\text{Loss}}(h, g) \triangleq \mathbb{E}_{X,Y}[\ell(h(g(X, Y)), Y)], \tag{3.4}$$

where the expectation is over $P_{X,Y}$ and the randomness in g and h. Intuitively the trivial solution to the problem of minimizing the adversaries ability of inferring Y is to publish $\hat{X}$ which is independent of X. This solution is however of little value, since the data no longer is useful for learning the non-private variables from $\hat{X}$. The proposed solution is therefore to enforce a distortion constraint for the privatizer, limiting the maximum difference between the privatized and original data. The expected distortion based on a distortion function $d(x, \hat{x})$ is

$$\mathbb{E}_{X,Y}[d(g(X, Y), X)] \tag{3.5}$$

with expectation over $P_{X,Y}$ and randomness in g. The result is a mini-max game with a privatizer aiming to be both privacy and utility perceiving, and an adversary trying to minimize its expected loss, or equivalently maximize the negative of the expected loss. This can be formulated as the following

$$\min_{g(\cdot)} \max_{h(\cdot)} -f_{\text{Loss}}(h, g) \tag{3.6}$$

$$\text{s.t.} \quad \mathbb{E}_{X,Y}[d(g(X, Y), X)] \leq \mathsf{D}. \tag{3.7}$$

There are no restrictions on the adversary, with different loss functions and decision rules leading to different adversarial models. Various loss functions are discussed in [2] under both hard and soft decision rules.

For *hard decision rules* the adversary, $\hat{Y} = h(g(X, Y))$, is an estimate of Y. For continuous Y the considered loss function for the adversary is the squared loss

$$\ell(h(g(X, Y), Y) = (h(g(X, Y)) - Y)^2, \tag{3.8}$$

also known as the ℓ_2 loss. Here the adversary's optimal decision rule is $h^* = \mathbb{E}[Y \mid g(X,Y)]$, which is the conditional mean of Y given $\hat{X} = g(X,Y)$. The mini-max game (Equation 3.6) then simplifies to

$$\min_{g(\cdot)} -\{\text{mmse}(Y \mid g(X,Y))\}, \tag{3.9}$$

where mmse stands for the minimum mean square error. For the case when Y is discrete, the adversary can maximize its classification accuracy by considering a 0-1 loss function given by

$$\ell(h(g(X,Y)),Y) = \begin{cases} 0, & \text{if } h(g(X,Y)) = Y \\ 1, & \text{otherwise.} \end{cases} \tag{3.10}$$

Here the adversary's optimal decision rule is the MAP decision rule: $h^* = \arg\max_{y\in\mathcal{Y}} P_{Y|\hat{X}}(y \mid \hat{x})$, and the mini-max game simplifies to

$$\min_{g(\cdot)} -(1 - \max_{y\in\mathcal{Y}} P_{Y|\hat{X}}(y \mid \hat{x}) = \min_{g(\cdot)} \max_{y\in\mathcal{Y}} P_{Y|\hat{X}}(y \mid \hat{x})) - 1. \tag{3.11}$$

For *soft decision rules* the adversary, $h(g(X,Y)) = \hat{Y}$, can be seen as a distribution over $\mathcal{Y}$; i.e. $h(g(X,Y)) = P_{\hat{Y}|\hat{X}}(\hat{y} \mid \hat{x}))$ for $\hat{y} \in \mathcal{Y}$. The performance is then measured under a log-loss

$$\ell(h(g(X=x,Y=y)),y) = \log \frac{1}{P_{\hat{Y}|\hat{X}}(\hat{y} \mid \hat{x})} \tag{3.12}$$

and the objective of the adversary simplifies to

$$\max_{h(\cdot)} -\mathbb{E}\left[\log \frac{1}{P_{\hat{Y}|\hat{X}}(h(g(X,Y)) \mid g(X,Y))}\right] = -\mathsf{H}(Y \mid g(X,Y)). \tag{3.13}$$

The mini-max game then reduces to

$$\min_{g(\cdot)} -\mathsf{H}(Y \mid g(X,Y)) = \min_{g(\cdot)} \mathsf{I}(g(X,Y);Y) - \mathsf{H}(Y). \tag{3.14}$$

Lastly, [2] introduces a more general α-loss function, which interpolates between the 0-1 loss and the log-loss via

$$\ell(h(g(X=x,Y=y)),y) = \frac{\alpha}{\alpha-1}(1 - P_{\hat{Y}|\hat{X}}(\hat{y} \mid \hat{x})^{1-\frac{1}{\alpha}}), \tag{3.15}$$

for any $\alpha > 1$. It is shown that as α becomes very large ($\alpha \to \infty$), the loss will approach that of the 0-1 (MAP) adversary. When α becomes smaller the estimator becomes more probabilistic. While as α approaches 1 the loss becomes the logarithmic loss. Using the Arimoto mutual information [13] [14] I_α^{a} and a Rényi entropy [15] term H_α results in the mini-max game formulation

$$\min_{g(\cdot)} -\mathsf{H}_\alpha^{\mathrm{a}}(Y \mid g(X,Y)) = \min_{g(\cdot)} \mathsf{I}_\alpha^{\mathrm{a}}(g(X,Y);Y) - \mathsf{H}_\alpha(Y). \tag{3.16}$$

The above mini-max game formulations can be solved directly when the data holder has access to $P_{X,Y}$. For the case when $P_{X,Y}$ is unknown [2] proposes a data-driven version of GAP. The data-driven version represents the privacy mechanism via a conditional generative model $g(X, Y; \theta_{\mathrm{p}})$ parameterized by θ_{p}, competing against a computational adversary modeled as a neural network $h(g(X, Y; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}})$ parameterized by θ_{adv}. The parameters θ_{p} and θ_{adv} can then be optimized through

$$\min_{\theta_{\mathrm{p}}} \max_{\theta_{\mathrm{adv}}} -\frac{1}{n} \sum_{i=1}^{n} \ell(h(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}}), y_{(i)}) \tag{3.17}$$

$$\text{s.t.} \quad \mathbb{E}[d(g(X, Y), X)] \leq \mathsf{D}. \tag{3.18}$$

This data-driven version of GAP is further investigated in Section 4.1, where the implementation of a binary data model is presented. Also in Section 5.1 achieved results from the implemented model is compared to ones reported in [2].

3.2 Private Information Retrieval and Extensions

Private information retrieval (PIR) schemes have the goal of allowing a user to retrieve information privately from a database. That is to obtain information stored in a database (or a set of databases) without disclosing any information (in an information-theoretic sense) about what information the user wants to retrieve to server(s) storing the information. Meaning the server should be unable to infer the index or identity of the file that the user retrieved. The concept of PIR was first introduced in [16], and since then, there have been numerous attempts of creating PIR schemes for both single and multiple database scenarios.

When evaluating the performance of PIR schemes, one usually measures the download (or PIR) rate, neglecting the upload cost (number of queries) since the download size typically greatly exceeds the upload. The optimal PIR scheme is one achieving the best possible PIR rate, called the PIR capacity.

The most trivial PIR scheme achieving information-theoretic privacy is the scheme where the user downloads the entire database directory to retrieve the desired file. In a single-server scenario, this is also the only possible scheme guaranteeing full information-theoretic privacy. This method is, however, highly inefficient when the directory size grows larger. Therefore, it is useful to consider a relaxed form of PIR, referred to as weakly-private information retrieval (WPIR). WPIR was first addressed independently by [17] and [18] to achieve better rates than the PIR capacity. By relaxing the perfect privacy condition and allowing for some leakage of private information, it has been shown that it is possible to improve the download rate. Moreover, an exact expression for the WPIR capacity was derived using mutual information and maximal leakage (see Section 2.1.6) as the privacy metrics [19]. In addition to relaxing the condition of full privacy of PIR (as WPIR), [20] proposes a new scheme that simultaneously relaxes the conditions of perfect retrievability and full privacy, named lossy weakly-private information retrieval (LWPIR). From [20], the following scheme is considered.

Definition 3.2.1. An M-file LWPIR scheme for a single server storing M files.

- A single server storing M files $\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(M)}$, each with β symbols from the alphabet $\mathcal{X}$, where $\boldsymbol{X}^{(m)} = (X_1^{(m)}, \ldots, X_\beta^{(m)})$, for $m \in [M]$, are assumed to be independent and identically distributed.

- The objective is to find a scheme allowing a user to obtain a file with index M while, to some degree keep M private from the server.

- The user generates the random query $Q \in \mathcal{Q}$, for some set $\mathcal{Q}$, according to the conditional distribution $P_{Q|M}(q \mid m)$, which is sent to the server.

- Given the query Q and the files $\boldsymbol{X}$, the server responds with the answer $A = A(Q, \boldsymbol{X}) \in \mathcal{A}$, for some set $\mathcal{A}$.

- Then finally the user tries to reconstruct the file $\hat{\boldsymbol{X}}^{(M)} = \hat{\boldsymbol{X}}^{(M)}(Q, A)$, where $\hat{\boldsymbol{X}}^{(M)} = (\hat{\boldsymbol{X}}_1^{(M)}, \ldots, \hat{\boldsymbol{X}}_\beta^{(M)}) \in \hat{\mathcal{X}}^{(\beta)}$, for some set $\hat{\mathcal{X}}$.

An illustration of the full system can be seen in Figure 3.2.

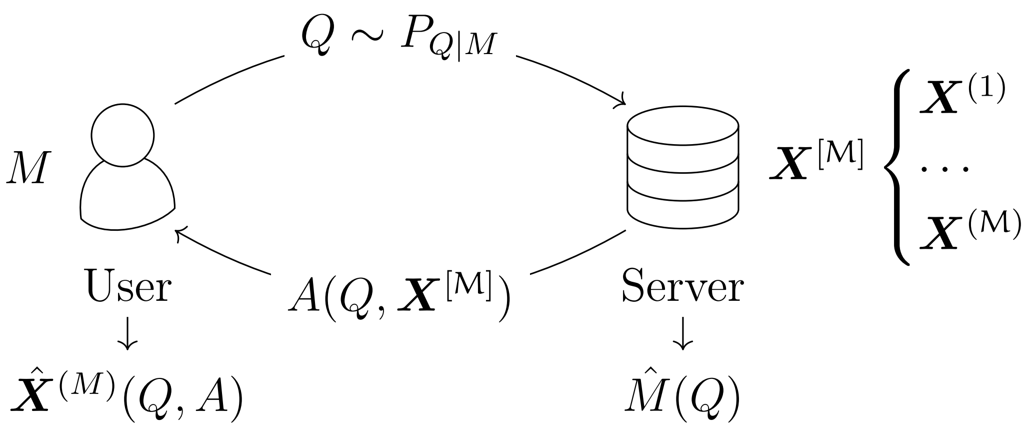

$Q \sim P_{Q|M}$
M
User
$\downarrow$
$\hat{\boldsymbol{X}}^{(M)}(Q, A)$
$A(Q, \boldsymbol{X}^{[\mathsf{M}]})$
Server
$\downarrow$
$\hat{M}(Q)$
$\boldsymbol{X}^{[\mathsf{M}]}$
$\boldsymbol{X}^{(1)}$
$\dots$
$\boldsymbol{X}^{(\mathsf{M})}$

Figure 3.2: Illustration of LWPIR system

From this definition $P_{Q|M}$ is seen as the privacy mechanism, and the information leakage is regarding the server inference of the index M, given the random outcome Q from $P_{Q|M}$. It is assumed that the server is *honest-but-curious*, meaning it serves the user correctly according to the query, but attempts to learn what the requested file was. Finding an optimal scheme results in a trifold tradeoff between the download rate, the distortion between the original and reconstructed file, and the amount of information leaked to the server concerning what file the user wants. Here schemes with no distortion reduced to a WPIR scheme, and further schemes with no leakage and no distortion reduces to a PIR scheme.

For ordinary PIR, the requirement of perfect privacy, that is, a server should be incapable of differentiating the answer, implies that the returned answer's file size must be equal for all requested files. However, this limitation is not present for LWPIR, where the download cost may be different depending on the file. For the single-server LWPIR scheme, similar to the work [20, 21], the download rate in this thesis is defined as the average coded length together with the entropy of the message (referred to as *information rate*),

$$\mathsf{R} \triangleq \frac{\mathsf{H}(A \mid Q)}{\beta} = \frac{1}{\beta}\sum_{q\in\mathcal{Q}} P_Q(q)\mathsf{H}(A \mid Q = q). \tag{3.19}$$

The distortion of the reconstructed file is defined as

$$\mathbb{E}_{M,Q,\boldsymbol{X}}\Big[d(\boldsymbol{X}^{(M)}, \hat{\boldsymbol{X}}^{(M)})\Big] \triangleq \frac{1}{\beta}\sum_{i=1}^{\beta} \mathbb{E}_{M,Q,X_i^{(M)}}\Big[d(X_i^{(M)}, \hat{X}_i^{(M)})\Big] \tag{3.20}$$

Here, $d : \mathcal{X} \times \hat{\mathcal{X}} \to \mathbb{R}_{\geq 0}$ is a per-symbol distortion function chosen based on the type of data, and

$$\mathbb{E}_{Q,\boldsymbol{X}^{(m)}}\Big[d(\boldsymbol{X}^{(m)}, \hat{\boldsymbol{X}}^{(m)})\Big] \triangleq \frac{1}{\beta}\sum_{i=1}^{\beta} \mathbb{E}_{Q,X_i^{(m)}}\Big[d(X_i^{(m)}, \hat{X}_i^{(m)})\Big], \qquad \forall m \in [\mathsf{M}]. \tag{3.21}$$

For leakage, two information-theoretic measures are considered (introduced in [19]), namely mutual information and MaxL. For mutual information, the information leakage is quantified by

$$\rho^{(\mathsf{MI})}(P_{Q|M}) \triangleq \mathsf{I}(M; Q). \tag{3.22}$$

The MaxL privacy metric, first introduced in [22], [23] is quantified by

$$\rho^{(\mathsf{MaxL})}(P_{Q|M}) \triangleq \mathsf{MaxL}(M;Q) \tag{3.23}$$

$$= \log_2 \sum_{q\in\mathcal{Q}} \max_{m\in[M]} P_{Q|M}(q \mid m). \tag{3.24}$$

It is also possible to define MaxL based on the *min-entropy* information leakage $I_\infty(M;Q)$ for the privacy mechanism $P_{Q|M}$, where

$$\mathsf{I}_\infty(M;Q) \triangleq \mathsf{H}_\infty(M) - \mathsf{H}_\infty(M \mid Q), \tag{3.25}$$

and $\mathsf{H}_\infty(M)$ is the *min-entropy* (see 2.1.7). From these definitions, [20] then denotes and defines the leakage for a given $P_{Q|M}$ as the probability of the maximum-likelihood (ML) guess

$$\rho(P_{Q|M}) \triangleq \frac{1}{\mathsf{M}} \sum_{q\in\mathcal{Q}} \max_{m\in[\mathcal{M}]} P_{Q|M}(q \mid m), \tag{3.26}$$

which is the leakage metric we used in this thesis. Here, leakage is in the range $1/\mathsf{M} \leq \rho \leq 1$, where $\rho = 1/\mathsf{M}$ corresponds to "no-leakage" (best effort is random guess) and $\rho = 1$ corresponds to "no-privacy" (always correct guess).

When the data is binary data, a suitable distortion function is the Hamming distance (Section 2.1.5). The distortion is here measured per symbol, and the distortion is the average number of incorrect reconstructed symbols of $\mathbf{X}^{(M)}$. The best estimation is then the per-symbol maximum likelihood estimate

$$\hat{\mathbf{X}}^{(m)} = (\hat{X}_1^{(m)}, \ldots, \hat{X}_\beta^{(m)}), \tag{3.27}$$

where

$$\hat{X}_i^{(m)}(q,a) \triangleq \arg\max_{y\in\mathcal{X}} \Pr[A = a \mid M = m, Q = q, X_i^{(m)} = y]. \tag{3.28}$$

The goal is then to find schemes with the minimum download rate given constraints for distortion and leakage. It was shown in [20, 21] that this optimization problem can be solved numerically given the data distribution for the files stored on the server. However, for the case where the distribution is unknown, it is interesting to consider a data-driven approach where an optimal scheme is learned directly from the data. In Section 4.2 a data-driven model is presented which utilizes neural networks in a GAN fashion (see Section 2.2.8) to find LWPIR schemes.

Chapter 4

Methodology and Model Architecture

This chapter will present the two models implemented in this thesis. First the Context-Aware Generative Adversarial Privacy model (GAP), already introduced in Section 3.1. Then the LWPIR GAN model, a new framework developed with ambitions of finding LWPIR schemes, as described in Section 3.2. Each model is presented in the following two sections, with subsections for general model design, loss measures used and the training algorithm. The results and findings for each model are presented the next chapter, Chapter 5.

4.1 GAP Model

As introduced in Section 3.1, Generative Adverserial Privacy illustrates a seemingly very compelling privacy framework built in a GAN fashion (Section 2.2.8). As a starting point to further investigate the use of GANs in a privacy perceiving context, it is therefore interesting to first implement the GAP framework with the goal of achieving similar results to those described in [2]. This implementation considers the case where the public and private variables are binary valued random variables. The public and private variables are denoted as (X, Y), with the joint probability $p_{j,i}$ for $(X, Y) = (i, j)$, where $i, j \in \{0, 1\}$. The next section introduces the theoretical foundation for the binary GAP model and then describes the structure of the neural network framework.

4.1.1 Model Design

Starting with the fundamentals, the framework consist of the two neural networks, the adversary and privatizer, competing against each other in a GAN fashion. As described in Section 3.1, the goal of the privatizer is to find a privacy mapping from X, Y to $\hat{X}$ which prevents the adversary from correctly inferring the private variable Y (*private-data dependent* privacy mechanisms from [2]). The adversary considered is a strong MAP adversary which;

1. has access to the joint distribution $p_{j,i}$,
2. knows of the learned privacy scheme, and
3. is able to compute the MAP (Section 2.2.9) rule.

Theory

The privacy mechanism $g(X,Y)$ maps the private and public variable pair (X,Y) to $\hat{X}$. Since the variables are binary this is represented by the conditional distribution $P_{\hat{X}|X,Y}$ as

$$
\begin{aligned}
P_{\hat{X}|X,Y}(0 \mid 0,0) = s_{0,0},\ \ P_{\hat{X}|X,Y}(0 \mid 0,1) = s_{0,1},\\
P_{\hat{X}|X,Y}(1 \mid 1,0) = s_{1,0},\ \ P_{\hat{X}|X,Y}(1 \mid 1,1) = s_{1,1}.
\end{aligned}
$$

The marginal distribution of $\hat{X}$ is then given by

$$
\begin{aligned}
P_{\hat{X}}(0) &= \sum_{x,y} P_{\hat{X}|X,Y}(0 \mid x,y) P_{X,Y}(x,y)\\
&= s_{0,0}p_{0,0} + s_{0,1}p_{0,1} + (1-s_{1,0})p_{1,0} + (1-s_{1,1})p_{1,1},\\
P_{\hat{X}}(1) &= \sum_{x,y} P_{\hat{X}|X,Y}(1 \mid x,y) P_{X,Y}(x,y)\\
&= (1-s_{0,0})p_{0,0} + (1-s_{0,1})p_{0,1} + s_{1,0}p_{1,0} + s_{1,1}p_{1,1}.
\end{aligned}
$$

The inference accuracy of the adversary is

$$
\begin{aligned}
P_{Y,\hat{X}}(0,0) &= \sum_{x} P_{X,Y}(x,0) P_{\hat{X}|X,Y}(0 \mid x,0) = p_{1,0}(1-s_{1,0}) + p_{0,0}s_{0,0},\\
P_{Y,\hat{X}}(1,0) &= \sum_{x} P_{X,Y}(x,1) P_{\hat{X}|X,Y}(0 \mid x,1) = p_{1,1}(1-s_{1,1}) + p_{0,1}s_{0,1},\\
P_{Y,\hat{X}}(0,1) &= \sum_{x} P_{X,Y}(x,0) P_{\hat{X}|X,Y}(1 \mid x,0) = p_{1,0}s_{1,0} + p_{0,0}(1-s_{0,0}),\\
P_{Y,\hat{X}}(1,1) &= \sum_{x} P_{X,Y}(x,1) P_{\hat{X}|X,Y}(1 \mid x,1) = p_{1,1}s1,1 + p_{0,1}(1-s_{0,1}).
\end{aligned}
$$

Defining $\mathbf{s} = \{s_{0,0}, s_{0,1}, s_{1,0}, s_{1,1}\}$, the MAP adversary's inference accuracy for $\hat{X} = 0$, is given by

$$
P_d^{(B)}(\mathbf{s}, \hat{X} = 0) \triangleq \max\{\Pr[Y = 1, \hat{X} = 0], \Pr[Y = 0, \hat{X} = 0]\}, \tag{4.1}
$$

and for $\hat{X} = 1$

$$
P_d^{(B)}(\mathbf{s}, \hat{X} = 1) \triangleq \max\{\Pr[Y = 1, \hat{X} = 1], \Pr[Y = 0, \hat{X} = 1]\}. \tag{4.2}
$$

Then, given a fixed privacy mechanism $\mathbf{s}$, the MAP adversary's inference accuracy is defined as

$$
P_d^{(B)} \triangleq \max_{h(\cdot)} \Pr[h(g(X,Y)) = Y] = P_d^{(B)}(\mathbf{s}, \hat{X} = 0) + P_d^{(B)}(\mathbf{s}, \hat{X} = 1). \tag{4.3}
$$

The optimal privacy mechanism $\mathbf{s}$ (referred to as the private-data dependent (PDD) privacy mechanism in [2]) can be found by linearly solving the constrained optimization problem parameterized by $\boldsymbol{p} = \{p_{0,0}, p_{0,1}, p_{1,0}, p_{1,1}\}$ and the distortion constraint D

$$\begin{aligned}
\min_{s_{1,1},s_{0,1},s_{1,0},s_{0,0},t_0,t_1} \quad & t_0 + t_1 && (4.4)\\
\text{s.t.} \quad & 0 \leq s_{1,1}, s_{0,1}, s_{1,0}, s_{0,0} \leq 1 && (4.5)\\
& p_{1,1}(1-s_{1,1}) + p_{0,1}s_{0,1} \leq t_0 && (4.6)\\
& p_{1,0}(1-s_{1,0}) + p_{0,0}s_{0,0} \leq t_0 && (4.7)\\
& p_{1,1}s_{1,1} + p_{0,1}(1-s_{0,1}) \leq t_1 && (4.8)\\
& p_{1,1}s_{1,0} + p_{0,0}(1-s_{0,0}) \leq t_1 && (4.9)\\
& p_{1,1}(1-s_{1,1}) + p_{0,1}(1-s_{0,1}) + p_{1,0}(1-s_{1,0}) + p_{0,0}(1-s_{0,0}) \leq D, && (4.10)
\end{aligned}$$

where t_0 and t_1 are variables for the maxima in Equation 4.1 and Equation 4.2.

Neural Network Architecture

The optimal MAP adversary can however not be used during training since this does not give the privatizer enough of a gradient for optimal training. Therefore training of the privatizer is done against a computational adversary represented by a neural network, intending to achieve a privatization scheme that matches the optimal game-theoretic one. The privatizer is modeled as a single-layer neural network classifier with the parameters $\theta_{\text{p}} = \mathbf{s} = \{s_{0,0}, s_{0,1}, s_{1,0}, s_{1,1}\}$. While the adversary is modeled as a two-layer neural network classifier with the parameters $\theta_{\text{adv}} = (\theta_{\text{adv},0}, \theta_{\text{adv},1})$, where $\theta_{\text{adv},0} = \Pr[Y = 0 \mid \hat{x} = 0]$ and $\theta_{\text{adv},1} = \Pr[Y = 1 \mid \hat{x} = 1]$. The resulting formulation for the privatizer belief of an adversary guessing $y_{(i)} = 1$ condition on the input $(x_{(i)}, y_{(i)})$ is

$$h(g(x_{(i)}, y_{(i)}; \mathbf{s}); \theta_{\text{adv}}) = \theta_{\text{adv},1} \Pr[\hat{x}_{(i)} = 1] + (1 - \theta_{\text{adv},0}) \Pr[\hat{x}_{(i)} = 0], \qquad (4.11)$$

where

$$\begin{aligned}
\Pr[\hat{X} = 0] =& x_{(i)}y_{(i)}(1-s_{1,1}) + (1-x_{(i)})y_{(i)}s_{0,1}\\
&+ x_{(i)}(1-y_{(i)})(1-s_{1,0}) + (1-x_{(i)})(1-y_{(i)})s_{0,0},\\
\Pr[\hat{X} = 1] =& x_{(i)}y_{(i)}(1-s_{1,1}) + (1-x_{(i)})y_{(i)}s_{0,1}\\
&+ x_{(i)}(1-y_{(i)})(1-s_{1,0}) + (1-x_{(i)})(1-y_{(i)})s_{0,0}.
\end{aligned}$$

An illustration of the neural network structure can be seen in Figure 4.1. The following two subsections describe the loss measures and the training algorithm used for finding the optimal parameters $\theta_{\text{p}}, \theta_{\text{adv}}$.

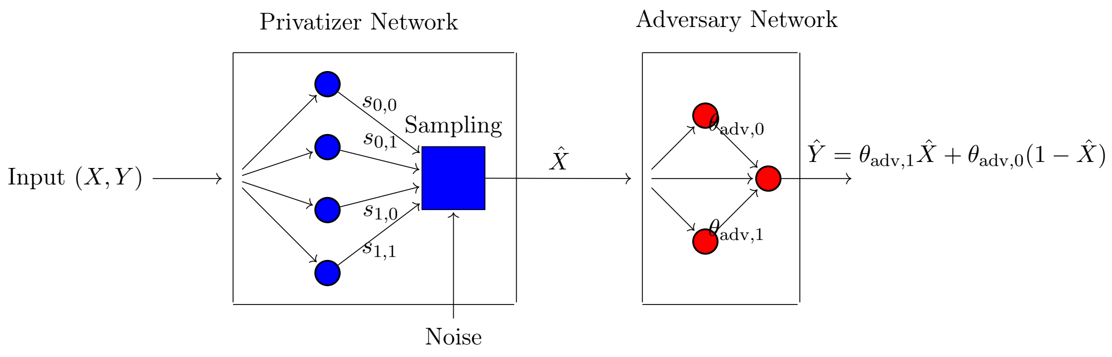


Figure 4.1: Neural network structure of the privatizer and adversary.

4.1.2 Loss Measures

As already mentioned in Section 3.1, many different loss measures where proposed in [2] leading to different model outcomes. Since this implementation considers binary data, it makes sense to use an empirical log-loss for the adversary, similar to what was introduced in Equation 3.17. The adversary's loss function is

$$\begin{aligned}\ell(h(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}}), y_{(i)}) = &- y_{(i)} \log h(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}}) \\ &- (1 - y_{(i)}) \log(1 - h(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}})).\end{aligned}$$

The adversary then learns the optimal θ^*_{adv} by maximizing

$$\begin{aligned}-f_{\mathrm{XE-Loss}}(h(g(X, Y; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}}), Y) = &- \frac{1}{n} \sum_{i=1}^{n} y_{(i)} \log h(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}}) \\ &- (1 - y_{(i)}) \log(1 - h(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}})).\end{aligned}$$

Similarly, loss of the privatizer is the negative mean absolute error of the adversary. Given a fixed adversary θ_{adv}, the privatizer then learns the optimal θ_{p} by minimizing $-f_{\mathrm{XE-Loss}}(h(g(X, Y; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}}), Y)$ subject to a distortion constraint,

$$-f_{\mathrm{XE-Loss}}(h(g(X, Y; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}}), Y) + p_t \max\{1, d(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}), x_{(i)}) - D\}.$$

Here the distortion considered is the Hamming distance (Section 2.1.5). The variable p_t is a penalty constraint which is gradually increased for each training iteration.

4.1.3 Training Algorithm

The used training algorithm for the network is taken from [2], and is as follows

Algorithm 2: Alternating minimax privacy preserving algorithm

```
input  : Dataset $\mathcal{D}$, distortion parameter $D$, iteration number $T$
output: Optimal privatizer parameter $\theta_{\mathrm{p}}$
procedure ALERNATE MINIMAX($\mathcal{D}$, D, T)
    Initialize $\theta_{\mathrm{p}}^1$ and $\theta_{\mathrm{adv}}^1$
    for number of training iterations do
        Random minibatch of M datapoints $\{x(1), ..., x(\mathsf{M})\}$ drawn from full dataset
        Generate $\{\hat{x}_{(1)}, ..., \hat{x}_{(\mathsf{M})}\}$ via $\hat{x}_{(i)} = g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}^t)$
        Update the parameter $\theta_{\mathrm{adv}}^{t+1}$ for the adversary
            $\theta_{\mathrm{adv}}^{t+1} = \max_{\theta_{\mathrm{adv}}} -\frac{1}{\mathsf{M}} \sum_{i=1}^{\mathsf{M}} \ell(h(\hat{x}_{(i)}; \theta_{\mathrm{adv}}), y_{(i)})$
        Compute the descent direction $\nabla_{\theta_{\mathrm{p}}} \ell(\theta_{\mathrm{p}}, \theta_{\mathrm{adv}}^{t+1})$, where
            $\ell(\theta_{\mathrm{p}}, \theta_{\mathrm{adv}}^{t+1}) = -\frac{1}{\mathsf{M}} \sum_{i=1}^{\mathsf{M}} \ell(h(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}); \theta_{\mathrm{adv}}^{t+1}), y_{(i)})$
            subject to $\frac{1}{\mathsf{M}} \sum_{i=1}^{\mathsf{M}} [d(g(x_{(i)}, y_{(i)}; \theta_{\mathrm{p}}), x_{(i)})] \leq D$
        Update the parameter $\theta_{\mathrm{p}}^{t+1}$ for the privatizer
            $\theta_{\mathrm{p}}^{t+1} = \theta_{\mathrm{p}}^t - \alpha_t \nabla_{\theta_{\mathrm{p}}} \ell(\theta_{\mathrm{p}}, \theta_{\mathrm{adv}}^{t+1}), \qquad \alpha_t > 0$
        Exit if solution converged
    return $\theta_{\mathrm{p}}^{t+1}$
```

Since the framework is built in a GAN fashion, the training needs to abide by the synchronisation characteristics described in Section 2.2.8. The algorithm is therefore very similar to Algorithm 1, where first the adversary is trained on a fixed privatizer and then the privatizer is trained given a fixed adversary. Each weight update to the privatizer is, as described in the previous section, subject to the distortion constraint. Also α_t is used as a hyperparameter that gradually scales the loss (increases for each iteration), with the intention to achieve more stable training. The algorithm ends after a set number of training iterations T, or earlier if the model converges on a solution.

4.2 LWPIR GAN Model

The second model implemented in this thesis is a data driven GAN model designed with goal of finding LWPIR schemes, titled *lossy weakly-private information retrieval GAN* (LWPIR GAN). As introduced in Section 3.2, LWPIR aims to enable users to retrieve files from a server with quantifiable measures for

1. information leakage, regarding the servers ability to infer which file was retrieved,
2. file distortion, between the original file on the server and the user's reconstructed file, and
3. download rate, amount of data needed to be downloaded for the retrieval of the file.

Ideally the model should be fully general, meaning it should handle a dataset containing M binary files $\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(\mathsf{M})}$ stored on a single server, where each file $\boldsymbol{X}^{(m)} = (\boldsymbol{X}_1^{(m)}, \ldots, \boldsymbol{X}_\beta^{(m)})$ is a binary random vector of length β. However due to challenges in development and time constraints, the scope of this implementation is limited to the case where a single server stores two files each containing one binary bit; $\mathsf{M} = 2$ and $\beta = 1$. The entries are considered to be independent and identically distributed (i.i.d.) according to $\Pr[X_i^{(m)} = 0] = \Pr[X_i^{(m)} = 1] = 1/2$. It is also assumed that the files are independent, giving

$$P_X(x_1^{(1)}, x_1^{(2)}) = \frac{1}{2^{\mathsf{M}\beta}} = \frac{1}{2^{2\cdot 1}} = \frac{1}{4}. \tag{4.12}$$

The marginal distribution of the requested file index $M = m$ is denoted as p_m, where $m \in [1 : \mathsf{M}]$, and is assumed to be

$$p_m = \frac{1}{\mathsf{M}} = \frac{1}{2}. \tag{4.13}$$

The next section introduces the theoretical formulations for the LWPIR model and then describes the structure of the neural network framework implemented.

4.2.1 Model Design

Similar to the GAP implementation described in the previous Section 4.1, the framework is built in a GAN fashion (Section 2.2.8) with competing neural networks. Before explaining the neural network framework it is helpful to first introduce the theoretical foundation for the LWPIR scheme. As described in Section 3.2, the user generates a query $Q = q \in \mathcal{Q}$, with $|\mathcal{Q}| = \alpha$, according to a privacy mechanism $Q = f_Q(M)$ and send it to the server. Given the query Q the server responds with an answer A, generated according to an answer function $A = A(Q, \boldsymbol{X}^{([\mathsf{M}])})$. Additionally the server, curios about the index M, tries to guess $\hat{M} = M$ based on the query Q and the files $\boldsymbol{X}$. Consequently the LWPIR scheme consists of the privacy mechanism $f_Q(M)$ and the answer function $A(Q, \boldsymbol{X}^{([\mathsf{M}])})$, chosen based on constraints regarding information leakage, distortion and download rate.

Neural Network Architecture

As with the GAP implementation in Section 4.1, the framework consists of a privatizer model (generating the queries Q given the index M) competing against an adversary model (inferring the index $\hat{M} = m$ from the query Q). Additionally the framework consists of a third neural network, the *answer model*, generating an answer $\hat{\boldsymbol{X}}$ based on the query Q, that also competes against the privatizer and adversary.

As mentioned in the introduction, even though the presented formulations are general regarding the number of files and file size, this implementation considers the case for a single server storing two files each containing one bit. The implemented model is however general for the number of queries α, which is specified as an input parameter. The final model was tested for different number of queries, but for illustration purposes this considers a model with $\alpha = 4$ (where $\mathsf{M} = 2$ and $\beta = 1$).

Privatizer network Starting with the privatizer, the privacy mechanism is represented by a conditional distribution $P_{Q|M}(q \mid m)$ that maps the requested file index M to a randomized output Q given by

$$P_{Q|M}(q_n \mid m) = p_{m,q_n}, \quad m \in [1 : \mathsf{M}] \text{ and } n \in [1 : \alpha] \tag{4.14}$$

and the marginal distribution of Q can be expressed as

$$\begin{aligned} P_Q(q_n) &= \sum_{m=1}^{\mathsf{M}} P_M(m) P_{Q|M}(q_n \mid m) & (4.15)\\ &= \sum_{m=1}^{\mathsf{M}} p_m \cdot p_{m,q_n}, \quad n \in [1 : \alpha], & (4.16)\end{aligned}$$

These *transition probabilities* p_{m,q_n} can be represented by a *transition probability matrix*

$$\mathsf{P} \triangleq \begin{pmatrix} p_{1,q_1} & p_{1,q_2} & \cdots & p_{1,q_\alpha} \\ \vdots & & & \\ p_{\mathsf{M},q_1} & p_{\mathsf{M},q_2} & \cdots & p_{\mathsf{M},q_\alpha} \end{pmatrix}. \tag{4.17}$$

The privatizer is then modeled as a single-layer neural network classifier with the parameters $\theta_{\mathrm{p}} = \mathsf{P} = \{p_{1,q_1}, p_{1,q_2}, p_{1,q_3}, p_{1,q_4}, p_{2,q_1}, p_{2,q_2}, p_{2,q_3}, p_{2,q_4}\}$ with the input P_M as a is a one-hot encoding (Section 2.2.6)

$$P_M(1) = [1, 0], \tag{4.18}$$
$$P_M(2) = [0, 1]. \tag{4.19}$$

An illustration of the privatizer network can be seen in Figure 4.2.

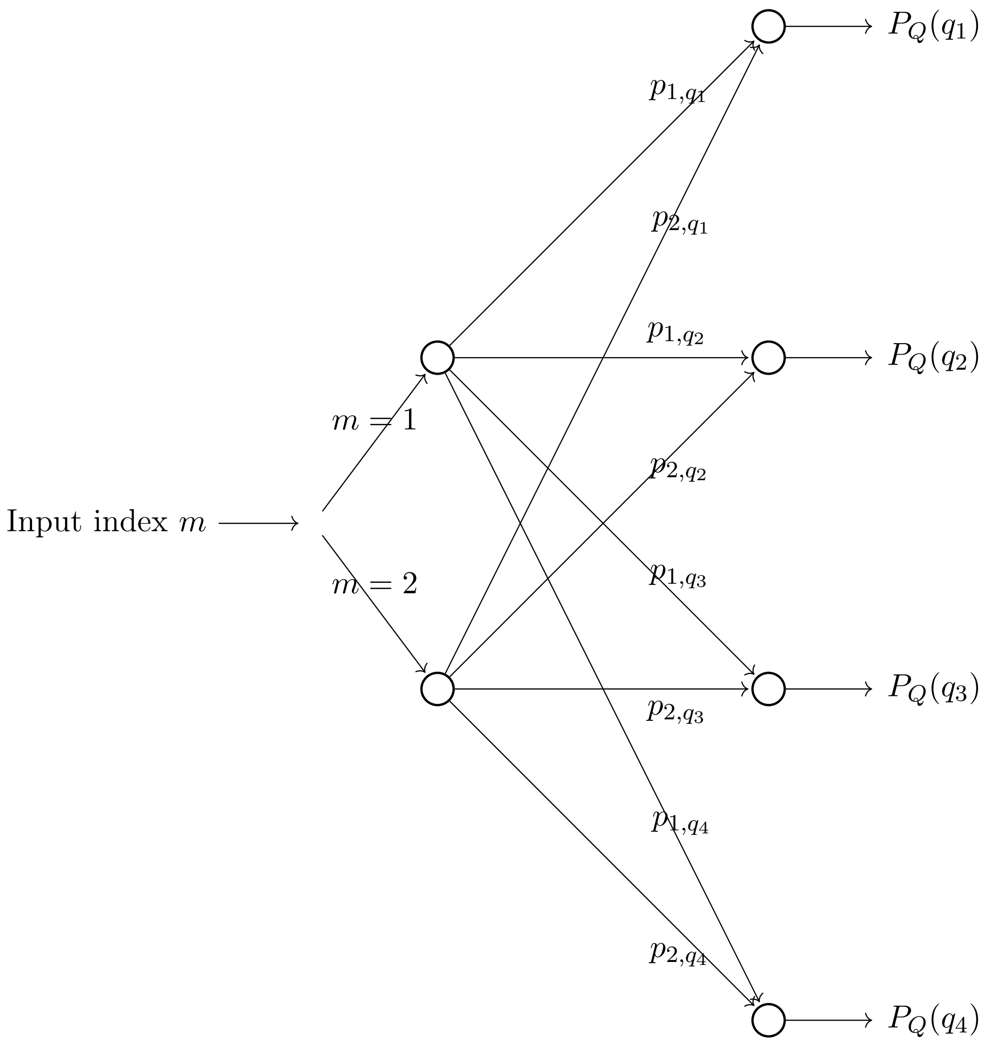


Figure 4.2: Illustration of the privatizer neural network in the LWPIR model.

Answer network The server after receiving the query $Q = q_n$ sent by the user, responds with the answer $(\hat{\boldsymbol{X}}^{(1)}, \hat{\boldsymbol{X}}^{(2)}, \cdots, \hat{\boldsymbol{X}}^{(\mathsf{M})})$ according to an answer function $(\hat{\boldsymbol{X}}^{(1)}, \hat{\boldsymbol{X}}^{(2)}, \cdots, \hat{\boldsymbol{X}}^{(\mathsf{M})}) = f_A(\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \cdots, \boldsymbol{X}^{(\mathsf{M})}, Q)$. Similar to the concept of the privatizer, the answer function f_A is represented by a conditional distribution

$$P_{\hat{\boldsymbol{X}}^{([\mathsf{M}])}|\boldsymbol{X}^{([\mathsf{M}])},Q}(\hat{\boldsymbol{x}}^{(1)}, \hat{\boldsymbol{x}}^{(2)}, \ldots, \hat{\boldsymbol{x}}^{(\mathsf{M})} \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(\mathsf{M})}, q_n). \tag{4.20}$$

By using the shorthand notation $\boldsymbol{W} \triangleq (\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \cdots, \boldsymbol{X}^{(\mathsf{M})}) \in \{0,1\}^{\mathsf{M}\beta}$ and $\hat{\boldsymbol{W}} \triangleq (\hat{\boldsymbol{X}}^{(1)}, \hat{\boldsymbol{X}}^{(2)}, \cdots, \hat{\boldsymbol{X}}^{(\mathsf{M})})$, the transitional probabilities of the answer scheme, given a query $Q = q_n, n \in [1:\alpha]$, are denoted as

$$P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}, q_n) = s_{\boldsymbol{w},\hat{\boldsymbol{w}};q_n}, \tag{4.21}$$

and the corresponding transition probability matrix is defined as

$$\mathsf{S}_{q_n} \triangleq (s_{\boldsymbol{w},\hat{\boldsymbol{w}};q_n})_{\boldsymbol{w},\hat{\boldsymbol{w}}\in\{0,1\}^{\mathsf{M}\beta}}. \tag{4.22}$$

Given $\alpha = 4$, $\mathsf{M} = 2$ and $\beta = 1$, the answer scheme is then represented by the conditional distribution

$$\mathsf{S}_{q_n} = \begin{pmatrix} \Pr[\hat{\boldsymbol{W}}=(0,0)|\boldsymbol{W}=(0,0),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(0,1)|\boldsymbol{W}=(0,0),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(1,0)|\boldsymbol{W}=(0,0),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(1,1)|\boldsymbol{W}=(0,0),Q=q_n] \\ \Pr[\hat{\boldsymbol{W}}=(0,0)|\boldsymbol{W}=(0,1),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(0,1)|\boldsymbol{W}=(0,1),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(1,0)|\boldsymbol{W}=(0,1),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(1,1)|\boldsymbol{W}=(0,1),Q=q_n] \\ \Pr[\hat{\boldsymbol{W}}=(0,0)|\boldsymbol{W}=(1,0),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(0,1)|\boldsymbol{W}=(1,0),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(1,0)|\boldsymbol{W}=(1,0),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(1,1)|\boldsymbol{W}=(1,0),Q=q_n] \\ \Pr[\hat{\boldsymbol{W}}=(0,0)|\boldsymbol{W}=(1,1),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(0,1)|\boldsymbol{W}=(1,1),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(1,0)|\boldsymbol{W}=(1,1),Q=q_n] & \Pr[\hat{\boldsymbol{W}}=(1,1)|\boldsymbol{W}=(1,1),Q=q_n] \end{pmatrix}$$

$$= \begin{pmatrix} s_{0,0;q_n} & s_{0,1;q_n} & s_{0,2;q_n} & s_{0,3;q_n} \\ s_{1,0;q_n} & s_{1,1;q_n} & s_{1,2;q_n} & s_{1,3;q_n} \\ s_{2,0;q_n} & s_{2,1;q_n} & s_{2,2;q_n} & s_{2,3;q_n} \\ s_{3,0;q_n} & s_{3,1;q_n} & s_{3,2;q_n} & s_{3,3;q_n} \end{pmatrix}.$$

The answer model is modeled as a single-layer neural network classifier with the parameters $\theta_{\text{ans}} = \mathsf{S}_{q_n}$, with the input $P_{\boldsymbol{W}}(\boldsymbol{w})$, $\boldsymbol{w} \in \{0,1\}^{\mathsf{M}\beta}$ which, similar to the privatizer model, is a one-hot encoding (Section 2.2.6) on the form

$$P_{\boldsymbol{W}}(0,0) = [1,0,0,0], \tag{4.23}$$
$$P_{\boldsymbol{W}}(0,1) = [0,1,0,0], \tag{4.24}$$
$$P_{\boldsymbol{W}}(1,0) = [0,0,1,0], \tag{4.25}$$
$$P_{\boldsymbol{W}}(1,1) = [0,0,0,1]. \tag{4.26}$$

The model weight then corresponds to $P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}, q)$, which is used to calculate the distortion loss described in Equation 4.34. The output of the model is $P_{\hat{\boldsymbol{W}}|Q}(\hat{\boldsymbol{w}} \mid q_n)$ given by

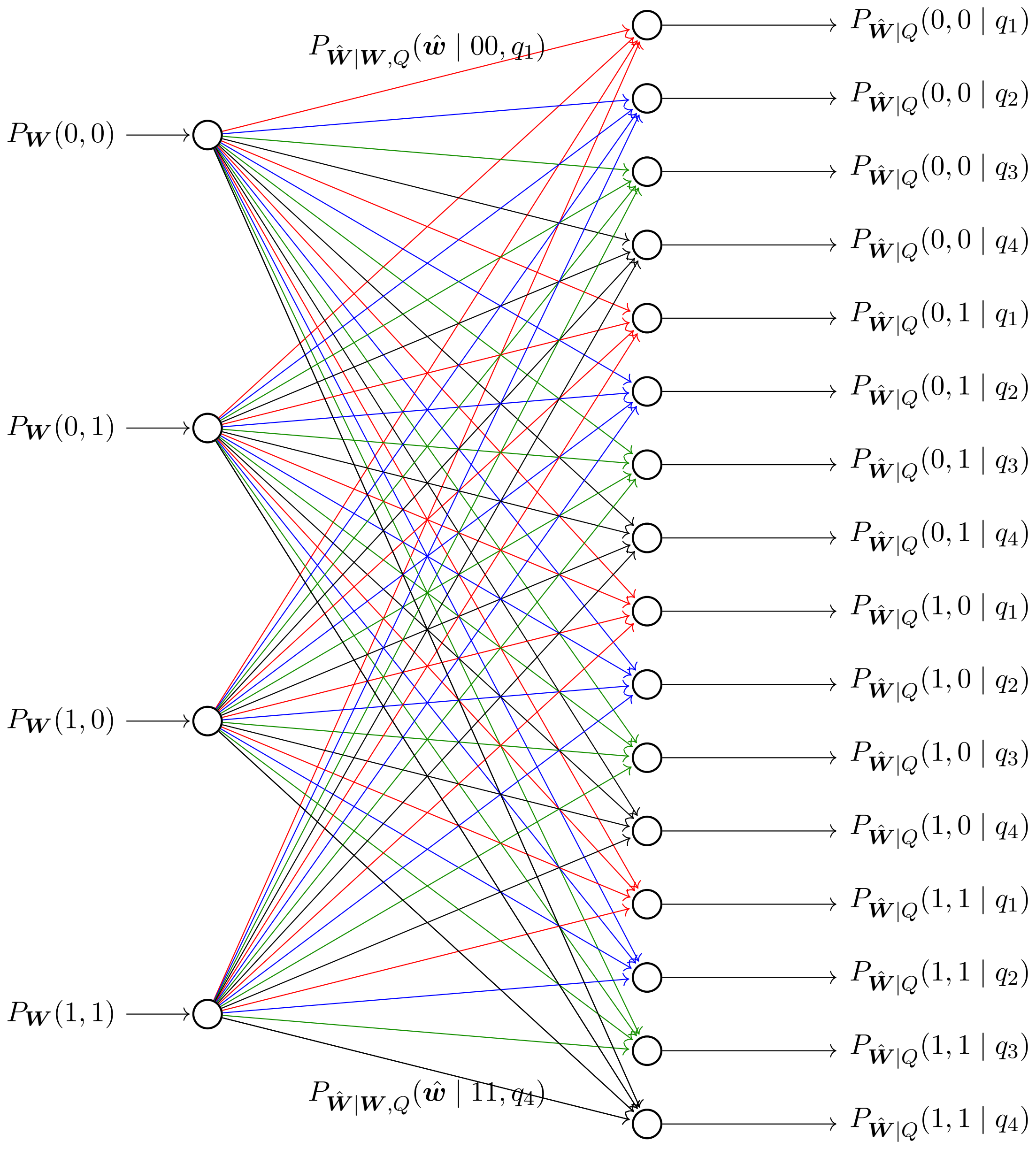
$$P_{\hat{\boldsymbol{W}}|Q}(\hat{\boldsymbol{w}} \mid q_n) = \sum_{\boldsymbol{w}\in\{0,1\}^{\mathsf{M}\beta}} P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}, q_n) P_{\boldsymbol{W}}(\boldsymbol{w}), \tag{4.27}$$

which is used to calculate the rate loss described in Equation 4.35. From $P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}, q)$ it is also possible to further calculate the answer probability $P_{\hat{\boldsymbol{W}}|\boldsymbol{W}}(\hat{\boldsymbol{w}} \mid \boldsymbol{w})$ by

$$P_{\hat{\boldsymbol{W}}|\boldsymbol{W}}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}) = \sum_{q\in\mathcal{Q}} P_{Q|\boldsymbol{W}}(q_n \mid \boldsymbol{w}) P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}, q_n) \tag{4.28}$$
$$\overset{(a)}{=} \sum_{q\in\mathcal{Q}} P_Q(q_n) P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}, q_n), \tag{4.29}$$

where (a) follows since without loss of generality, the queries generated by the user can be assumed to be independent of the files stored in the server, i.e., Q and $\boldsymbol{W}$ are independent.

An illustration of the answer network can be seen in Figure 4.3.



$P_{\boldsymbol{W}}(0,0)$
$P_{\boldsymbol{W}}(0,1)$
$P_{\boldsymbol{W}}(1,0)$
$P_{\boldsymbol{W}}(1,1)$
$P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid 00, q_1)$
$P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid 11, q_4)$
$P_{\hat{\boldsymbol{W}}|Q}(0,0 \mid q_1)$
$P_{\hat{\boldsymbol{W}}|Q}(0,0 \mid q_2)$
$P_{\hat{\boldsymbol{W}}|Q}(0,0 \mid q_3)$
$P_{\hat{\boldsymbol{W}}|Q}(0,0 \mid q_4)$
$P_{\hat{\boldsymbol{W}}|Q}(0,1 \mid q_1)$
$P_{\hat{\boldsymbol{W}}|Q}(0,1 \mid q_2)$
$P_{\hat{\boldsymbol{W}}|Q}(0,1 \mid q_3)$
$P_{\hat{\boldsymbol{W}}|Q}(0,1 \mid q_4)$
$P_{\hat{\boldsymbol{W}}|Q}(1,0 \mid q_1)$
$P_{\hat{\boldsymbol{W}}|Q}(1,0 \mid q_2)$
$P_{\hat{\boldsymbol{W}}|Q}(1,0 \mid q_3)$
$P_{\hat{\boldsymbol{W}}|Q}(1,0 \mid q_4)$
$P_{\hat{\boldsymbol{W}}|Q}(1,1 \mid q_1)$
$P_{\hat{\boldsymbol{W}}|Q}(1,1 \mid q_2)$
$P_{\hat{\boldsymbol{W}}|Q}(1,1 \mid q_3)$
$P_{\hat{\boldsymbol{W}}|Q}(1,1 \mid q_4)$

Figure 4.3: Illustration of the answer neural network in the LWPIR model. Each edge color represents the different queries; red: $Q = 1$, blue: $Q = 2$, green: $Q = 3$, black: $Q = 4$.

Adversary network The server also tries to infer the index $\hat{M} = m$ with the probability

$$P_{M,Q}(m, q_n) = P_M(m)P_{Q|M}(q_n \mid m) = p_m \cdot p_{m,q_n} \tag{4.30}$$

Given $Q = q_n$, the probability of MAP(REF MAP) correct guess for the server is given by

$$P_c(\mathsf{P}, Q = q_n) = \max_{m\in[1:\mathsf{M}]} \{\Pr[M = m, Q = q_n]\} \overset{(a)}{=} \frac{1}{\mathsf{M}} \max_{m\in[1:\mathsf{M}]} p_{m,q_n}, \tag{4.31}$$

where (a) holds if $p_m = {}^1\!/\!_{\mathsf{M}}$, $m \in [1 : \mathsf{M}]$, i.e., $M \sim \mathcal{U}([1 : \mathsf{M}])$. Then, given a fixed privacy mechanism P, the average correct probability of the MAP guess for the server is equal to

$$P_c(\mathsf{P}) = \sum_{n=1}^{\alpha} \max_{m\in[1:\mathrm{M}]} \{P_M(m)P_{Q|M}(q_n \mid m)\} = \sum_{n=1}^{\alpha} \max_{m\in[1:\mathrm{M}]} p_{m,q_n}. \tag{4.32}$$

Similar to the GAP implementation the final privatizer is evaluated against a MAP adversary, however during training a computational adversary is used. The adversary is modeled as a two-layer neural network classifier with the parameters $\theta_{\text{adv}} = P_{\hat{M}|Q}(\hat{m} \mid q)$, and can be expressed by the equation

$$\Pr[\hat{M} = m] = \sum_{q\in Q} P_Q(q) \cdot P_{\hat{M}|Q}(\hat{m} \mid q) \tag{4.33}$$

An illustration of the adversary network can be seen in Figure 4.4.

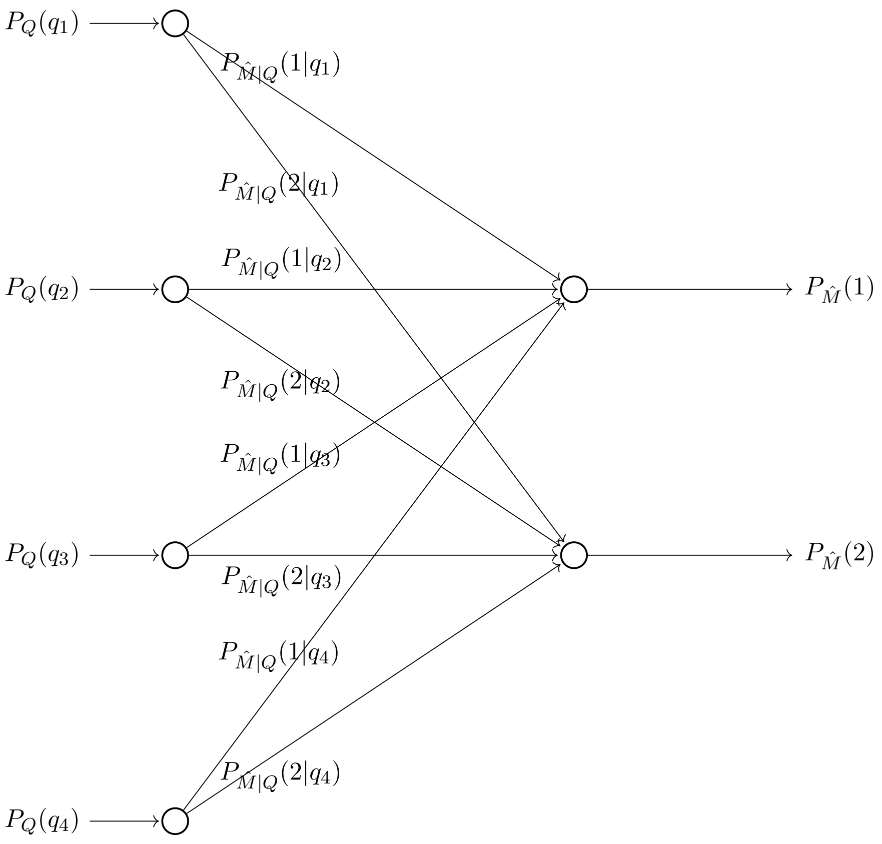


Figure 4.4: Illustration of the adversary neural network in the LWPIR model.

The privatizer, adversary and answer network were all built using the transposed convolutional layer, described in Section 2.2.7. The complete LWPIR model then consists of the three neural networks; privatizer, adversary and answer network, competing against each other in a GAN fashion. An illustration of the full model, where each model is abstracted, can be seen in Figure 4.5.

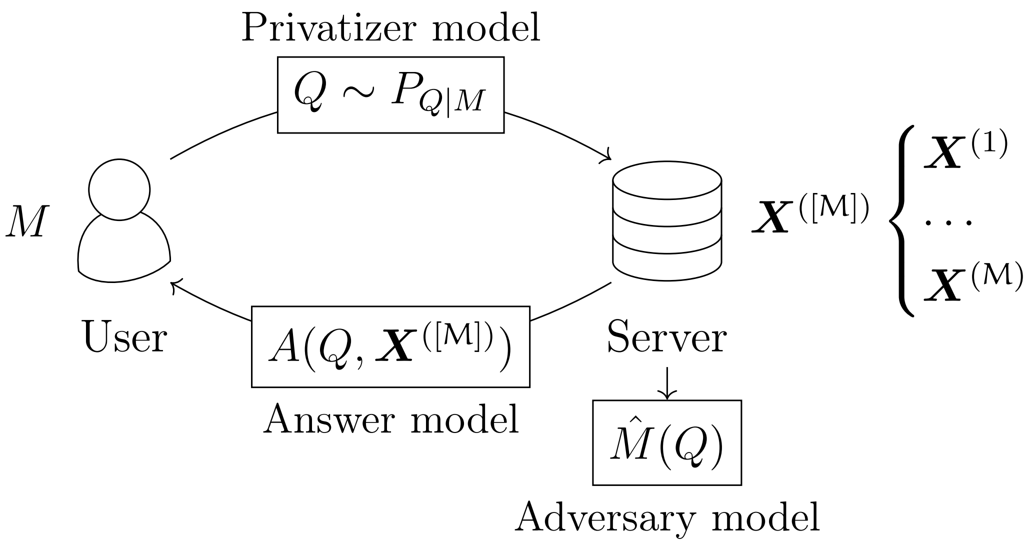


Figure 4.5: Illustration of the complete LWPIR model.

4.2.2 Loss Measures

As stated in the introduction of this section, the three different losses considered are file distortion, download rate and information leakage. The distortion is defined as the average per-symbol distortion between two binary vectors $\boldsymbol{x}^{(m)}$ and $\hat{\boldsymbol{x}}^{(m)}$. Based on a given query $Q = q_n, n \in [1 : \alpha]$, the distortion is defined as

$$\begin{aligned}
\mathsf{D}_{q_n}^{(m)} &\triangleq \mathbb{E}_{\boldsymbol{X}^{(m)},\hat{\boldsymbol{X}}^{(m)}}[d(\boldsymbol{X}^{(m)}, \hat{\boldsymbol{X}}^{(m)}) \mid Q = q_n] \\
&= \frac{1}{\beta}\sum_{i=1}^{\beta} \mathbb{E}_{X_i^{(m)},\hat{X}_i^{(m)}}[d(X_i^{(m)}, \hat{X}_i^{(m)}) \mid Q = q_n] \\
&= \frac{1}{\beta}\sum_{i=1}^{\beta} \sum_{x_i^{(m)},\hat{x}_i^{(m)} \in \{0,1\}} P_{X_i^{(m)},\hat{X}_i^{(m)}|Q}(x_i^{(m)}, \hat{x}_i^{(m)} \mid q_n) d(x_i^{(m)}, \hat{x}_i^{(m)}) \\
&\overset{(a)}{=} \frac{1}{\beta}\sum_{i=1}^{\beta} \sum_{x_i^{(m)},\hat{x}_i^{(m)} \in \{0,1\}} P_{X_i^{(m)}}(x_i^{(m)}) P_{\hat{X}_i^{(m)}|X_i^{(m)},Q}(\hat{x}_i^{(m)} \mid x_i^{(m)}, q_n) d(x_i^{(m)}, \hat{x}_i^{(m)}),
\end{aligned}$$

for an arbitrary requested file index $M = m$, where $d(x_i^{(m)}, \hat{x}_i^{(m)})$ is the Hamming distance (Section 2.1.5). Here, (a) is from the fact that Q and $\boldsymbol{W}$ are independent. To evaluate the conditional distribution $P_{\hat{X}_i^{(m)}|X_i^{(m)},Q}$ of a given answer scheme S_{q_n}, one first compute

$$P_{\boldsymbol{X}^{(m)},\hat{\boldsymbol{X}}^{(m)}|Q} = \sum_{\boldsymbol{x}^{(1)},\dots,\boldsymbol{x}^{(m-1)},\boldsymbol{x}^{(m+1)},\dots,\boldsymbol{x}^{(\mathsf{M})}} \sum_{\hat{\boldsymbol{x}}^{(1)},\dots,\hat{\boldsymbol{x}}^{(m-1)},\hat{\boldsymbol{x}}^{(m+1)},\dots,\hat{\boldsymbol{x}}^{(\mathsf{M})}} P_{\boldsymbol{W},\hat{\boldsymbol{W}}|Q}(\boldsymbol{w}, \hat{\boldsymbol{w}} \mid q_n),$$

and then obtain

$$P_{X_i^{(m)},\hat{X}_i^{(m)}|Q} = \sum_{x_1^{(m)},\dots,x_{i-1}^{(m)},x_{i+1}^{(m)},\dots,x_\beta^{(\mathsf{M})}} P_{\boldsymbol{X}^{(m)},\hat{\boldsymbol{X}}^{(m)}|Q}(\boldsymbol{x}^{(m)}, \hat{\boldsymbol{x}}^{(m)} \mid q_n).$$

The average distortion for a given requested file index $M = m$ is then given by

$$\mathsf{D}^{(m)} = \sum_{n=1}^{\alpha} P_{Q|M}(q_n \mid m)\mathsf{D}_{q_n}^{(m)} = \sum_{n=1}^{\alpha} p_{m,q_n}\mathsf{D}_{q_n}^{(m)},$$

and the average distortion for the whole scheme is equal to

$$f_{\text{D-Loss}} = \mathbb{E}_M[\mathsf{D}^{(M)}] = \sum_{m=1}^{\mathsf{M}} p_m \mathsf{D}^{(m)}. \tag{4.34}$$

The download rate is defined as the *per-symbol information download rate* (see Equation 3.19). Given a query $Q = q_n$ the server encodes $\mathbf{W}$ into $\hat{\mathbf{W}}$ with the rate defined as

$$\frac{\mathsf{H}(\hat{\mathbf{W}} \mid Q = q_n)}{\beta}, \tag{4.35}$$

calculated from $P_{\hat{\boldsymbol{W}}|Q}(\hat{\boldsymbol{w}} \mid q_n)$ corresponding to the weights of the answer model. The overall efficiency is measured in terms of the average per-symbol information download rate over all random queries, i.e., the rate of the entire scheme is defined as

$$f_{\text{R-Loss}} = \frac{1}{\beta}\sum_{n=1}^{\alpha} P_Q(q_n)\mathsf{H}(\hat{\mathbf{W}} \mid Q = q_n).$$

For leakage three different measures were tested in this implementation, namely mean square error, cross-entropy and a general α-loss function (first introduced in Equation 3.15). The mean square error function is defined as

$$f_{\text{MSE-Loss}} = \frac{1}{\mathsf{M}}\sum_{i=1}^{\mathsf{M}}(M_i - \hat{M}_i)^2,$$

while the cross-entropy loss is defined as

$$f_{\text{CE-Loss}} = -\sum_{i=1}^{\mathsf{M}} M_i \cdot \log(\hat{M}_i).$$

Here $\hat{M}$ is the adversary estimate of M given a query Q, and the log is of base 2.

The third leakage measure tested was a general α-loss function, first introduced in Section 3.1, defined as

$$f_{\text{alpha-Loss}} = \sum_{i=1}^{\mathsf{M}} \frac{\alpha}{\alpha - 1}(M_i(1 - \hat{M}_i^{1-\frac{1}{\alpha}}) + (1 - M_i)(1 - (1 - \hat{M}_i^{1-\frac{1}{\alpha}}))),$$

for a constant $\alpha > 1$.

The total loss function combines the three losses and also, similar to the GAP implementation, scales the distortion- and rate loss according to two input hyper-parameters ζ and η

$$f_{\text{T-Loss}} = (\zeta \cdot f_{\text{D-Loss}}) - f_{\text{L-Loss}} + (\eta \cdot f_{\text{R-Loss}}),$$

where $f_{\text{L-Loss}}$ denotes the loss function that we will use for leakage. Training of the model is done by minimizing the described loss through gradient descent. The next section describes the algorithm used for training the LWPIR model.

4.2.3 Training Algorithm

The training of the LWPIR model is done with the following algorithm.

```
Algorithm 3: LWPIR model training algorithm
input  : Dataset $\mathcal{D}$, leakage parameter $L$, distortion parameter $D$, rate
         parameter $R$, iteration number $T$, minibatch size $J$
output: Optimal privatizer and answer parameters $\theta_{\text{p}}, \theta_{\text{ans}}$
procedure ALERNATE MINIMAX($\mathcal{D}$, L, D, R, T, J)
    Initialize $\theta^1_{\text{p}}$, $\theta^1_{\text{ans}}$ $\theta^1_{\text{adv}}$
    for number of training iterations do
        for k steps do
            Sample minibatch of J file indexes $\{m_{(1)}, \dots, m_{(J)}\}$ drawn
              according to the the marginal distribution $p_m = \frac{1}{\mathsf{M}} = \frac{1}{2}$
            Generate the query probabilities $\{P_Q(q_{(1)}), \dots, P_Q(q_{(J)})\}$ via
              $P_Q(q_{(i)}) = g(m_{(i)}, q_{(i)}; \theta^t_{\text{p}})$
            Compute the descent direction $\nabla_{\theta_{\text{adv}}} \ell(\theta^t_{\text{adv}}, \theta^t_{\text{p}})$, where
                $\ell(\theta_{\text{adv}}, \theta^t_{\text{p}}) = f_{\text{MSE-Loss}}/f_{\text{CE-Loss}}/f_{\text{alpha-Loss}}$
            Update the parameter $\theta^{t+1}_{\text{adv}}$ for the adversary
                $\theta^{t+1}_{\text{adv}} = \max\limits_{\theta_{\text{adv}}} -\frac{1}{J}\sum_{i=1}^{J} \ell(h(P_Q(q_{(i)}); \theta_{\text{adv}}), m_{(i)})$
        Sample new minibatch of J file-indexes $\{m_{(1)}, \dots, m_{(J)}\}$
        Generate the query probabilities $\{P_Q(q_{(1)}), \dots, P_Q(q_{(J)})\}$ via
          $P_Q(q_{(i)}) = g(m_{(i)}, q_{(i)}; \theta^t_{\text{p}})$
        Generate the answer probabilities $\{P_{\hat{\boldsymbol{W}}|Q}(\hat{\boldsymbol{w}} \mid q_1), \dots, P_{\hat{\boldsymbol{W}}|Q}(\hat{\boldsymbol{w}} \mid q_j)\}$
          via $P_{\hat{\boldsymbol{W}}|Q}(\hat{\boldsymbol{w}} \mid q_i) = ans(P_{\boldsymbol{W}}(\boldsymbol{w}), P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}, q_{(i)}); \theta^t_{\text{ans}})$
        Compute the descent directions $\nabla_{\theta_{\text{p}}} \ell(\theta^t_{\text{p}}, \theta^{t+1}_{\text{adv}})$ and $\nabla_{\theta_{\text{ans}}} \ell(\theta^t_{ans}, \theta^t_{\text{p}})$,
          where $\ell(\theta^t_{\text{p}}, \theta^{t+1}_{\text{adv}})$ and $\ell(\theta^t_{\text{ans}}, \theta^t_{\text{p}})$ is the $T_{\text{loss}}$ subject to
                $f_{\text{D-Loss}} \leq \mathsf{D}$,
                $f_{\text{R-Loss}} \leq \mathsf{R}$, and
                $f_{\text{L-Loss}} \leq \mathsf{L}$
        Update the parameters $\theta^{t+1}_{\text{p}}$ and $\theta^{t+1}_{\text{ans}}$ for the privatizer and answer
          models
        $\theta^{t+1}_{\text{p}}, \theta^{t+1}_{\text{ans}} = \max\limits_{\theta_{\text{p}}, \theta_{\text{ans}}} - \sum_{i=1}^{J}$
          $f_{\text{T-Loss}}(P_{\hat{\boldsymbol{W}}|Q}(\hat{\boldsymbol{w}} \mid q_{(i)}), P_{\hat{\boldsymbol{W}}|\boldsymbol{W},Q}(\hat{\boldsymbol{w}} \mid \boldsymbol{w}, q_{(i)}), P_Q(q_{(i)}), m_{(i)})$
        Exit if solution converged
    return $\theta^{t+1}_{\text{p}}, \theta^{t+1}_{\text{ans}}$
```

Here $g()$ notates the privatizer function, $h()$ the adversary and $ans()$ the answer function. The algorithm follows the same principles as the algorithm presented in Algorithm 1 for training GAN models, where the algorithm alters between training the competing neural networks. Each training iteration consists of an inner loop where the adversary is trained on a fixed privatizer k times, and then the privatizer and answer models are trained on a fixed adversary once. Each weight update to the privatizer and answer model is subject to the three constraints regarding leakage (L), distortion (D) and rate (R). As introduced in the previous section (Section 4.2.2) the two hyperparameters ζ and η are used to scale the distortion and rate loss. The training finishes after T-iterations or if the model converges to a solution.

Chapter 5

Results and Findings

This chapter displays the results from the two implementations introduced in the previous chapter. First Section 5.1 presents the achieved results for the thesis implementation of the binary GAP model and compares them with both the paper implementation and the theoretical result for the scheme, as presented in [2]. Then Section 5.2 presents and discusses the results for the LWPIR model and compares them with the theoretical optimal schemes presented in [20,21]. Note that in [20,21] the download rate is quantified by the so-called *operational rate*. However, it can be shown that the the proposed schemes are also optimal in terms of the considered information rate in this thesis.

5.1 GAP Results

Some of the main motivations behind implementing this binary GAP model was to get a better understanding of the implementation details, and hopefully be able to achieve similar results to those presented in [2]. The model was trained using a synthetic dataset in which $Y = X \oplus N$, where $N \in \{0, 1\}$ is a random variable independent of X and following a *Bernoulli distribution* [24]. The dataset was generated with 10,000 training samples and 2000 test samples, following the *Bernoulli distribution* for (p, q) equal to (0.5, 0.25). Both the privatizer and adversary were trained using the Adam optimizer (Equation 2.49) with a learning rate of 0.005 and minibatch size of 400. For the penalty constraint p_t, introduced in Section 4.1.2, different initial values and scales were used for different result regions. The test results are for fully trained privatizers, limited by different distortion constrained, put up against MAP adversaries, as described in Section 4.1.1.

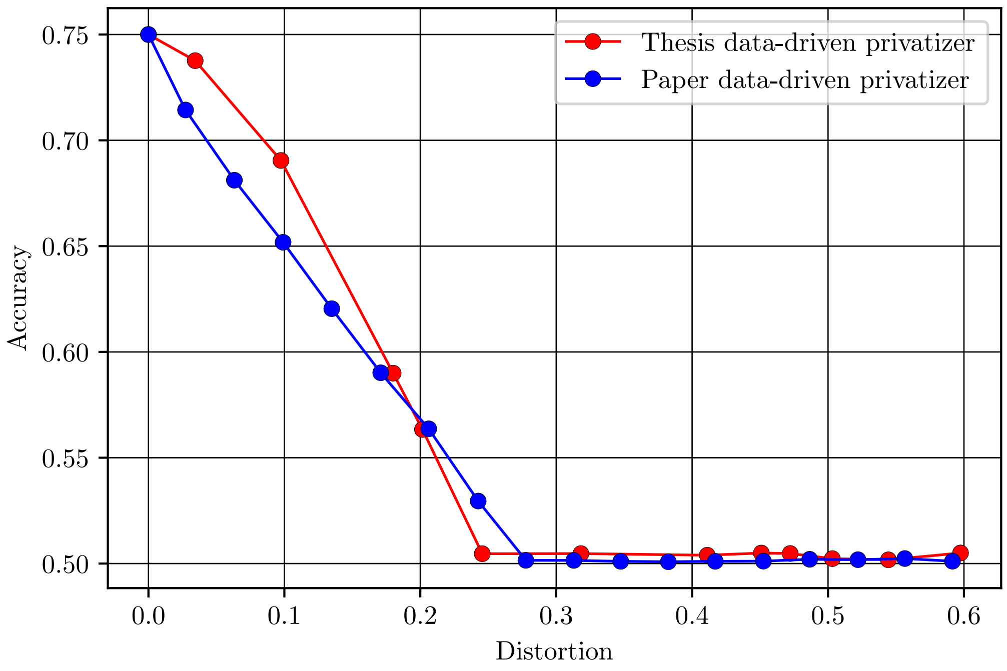


Figure 5.1: Privacy-distortion trade-off for binary data model. Performance of privacy mechanisms against MAP adversary for $p = 0.5$. Comparing implementation from [2] with model implemented in this thesis. Numerical results in Table A.1 and Table A.2.

Starting with comparing the thesis implementation with the paper model, seen in Figure 5.1. As seen from the figure it is apparent that, although the results of the thesis model is close to the results presented in the paper, there are still result regions with discrepancies. For certain distortions regions, such as around 0.15, it was especially challenging to obtain any results. Reasons for this could be due to not using the correct parameters, such as the for the penalty constraint p_t, and perhaps some randomness in the model. There is also an inherent difficulty in training neural networks in a GAN fashion because of the synchronization characteristics (see Section 2.2.8). The numerical thesis and paper results can be seen in Table A.1 and Table A.2.

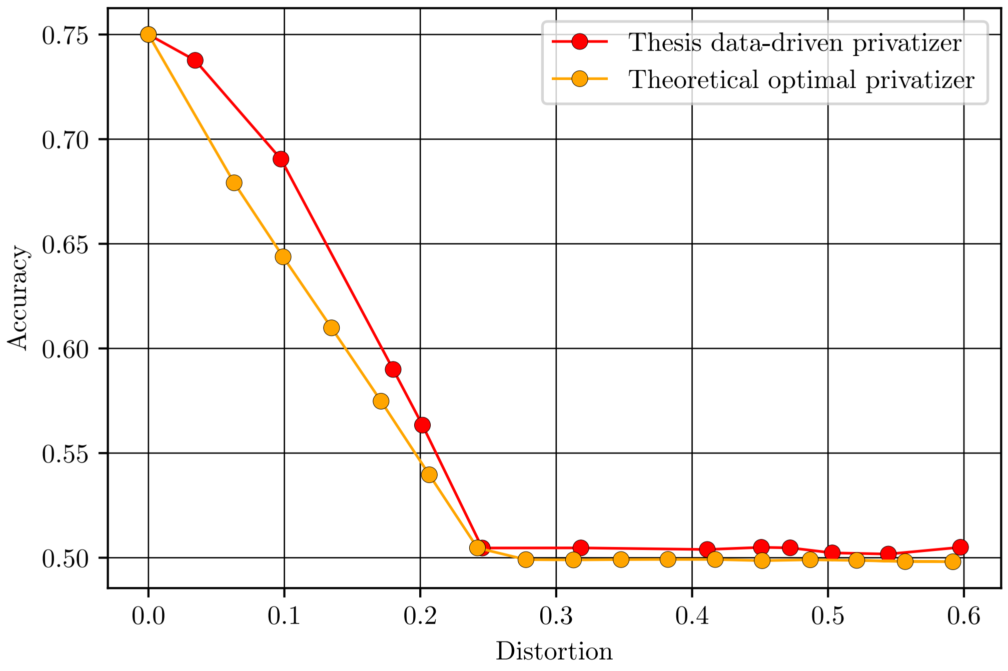


Figure 5.2: Privacy-distortion trade-off for binary data model. Performance of privacy mechanisms against MAP adversary for $p = 0.5$. Comparing theoretical optimal results (according to Equation 4.4) with results from model implemented in this thesis. Numerical results in Table A.1 and Table A.3.

Even though there are some discrepancies between the thesis- and paper implementation, the model still achieves some promising results. Figure 5.2 compares the thesis implementation against the optimal theoretical privatizer. Disregarding the regions where the model lacks results, the thesis implementation is relatively close to the theoretical optimal results. Overall implementing the model gave valuable insight into the theoretical foundations of the privacy perceiving mechanism presented in [2], and also essential experience in implementing a neural network in a GAN fashion. The numerical theoretical results can be seen in Table A.1 and Table A.3.

5.2 LWPIR Results

The goal for implementing the LWPIR model was to explore the opportunity of finding LWPIR schemes using neural networks. Ideally the implemented model should be fully general regarding the number of files and file sizes, however as already stated, this implementation considers the case for a single server storing two files each containing one binary bit. The entries are considered to be independent and identically distributed (i.i.d.) according to $\Pr[X_i^{(m)} = 0] = P[X_i^{(m)} = 1] = {}^1\!/_2$. It is also assumed that the files are independent, giving

$$P_X(x^{(1)}, x^{(2)}) = \frac{1}{2^{\mathsf{M}\beta}} = \frac{1}{2^{2\cdot 1}} = \frac{1}{4}. \tag{5.1}$$

The marginal distribution of the requested file index $M = m$ is denoted as p_m, where $m \in [1 : \mathsf{M}]$, and is assumed to be

$$p_m = \frac{1}{\mathsf{M}} = \frac{1}{2}. \tag{5.2}$$

The final model is evaluated by comparing the rate, distortion and leakage against a theoretical optimal LWPIR scheme presented in [20, 21]. However, before training and evaluating the complete model, it was interesting to first experiment with a model with a fixed optimal privatizer. This was done to asses the feasibility of training the complete model, since training with a fixed optimal privatizer should easier evaluate to a scheme closer to a theoretical optimal one. The weights for the optimal privatizer are fixed according to Theorem 2 [20].

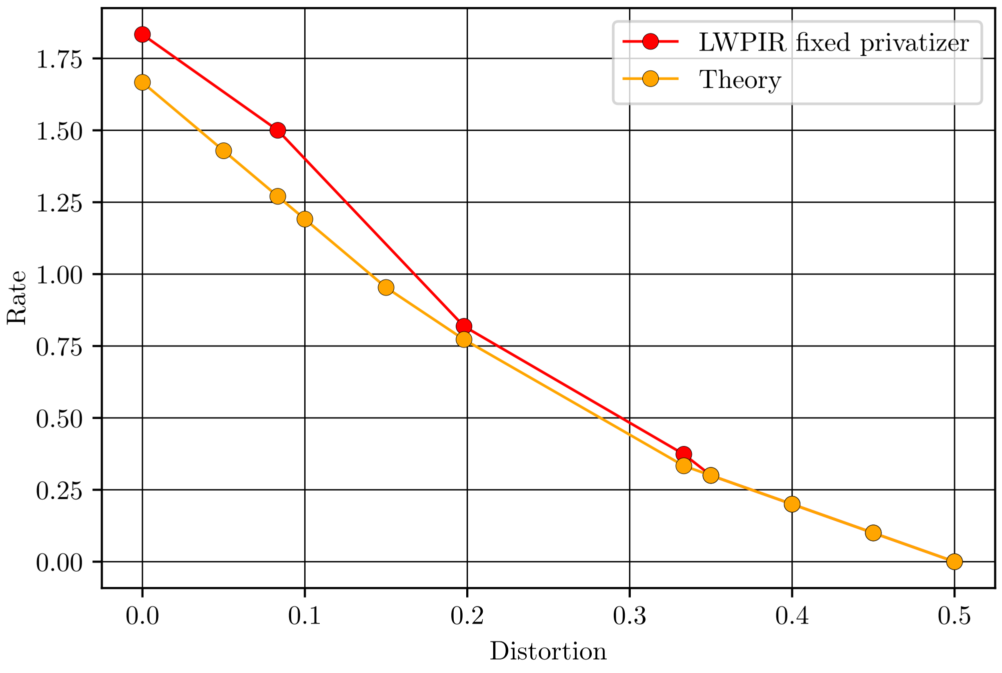


Figure 5.3: Rate-distortion trade-off for LWPIR model with fixed optimal privatizer. Comparing achieved results with theoretical results for leakage ${}^2\!/_3$ under various rate and distortion constraints. Numerical results in Table A.4.

Figure 5.3 shows the best achieved results for the LWPIR model with a fixed optimal privatizer for leakage $^2/_3$ compared with the theoretical optimal scheme from Theorem 2 [20]. The model was trained using a synthetic dataset containing 10 000 samples of file indexes M following the above mentioned distribution (5.2). Both the adversary and the answer network were trained using the Adam optimizer (Equation 2.49) with a learning rate of 0.01. For the hyperparameters ζ and η, described in Section 4.2.2, different initial values and increments were used for different result regions. The loss functions used are described in Section 4.2.2, with the leakage formulated as $f_{\text{MSE-Loss}}$ (Equation 5.1). The numerical results can be seen in Table A.4. Generally the results seem promising, with many result regions relatively close to the theoretical optimal ones. However, similar to the GAP results, for some results regions there were difficulties to obtain any good results - such as for distortion ranging from 0.2 to 0.3. The results could potentially be improved by choosing better values for the hyperparameters ζ and η, by altering other parameters such as the learning rate, or by using other loss functions. This is further explored in the evaluation of the complete model.

As mentioned earlier, three different leakage loss formulations where considered for training of the complete LWPIR model, namely mean square error (MSE), cross-entropy and a general α-loss (Section 4.2.2). The model was trained under various leakage, loss and rate constraints for each loss formulation. Only the best achieved results are included in this section, with the less optimal results discarded. The results are plotted, similar to the results for LWPIR model with the fixed privatizer, as a tradeoff between the rate and distortion for a given leakage. Even though the leakage is not directly specified as is the case with a fixed privatizer, the model is trained with leakage constraints and the results with similar leakage are plotted together to illustrate the difference against the theoretical optimal rate-distortion tradeoff. The training is done using the Adam optimizer with a learning rate of 0.0001 for the adversary and 0.009 for the privatizer and answer networks. Different sized learning rates were used since this showed more promising results.

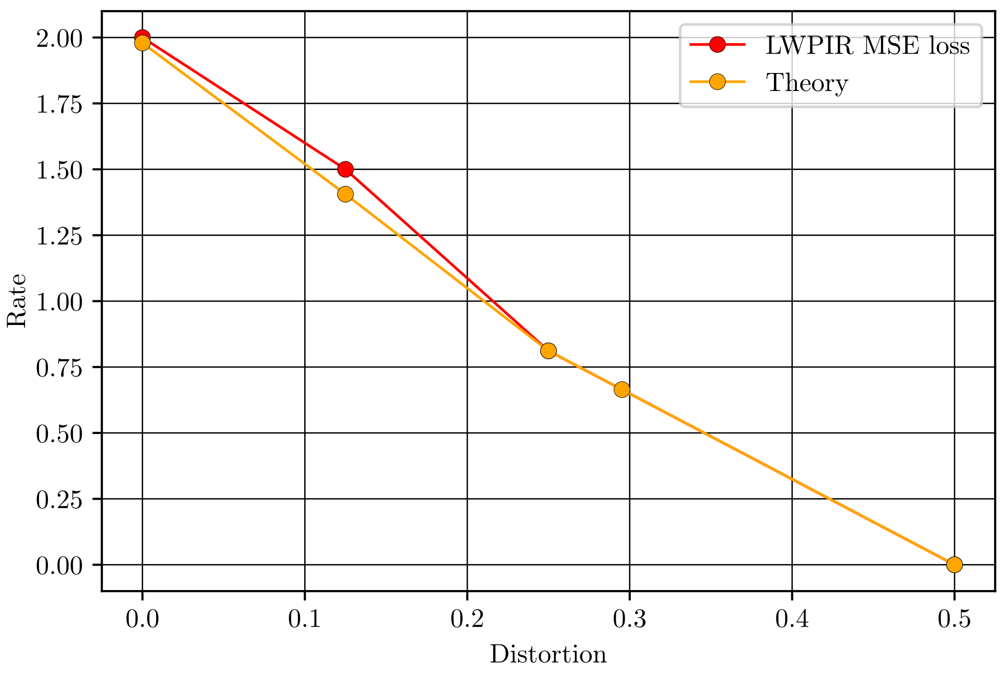


Figure 5.4: Rate-distortion trade-off for LWPIR model trained with MSE loss. Comparing achieved results with theoretical results for leakage $^1/_2$ under various rate and distortion constraints. Numerical results in Table A.5.

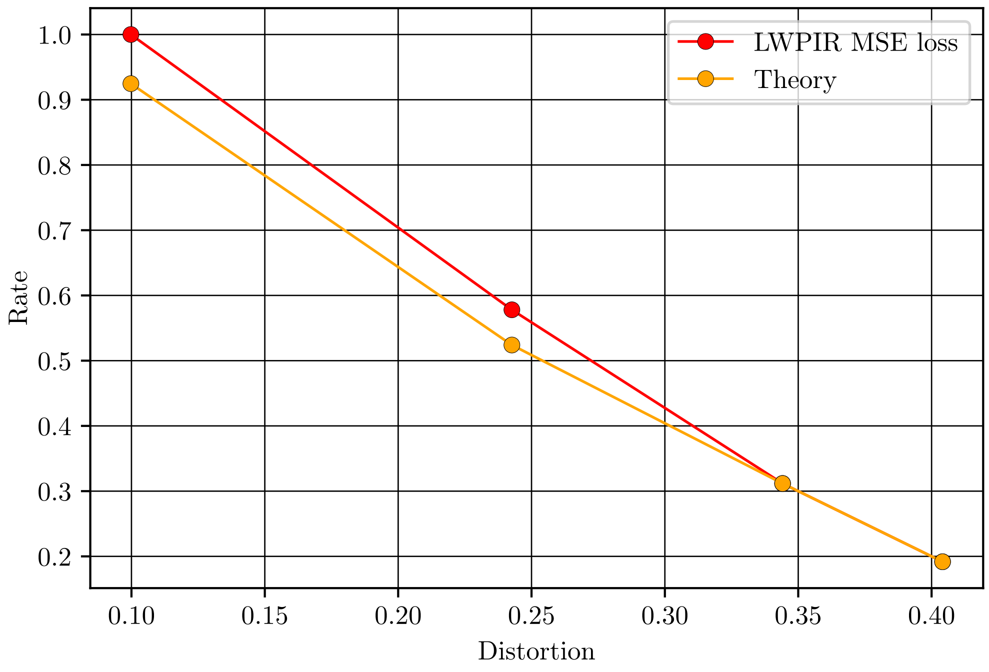


Figure 5.5: Rate-distortion trade-off for LWPIR model trained with MSE loss. Comparing achieved results with theoretical results for leakage in region 0.7-0.8 under various rate and distortion constraints. Numerical results in Table A.6.

Starting with the results for the MSE loss, Figure 5.4 and Figure 5.5 show results for leakage equal 1/2 and in the range 0.7-0.8 respectively. For leakage 1/2 four different optimal schemes were achieved for distortion 0, 0.125, 0.25 and 0.5, with rate equal to the theoretical optimal schemes. Also for the results plotted in Figure 5.5 for leakage 0.7-0.8, the model achieved several close to optimal schemes. It proved however to be a significant challenge to achieve schemes with certain distortion-rate trade-offs, since the model very often evaluated to the same schemes even with different distortion and rate constraints.

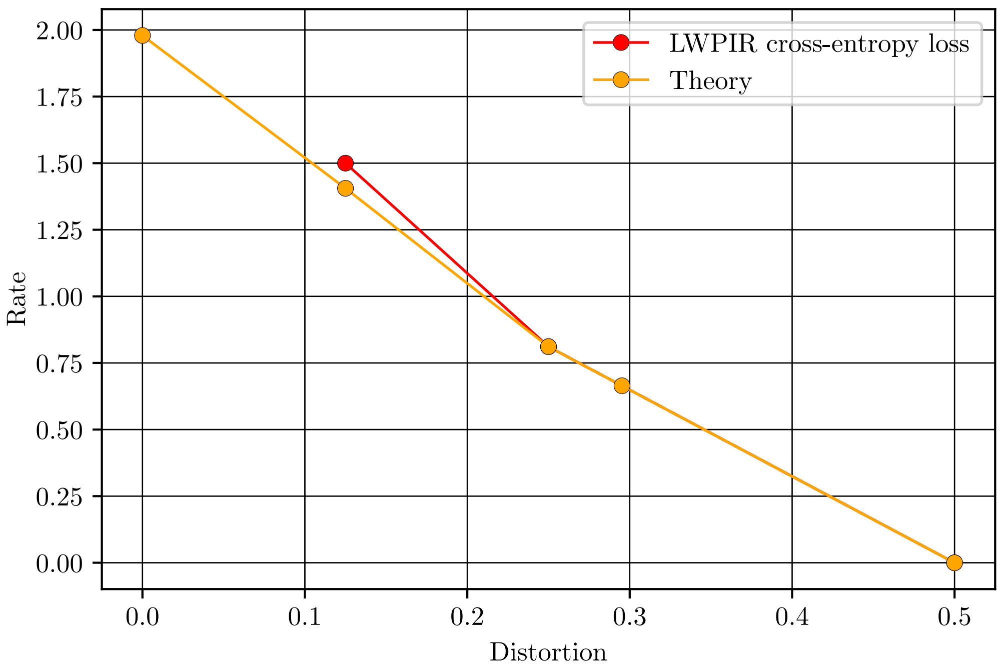


Figure 5.6: Rate-distortion trade-off for LWPIR model trained with cross-entropy loss. Comparing achieved results with theoretical results for leakage $^1/_2$ under various rate and distortion constraints. Numerical results in Table A.7.

Moving on to the results using the cross-entropy loss function, Figure 5.6 show the achieved results for leakage equal to $^1/_2$. As seen from the figure and the numerical results in Table A.7, the different schemes achieved were optimal but it was challenging to get schemes for other rate-distortion-leakage tradeoff. Similar to using the MSE loss it was very difficult to achieve schemes with other distortion-rate trade-offs than those presented, with no achieved results for any leakage other than $^1/_2$. Also since the initial results using other loss functions such as MSE proved more promising, the amount of experimenting with training using cross-entropy was limited compared to MSE.

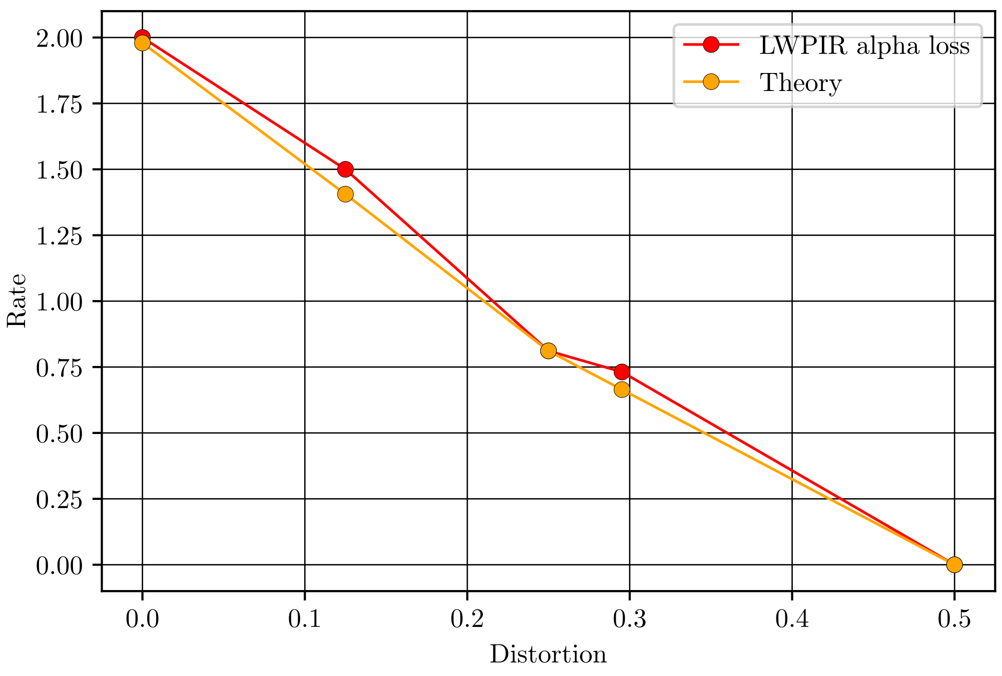


Figure 5.7: Rate-distortion trade-off for LWPIR model trained with alpha loss. Comparing achieved results with theoretical results for leakage $1/2$ under various rate and distortion constraints. Numerical results in Table A.8.

The last Figure 5.7 shows the results achieved using the α-loss (described in Section 4.2.2). Similar to the results for the other loss functions, a few optimal results were achieved, but it was still difficult to achieve schemes for other rate-distortion-leakage tradeoff. Compared to the other two loss functions, α-loss provides flexibility in the opportunity of different α values potentially resulting in improved training. It was however still difficult to obtain other optimal schemes than the ones presented. The next chapter further discusses the results of the thesis as a whole and potential for future work.

Chapter 6

Conclusions

The initial objective of this thesis was to investigate the use of neural networks for Lossy Weakly-Private Information Retrieval. Firstly, by examining previous work done within the field through implementing the *Context-Aware Generative Adversarial Privacy* model presented in [2], and then secondly explore the use of generative adversarial nets to find LWPIR schemes as presented in [20, 21].

The implementation and testing of the GAP model gave valuable insight into both the technical and practical aspects of using neural networks in a privacy perceiving context. The achieved GAP results indicated the usefulness of privacy mechanisms modeled by neural networks and also illustrated some of the challenges concerning training generative adversarial nets. Even though the GAP thesis results were considerably close to the results reported in [2], there were still some notable discrepancies. As discussed earlier, the reasons for the discrepancies could be due to not choosing the optimal hyperparameters like the penalty constraint or other implementation details, such as the optimizer and learning rate. Nevertheless, the GAP implementation gave a solid starting point for implementation of the new *LWPIR GAN* model introduced in this thesis.

The implementation of the LWPIR model turned out to be a more significant challenge than initially anticipated. As presented in the previous chapter, the achieved results were promising regarding being close to theoretical optimal ones, but it was challenging to get other schemes for different tradeoff than those presented. However, the results still display the potential for using neural networks as a tool for finding LWPIR schemes. It is conceivable that, given the suitable configurations (loss formulation, optimizer, hyperparameter values, etc.), the LWPIR model could achieve a wider range of optimal results than those presented in this thesis. Also, as explained earlier, even though this implementation only considers the more narrow case of a single server storing two files, each containing one bit, most of the theoretical foundation on which the model is built considers a broader, more general setting. Therefore, a natural continuation is to generalize the LWPIR GAN model to handle an arbitrary number of files and file sizes.

Appendices

Appendix A

Numerical Results

Distortion	Accuracy thesis data-driven privatizer
0	0.750002
0.034388	0.737651
0.097483	0.690472
0.180039	0.58997
0.201596	0.563361
0.245595	0.504647
0.318167	0.504702
0.411162	0.503944
0.450934	0.504967
0.47214	0.504739
0.503149	0.502324
0.544412	0.501762
0.5975	0.505

Table A.1: Privacy-distortion tradeoff for binary data GAP model implemented in this thesis. Performance of privacy mechanisms against MAP adversary for p = 0.5

Distortion	Accuracy paper data-driven privatizer
0	0.75
0.027281679	0.714335308
0.063213501	0.681137895
0.099062917	0.651809451
0.134870147	0.620434747
0.170898608	0.590148022
0.20620435	0.563672218
0.242604788	0.529536602
0.277703253	0.501515779
0.312964817	0.501486787
0.347475774	0.501040174
0.38248791	0.500846628
0.417033043	0.501053788
0.452457856	0.501130527
0.486507298	0.502013065
0.522028408	0.501817285
0.55650894	0.502367098
0.591561614	0.501094143

Table A.2: Privacy-distortion tradeoff for binary data GAP model presented in [2]. Performance of privacy mechanisms against MAP adversary for p = 0.5

Distortion	Accuracy theoretical optimal GAP privatizer
0	0.75
0.063049042	0.679136207
0.099075503	0.643773145
0.134706484	0.609837273
0.171134877	0.574786121
0.206645961	0.539634991
0.24205293	0.504702134
0.277739436	0.499089642
0.312749723	0.498956643
0.347751155	0.499113644
0.382246194	0.499188355
0.416998681	0.499164954
0.451567202	0.498606182
0.486944421	0.499031188
0.591908538	0.498083472
0.556791444	0.498168296
0.521150654	0.498737476

Table A.3: Privacy-distortion tradeoff for binary data model from [2]. Performance of theoretical optimal privacy mechanisms against MAP adversary for p = 0.5

Distortion	Rate theory	Rate LWPIR model	Difference
0	1.666666667	1.833333373	0.1666667064
0.05	1.428922292	-	-
0.08333333	1.270426057	1.50000006	0.2295740023
0.1	1.191177916	-	-
0.15	0.9534335413	-	-
0.19798252	0.7725619493	0.8184944391	0.04593248985
0.3333333	0.3333334415	0.3735464662	0.04021302473
0.34999996	0.30000008	0.3000000715	0.000000008474426993
0.39999995	0.2000001	0.2000000775	0.000000022513962
0.44999978	0.10000044	0.1000001729	0.000000267146531
0.4999998	0.0000004	2.60E-07	0.0000001404124201

Table A.4: Distortion-accuracy trade-off for LWPIR model with fixed optimal privatizer with leakage $2/3$

Leakage	Distortion	Rate theory	Rate LWPIR model	Difference
0.510389	0	1.979221	2	0.020779
0.5	0.125	1.405639	1.5	0.094361
0.5	0.25	0.811279	0.811279	0
0.5	0.5	0	0	0

Table A.5: Distortion-accuracy trade-off for LWPIR model with MSE leakage loss with leakage $1/2$

Leakage	Distortion	Rate theory	Rate LWPIR model	Difference
0.800436	0.099782	0.924676	1	0.075324
0.7492	0.24263562	0.524	0.578	0.054
0.81173	0.34413117	0.3117376	0.3117376	0
0.69176	0.40412	0.191761	0.191761	0

Table A.6: Distortion-accuracy trade-off for LWPIR model with MSE leakage loss with leakage in range 0.7-0.8

Leakage	Distortion	Rate theory	Rate LWPIR model	Difference
0.5	0.125	1.405639	1.5	0.094361
0.5	0.25	0.811279	0.811279	0
0.5	0.5	0	0	0

Table A.7: Distortion-accuracy trade-off for LWPIR model with cross-entropy leakage loss with leakage $1/2$

Leakage	Distortion	Rate theory	Rate LWPIR model	Difference
0.5	0	1.979221	2	0.020779
0.5	0.125	1.405639	1.5	0.094361
0.5	0.25	0.811279	0.811279	0
0.5	0.295251	0.664432	0.731324	0.066892
0.5	0.5	0	0	0

Table A.8: Distortion-accuracy trade-off for LWPIR model with alpha loss for leakage with leakage $1/2$

Appendix B

Theorems

Theorem 2 [20]. *Optimal tradeoff for* $\mathsf{M} = 2$, $\beta = 1$ *under various distortion* D *and leakage* L *constraints. The formulation for the optimal information rate* $R^*_{inf}(D, L)$ *and operational rate* $R^*_{op}(D, L)$ *(see Section 3.2) are*

$$R^*_{inf}(D, L) = \begin{cases} 4D(\mathsf{H}_b(1/4) - 2) + (3 - 2L), & \text{for } 0 \leq D \leq \frac{1-L}{2}, \\ 4\mathsf{H}_b(1/4)(1 - D - L) + 2L + 1, & \text{for } \frac{1-L}{2} \leq D \leq 1 - L, \\ -2D + 1, & \text{for } 1 - L \leq D \leq \frac{1}{2}, \end{cases}$$

$$R^*_{op}(D, L) = \begin{cases} -4D + (3 - 2L), & \text{for } 0 \leq D \leq 1 - L, \\ -2D + 1, & \text{for } 1 - L \leq D \leq \frac{1}{2} \end{cases}$$

Below are three schemes achieving these optimal curves, simultaneously for R^*_{op} *and* R^*_{inf}*.*

1) $0 \leq D \leq (1-L)/2$*: The scheme* $\mathcal{Q} = \{q_1, q_3, q_5, q_6\}$ *and the description is as follows:*

q	$P(q\|1)$	$P(q\|2)$
q_1	$2 - 2L - 4D$	$2 - 2L - 4D$
q_5	$4D$	$4D$
q_3	$2L - 1$	0
q_6	0	$2L - 1$

2) $(1-L)/2 \leq D \leq 1 - L$*: The scheme* $\mathcal{Q} = \{q_3, q_5, q_6, q_7\}$ *and the description is as follows:*

q	$P(q\|1)$	$P(q\|2)$
q_5	$4 - 4D - 4L$	$4 - 4D - 4L$
q_3	$2L - 1$	0
q_6	0	$2L - 1$
q_7	$4D + 2L - 2$	$4D + 2L - 2$

3) $1 - L \leq D \leq 1/2$*: The scheme* $\mathcal{Q} = \{q_3, q_6, q_7\}$ *and the description is as follows:*

q	$P(q\|1)$	$P(q\|2)$
q_3	$1 - 2D$	0
q_6	0	$1 - 2D$
q_7	$2D$	$2D$

Bibliography

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014.

[2] C. Huang, P. Kairouz, X. Chen, L. Sankar, and R. Rajagopal, “Context-aware generative adversarial privacy,” *Entropy*, vol. 19, p. 656, Dec 2017.

[3] T. Cover, J. Thomas, and J. W. . Sons, *Elements of Information Theory*. Online access: EBSCO Computers & Applied Sciences Complete, Wiley, 1991.

[4] G. Smith, “On the foundations of quantitative information flow,” in *Foundations of Software Science and Computational Structures*, pp. 288–302, 03 2009.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Adaptive Computation and Machine Learning series, MIT Press, 2016.

[6] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[7] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*, p. 318–362. Cambridge, MA, USA: MIT Press, 1986.

[9] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] A. Krizhevsky, “Learning multiple layers of features from tiny images,” tech. rep., Citeseer, 2009.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] J. Susskind, A. Anderson, and G. E. Hinton, “The toronto face dataset,” tech. rep., Technical Report UTML TR 2010-001, U. Toronto, 2010.

[13] S. ARIMOTO, “Information measures and capacity of order α for discrete memoryless channels,” *Topics in Information Theory*, pp. 41–52, 1977.

[14] C. Cai and S. Verdú, “Conditional rényi divergence saddlepoint and the maximization of α-mutual information,” *Entropy*, vol. 21, pp. 1–6, 10 2019.

[15] A. Rényi *et al.*, "On measures of entropy and information," in *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, The Regents of the University of California, 1961.

[16] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pp. 41–50, 1995.

[17] H. Lin, S. Kumar, E. Rosnes, A. G. i. Amat, and E. Yaakobi, "Weakly-private information retrieval," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1257–1261, 2019.

[18] I. Samy, R. Tandon, and L. Lazos, "On the capacity of leaky private information retrieval," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 1262–1266, 2019.

[19] H. Y. Lin, S. Kumar, E. Rosnes, A. G. i. Amat, and E. Yaakobi, "The capacity of single-server weakly-private information retrieval," in *2020 IEEE International Symposium on Information Theory (ISIT)*, pp. 1053–1058, 2020.

[20] Y. Yakimenka, H.-Y. Lin, E. Rosnes, and J. Kliewer, "Optimal rate-distortion-leakage tradeoff for single-server information retrieval." to be presented at 2021 IEEE International Symposium on Information Theory (ISIT'21), 2021.

[21] Y. Yakimenka, H.-Y. Lin, E. Rosnes, and J. Kliewer, "Optimal rate-distortion-leakage tradeoff for single-server information retrieval." submitted to IEEE Journal on Selected Areas in Communications, June 2021.

[22] G. Barthe and B. Kopf, "Information-theoretic bounds for differentially private mechanisms," in *2011 IEEE 24th Computer Security Foundations Symposium*, pp. 191–204, 2011.

[23] I. Issa, A. B. Wagner, and S. Kamath, "An operational approach to information leakage," *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1625–1657, 2020.

[24] Y. Dodge, "Bernoulli distribution," in *The Concise Encyclopedia of Statistics*, (New York, NY), pp. 36–37, Springer New York, 2008.