Information Bottleneck Methods for Fairness and Privacy in Machine Learning

by

Adam Gronowski

A thesis submitted to the
Department of Mathematics and Statistics
in conformity with the requirements for
the degree of Master of Applied Science

Queen's University
Kingston, Ontario, Canada
September 2022

Abstract

Designing machine learning algorithms that are accurate yet fair, not discriminating based on any sensitive attribute, and also private, not revealing users' personal information, has become of paramount importance for society to accept the widespread use of artificial intelligence (AI) for critical applications. In this thesis, we investigate the use of variational Information Bottleneck (IB) methods for fair and private machine learning.

We present a novel fair representation learning method termed Rényi Fair Information Bottleneck (RFIB) which incorporates constraints for utility, fairness, and compactness of representation. In contrast to prior work, we consider both demographic parity and equalized odds as fairness constraints, allowing for a more nuanced satisfaction of both criteria. We use the Rényi divergence in developing a loss function involving classical IB measures and show that its parameter α provides an extra degree of freedom that results in performance benefits. Applying the method to image classification, we study the influence of the α parameter and two other tunable IB parameters on achieving utility/fairness trade-off goals, and evaluate it using various utility, fairness, and compound utility/fairness metrics on three different image datasets (EyePACS, CelebA, and FairFace), showing that RFIB outperforms current state-of-the-art approaches.

Furthermore, we investigate how the problem of privacy relates to the problem of fairness and present a related method to jointly improve fairness and privacy termed Rényi Fair and Private Information Bottleneck (RFPIB). Using the Rényi divergence and IB measures, we develop a loss function designed to improve both privacy and multiple fairness metrics while also ensuring utility. Experimenting on the CelebA and EyePACS datasets, we study trade-offs between fairness, privacy, and utility. We significantly outperform a baseline ResNet50 network and show that tuning the Rényi divergence's α parameter can be used to simultaneously achieve these three desired criteria.

Co-Authorship

I would like to acknowledge and thank my collaborators, Dr. Philippe Burlina and William Paul from the Johns Hopkins University Applied Physics Laboratory, particularly for their work on our joint papers [38, 37] which form Chapter 5 of this thesis.

Acknowledgments

I would like to thank my supervisors, Professor Fady Alajaji and Professor Bahman Gharesifard. My undergraduate research project with Prof. Alajaji is what first got me interested in research in this field, and his support and encouragement was instrumental in my decision to pursue graduate studies. Profs. Alajaji and Gharesifard are both outstanding supervisors. Their care, support, and guidance has made my time as a Master's student a very rewarding and positive experience.

I am also grateful to my collaborators Dr. Philippe Burlina and William Paul from the Johns Hopkins University Applied Physics Laboratory. They were both a pleasure to work with, helpful, knowledgeable, and insightful, and I am very appreciative of their assistance.

I would like to acknowledge the Centre for Advanced Computing at Queen's University for providing access to their computing cluster to run the experiments presented in this thesis, as well as Michael Hanlan and Sridhar Ravichandran for providing technical support.

Finally, I would like to thank Jennifer Read for her assistance throughout my degree.

Contents

List of Tables

List of Figures

Acronyms

AD	Adversarial Debiasing
CAI	Conjunctive Accuracy Improvement
CFB	Conditional Fairness Bottleneck
CNN	Convolutional Neural Network
DR	Diabetic Retinopathy
ELBO	Evidence Lower Bound
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
i.i.d.	Independent and Identically Distributed
IA	Intelligent Augmentation
IB	Information Bottleneck
ITA	Individual Topology Angle
KL	Kullback-Leibler

ReLU	Rectified Linear Unit
ResNet	Residual Neural Network
RFIB	Rényi Fair Information Bottleneck
RFPIB	Rényi Fair and Private Information Bottleneck
SGD	Stochastic Gradient Descent
UMAP	Uniform Manifold Approximation and Projection
VAE	Variational Autoencoder

Chapter 1

Introduction

Machine learning algorithms are used for a variety of high stake applications such as loan approvals, police allocation, admission of students, and disease diagnosis. In spite of their vast benefits, the use of automated algorithms that are not designed to also address potential bias and fairly serve members of diverse groups can lead to harm and exacerbate social inequities, while using algorithms that are not designed to preserve privacy can lead to privacy breaches where an adversary gains access to private, personal information. The problem of developing algorithms that are both *accurate* and *fair*, i.e., do not discriminate against individuals because of their gender, race, age, or other protected attributes, as well as *private*, i.e., do not reveal a user's personal information, has now become paramount to the deployment of production-grade AI systems that could be accepted and adopted by society as a whole.

Fair machine learning methods have been developed for multiple domains such as automated healthcare diagnostics and treatments delivery [56], natural language processing [10], and others [98, 76, 56, 13], while private machine learning methods include the Privacy Funnel [18, 64] and the Conditional Privacy Funnel [80]. One way to create such fair or private machine learning methods is through learning *fair representations* or *private representations* that can be used with existing machine learning architectures instead of the raw input data. These representations would allow for making accurate predictions while ensuring fairness or privacy. However, developing fair or private representations is difficult as it may involve trade-offs between fairness and accuracy or privacy and accuracy and is further complicated by the existence of various metrics for measuring fairness outcomes, often tailored towards different applications and settings.

1.1 Contributions

Firstly, our work entails the development of a fair representation learning method termed Rényi Fair Information Bottleneck (RFIB) that addresses a number of trade-offs. Unlike most prior studies that tend to focus on satisfying a single type of fairness constraint, we consider here how to jointly address and balance two of the arguably most important definitions for fairness, *demographic parity* and *equalized odds*. We also examine different classical trade-offs between fairness and utility (commonly measured via accuracy) arising as a result of interventions on models or data to make models more fair, which may yield decreased bias, but may also result in affecting utility. We study how these trade-offs are impacted by "compactness." More specifically, we develop a variational approach taking into account different information-theoretic metrics that balance the above two constraints on fairness with utility and compactness. We also show how to analytically simplify the resulting loss function and relate it to the *Information Bottleneck (IB)* principle [90], and then exploit bounds to compute suitable metrics.

Secondly, we extend this method to jointly improve privacy and fairness, developing a method we call Rényi Fair and Private Information Bottleneck (RFPIB). We derive a cost function that incorporates fairness, privacy, utility, and compactness constraints, using a variational approach that maintains the use of the Rényi divergence of order α to create an upper bound on the mutual information between the data and its representation. Applying the method to image classification, we experiment on the CelebA and EyePACS datasets, investigating trade-offs between fairness, privacy, and accuracy.[1]

We thus make the following novel contributions:

1. We develop a novel variational method, RFIB, that balances a triplet of objectives, consisting of utility (accuracy), fairness (itself balancing two types of fairness constraints), and compression/compactness. Specifically, in contrast to prior work on fairness which narrowly uses either the demographic parity or equalized odds constraints, our loss includes both types of constraints. We relate analytically the resulting loss function to the classical IB method.

2. Operationally, we derive an upper bound on the mutual information between the data and its representation in terms of the Rényi divergence of order α. We study the effect of this added flexibility on achieving a balance between fairness and accuracy.

3. We compare, using various datasets such as CelebA, EyePACS, and FairFace, with methods of record that intervene on the model or methods that intervene on training data. We show that our RFIB method overall performs best. We establish these comparisons via a number of metrics that measure utility and

[1] Code to reproduce our experiments can be found at `https://github.com/AGronowski/RFPIB`.

fairness individually or in a combined metric, including two different types of fairness constraints.

4. We extend RFIB to also improve privacy, developing the novel variational method of RFPIB that balances objectives consisting of utility, multiple fairness criteria, and privacy.

5. We examine how tuning the Rényi divergence's α parameter and three IB parameters affects trade-offs between fairness, utility, and privacy. We evaluate the method using metrics for privacy, multiple definitions of fairness, and utility (accuracy). We show that RFPIB is able to improve upon all three of these criteria compared to a baseline on the CelebA amd EyePACS datasets.

1.2 Literature Review

We discuss different fairness methods as well as information bottleneck methods and other related work.

1.2.1 Fairness Approaches

We summarize existing work on fair machine learning in three categories. For a broad-strokes categorization of fairness approaches, one can think along the lines of *interventions* made either on: a) the model output, b) the training data, or c) the model itself. Each of these are motivated by different inductive biases.

Interventions on Model Outputs Including Recalibration and Thresholding

The inductive bias here is that irrespective of the cause of bias, fairness can be addressed at the output of the model. Some of these methods intervene on the model output via altering the decision threshold so that equal odds constraints are achieved, as in [40] by selecting an operating point where the receiver operating characteristic curves for different populations intersect (or variations on this approach). Alternatively, [75] uses re-calibration across subpopulations to debias models. While these methods are often effective – especially in case of debiasing models operating on categorical data – they have limitations in that they do not make more consequent changes on the data and the model itself to address the root causes of bias, as was argued also in [40]. This may especially be an issue for image/video applications and is the reason why an alternate path is pursued herein.

Interventions on Data Including Generative Models and Style Transfer

Such methods proceed from the inductive bias that interventions be carried on data, since data imbalance is a potential cause of biased models. Methods that modify the training data perform various operations, ranging from censoring sensitive information in the image domain to making it blind to protected factors, performing data augmentation, or using re-weighting of the data to achieve re-balancing either via data re-sampling or reweighing of the loss function. Data augmentation methods use generative models or style transfer/image translation. Notable approaches using generative models and based on generative adversarial networks (GANs) include [50, 39, 72]. In [77], image translation between protected populations was used to achieve normalized appearances. Translation and style transfer methods however have limitations in that they tend to collapse to using only a single style, a form of mode collapse, which may be an issue if data representative of a subpopulation entails "variations of styles," such as in images. As will be discussed later, these methods are also related to interventions used for domain adaptation and use cases of distributional shift and prior shift. Alternatively, [47] uses variational autoencoders for augmentation. Other examples of such generative methods for fairness include [86]. The methods in [72] use instead adversarial training along with a form of gradient descent in latent space to manipulate images and generate more data.

Generation of data using all of the above generative approaches has limitations as it may be hard to control the exact image markers that correspond to a specific protected factor without changing other markers (a problem known as *entanglement*). Also generation of images for canonical domains (faces, retinas, chest X-rays) tends to be relatively easily accomplished at present [51], but it may be more complicated for other domains. Generative methods may also yield artifacts in synthetic images, which could lead to a decrease in performance and in overall utility when debiasing without necessarily achieving significant gains in fairness. Aligned with these observations, it is suggested in [72] that for this reason model-altering fairness methods may outperform data-altering methods via generative models. These limitations motivate our approach, which falls in the category of interventions to the model itself, described next.

Interventions on Models via Adversarial and Variational Approaches

Such methods are grounded on the inductive bias that dependence of the prediction on protected factors is a cause of lack of fair predictions; as a result, these methods generally aim to remedy this dependence at the encoding of the data, rendering them blind to protected factors.

Studies such as [93] and [100] use an adversarial network to penalize the prediction network if it could predict a protected factor. Other works employ adversarial representation learning to remove protected factor information from latent representations, including [21, 7, 63, 81, 104, 48]. Applications using this principle include [74] which uses adversarial learning to develop fair models of cardiovascular disease risk, while [29] explores the statistical properties of fair representation learning and [36] applies an adversarial approach for continuous features. Similar to our work, [88] employs an information-theoretic approach to learn fair representations. But in contrast to the above methods that are based on an adversarial approach, our method does not use adversarial training.

In addition to the above methods, there also exist non-adversarial methods that modify the model via incorporation of multiple constraints in a variational setting. Many of these are closely related to the IB method that we discuss next.

1.2.2 Information Bottleneck Methods

The IB method, originally proposed by Tishby *et al.* [90], is a method that seeks to develop representations that are both compact and expressive by minimizing and maximizing two mutual information terms. Alemi *et al.* first developed a variational approximation of the IB method by parameterizing it using neural networks and this was followed by multiple variations such as the nonlinear information bottleneck [57] and conditional entropy bottleneck [24]. Many recent generalizations of the IB method have been developed including [64, 89, 46, 3, 94] while [32, 97] investigated its connections to deep learning theory and privacy applications. Many applications of the IB method to machine learning are included in the special issue [26].

Techniques related to the information bottleneck have been used for fair representation learning. First proposed by [98], fair representation learning consists of mapping input data to an intermediate representation that remains informative but discards unwanted information that could reveal the protected sensitive factors. This is related to the IB problem and several works have explored the connection between the two, such as Ghassami *et al.* [27] and Rodríguez-Gálvez *et al.* [80] where fair representations are acquired through the minimization and maximization of various mutual information terms.

Our work is closest to [80] but we depart from it in several important ways, including through the use of a more general loss formulation, satisfying broader constraints of fairness, classifying images rather than focusing solely on tabular data, and entailing the use of Rényi divergence. There have been several recent works based on Rényi information measures and its variants, but to our knowledge, we are the first to use Rényi divergence for fair representation learning. These include an IB problem under a Rényi entropy complexity constraint [94], bounding the generalization error in learning algorithms [22], Rényi divergence variational inference [60], Rényi differential privacy [69] and the analysis and development of deep generative adversarial networks [8, 85, 71, 58]. In addition, Baharlouei *et al.* [4] developed a fair representation method but one based on Rényi correlation rather than divergence.

1.2.3 Connections to Other Work

Fairness is related to domain adaptation (DA) [6, 25, 62, 101] which consists of training a neural network on a source dataset to obtain good accuracy on a target dataset that is different from the source. This is especially true for the case of a severe data imbalance that we consider in this work where training data is completely missing for a protected subgroup. In this case, achieving fairness is similar to the DA problem of improving performance on a complete target dataset that includes all groups after training on an incomplete source dataset.

Our work is also related to group distributionally robust optimization (GDRO) methods [84, 87] that address the problem of performance disparity among different subgroups by minimizing the worst-case loss among different subgroups. Reducing these accuracy differences among subgroups is something our method also addresses, but while this is the sole objective for GDRO methods, we consider this problem in relation to multiple other fairness criteria.

There are some other work that investigated finding a balance between accuracy and fairness such as Zhang *et al.* [102]. However, they achieved this objective by finding early stopping criteria rather than through a fair representation preprocessing method like we use here; also, unlike our method, finding a balance between multiple fairness constraints is not investigated.

Finally, there are hybrid methods, such as the one proposed by Paul *et al.* in [72], that use a combination of the previously discussed techniques of interventions on data, model, or output. However, such techniques are quite limited in scope compared to the more varied objectives considered here which involve jointly achieving utility, fairness, and compactness.

1.3 Outline

In the end of this chapter, we introduce preliminary background on information measure. In Chapter 2, we provide extensive background material on neural networks. In

Chapter 3 and Chapter 4, we discuss the Variational Autoencoder and the Information Bottleneck Method, two deep learning techniques that are related to our work. Chapter 5 examines the problem of fairness in machine learning and presents our fairness method, RFIB. In Chapter 6, we consider the problem of privacy in machine learning, how it relates to fairness, and present our method to jointly improve privacy and fairness, RFPIB. We finish with a conclusion in Chapter 7.

1.4 Information Measures

We describe information measures used in this work. For more details, see the texts [1] and [14]. In this thesis, we assume that $0 \log 0 = 0$ and $x \log \frac{x}{0} = \infty$ for all $x > 0$, which is justified by continuity, since $x \log x \to 0$ as $x \to 0$.

Definition 1 (Entropy)**.** *The entropy of a discrete random variable X with probability mass function P_X taking values in finite alphabet $\mathcal{X}$ is defined as*

$$H(X) := -\sum_{x \in \mathcal{X}} P_X(x) \log P_X(x). \tag{1.1}$$

Entropy is a measure of uncertainty or average amount of information contained by a random variable. It is nonnegative ($H(X) \geq 0$), with equality holding if and only if X is deterministic.

Definition 2 (Conditional entropy)**.** *Given a pair of random variables (X, Y) with a joint probability mass function $P_{X,Y}$ defined on $\mathcal{X} \times \mathcal{Y}$, the conditional entropy of X given Y is defined as*

$$H(X|Y) := -\sum_{(x,y) \in (\mathcal{X} \times \mathcal{Y})} P_{X,Y}(x, y) \log P_{X|Y}(x|y), \tag{1.2}$$

where $P_{X|Y}(\cdot|\cdot)$ is the conditional probability mass function of Y given X.

Definition 3 (Joint entropy)**.** *The joint entropy of two discrete random variables X and Y with a joint probability mass function $P_{X,Y}$ defined on $\mathcal{X} \times \mathcal{Y}$ is defined as*

$$H(X,Y) := -\sum_{(x,y)\in(\mathcal{X}\times\mathcal{Y})} P_{X,Y}(x,y)P_{X,Y}(x,y). \tag{1.3}$$

Definition 4 (Cross entropy)**.** *The cross entropy of two probability distributions P_X and Q_X defined on a common discrete alphabet $\mathcal{X}$ is defined as:*

$$H\left(P_X; Q_X\right) := -\sum_{x\in\mathcal{X}} P_X(x)\log Q_X(x). \tag{1.4}$$

Note that with a small abuse of notation, similar notation is used in the literature for joint entropy and cross entropy.

Cross entropy is the average number of bits necessary to encode symbols from P_X using Q_X. Some of its properties include:

- $H\left(P_X; P_X\right) = H\left(P_X\right)$.
- $H\left(P_X; Q_X\right) \geq H\left(P_X\right)$.
- $H\left(P_X; Q_X\right) \neq H\left(Q_X; P_X\right)$ for any $P_X \neq Q_X$ (non-symmetric).

Definition 5 (Kullback-Leibler (KL) divergence)**.** *The KL divergence between discrete distributions P_X and Q_X taking values on a common discrete alphabet $\mathcal{X}$ is defined as*

$$D_{KL}\left(P_X \| Q_X\right) := \sum_{x\in\mathcal{X}} P_X(x)\log\frac{P_X(x)}{Q_X(x)}. \tag{1.5}$$

KL divergence can be thought of as a measure of distance or dissimilarity between two probability distributions. It is the amount of information lost when a distribution P_X is approximated by Q_X. It is always nonnegative; $D_{KL}\left(P_X \| Q_X\right) \geq 0$ with equality if and only if the two distributions are equal, $P_X(x) = Q_X(x)$ for all $x \in \mathcal{X}$, and it is nonsymmetric.

KL divergence is related to cross entropy in that

$$D(P_X \| Q_X) = H(P_X; Q_X) - H(P_X). \tag{1.6}$$

While cross entropy is the average number of bits necessary to encode symbols from P_X using Q_X, KL divergence is the average number of *additional* bits needed to encode symbols from P_X using Q_X. In practice, it is often desired to minimize the KL divergence. In the machine learning community, the cross entropy $H(P_X; Q_X)$ term is often considered instead of the KL divergence, but the objective is still finding a distribution Q_X that minimizes the KL divergence; clearly, minimizing the cross entropy is equivalent to minimizing the KL divergence.

Definition 6 (Rényi divergence [79]). *For $\alpha > 0$, $\alpha \neq 1$, the Rényi divergence of order α between discrete distributions P_X and Q_X taking values on a common discrete alphabet (support) $\mathcal{X}$ is defined as* [2]

$$D_\alpha(P_X \| Q_X) := \frac{1}{\alpha - 1} \log \left(\sum_{x \in \mathcal{X}} P_X(x)^\alpha Q_X(x)^{1-\alpha} \right). \tag{1.7}$$

Rényi divergence is a generalization of KL divergence. Like KL divergence, it is

[2]If P and Q are probability density functions with common support $\mathcal{X}$, then $D_\alpha(P||Q) = \frac{1}{\alpha-1} \log \left(\int_{\mathcal{X}} P(x)^\alpha Q(x)^{1-\alpha}\, dx \right)$.

nonnegative; $D_\alpha\left(P_X \| Q_X\right) \geq 0$ with equality if and only if $P_X(x) = Q_X(x)$ for all $x \in \mathcal{X}$. It is non-decreasing in α.

Theorem 1. *For two distributions P_X and Q_X with common alphabet, we have that*

$$\lim_{\alpha \to 1} D_\alpha\left(P_X \| Q_X\right) = D_{KL}\left(P_X \| Q_X\right). \tag{1.8}$$

Definition 7 (Mutual information)**.** *The mutual information of two random variables X and Y with joint distribution $P_{X,Y}$ and marginal distributions P_X and P_Y is defined as:*

$$I(X;Y) := D_{KL}\left(P_{X,Y}(x,y) \| P_X(x)P_Y(y)\right). \tag{1.9}$$

Mutual information is the average amount of information that X contains about Y and vice-versa. Equivalently, it is the reduction in the uncertainty of Y due to the knowledge X and vice-versa, as it can be written as

$$\begin{aligned} I(X;Y) &= H(X) - H(X|Y) \\ &= H(Y) - H(Y|X). \end{aligned} \tag{1.10}$$

Mutual information is symmetric; $I(X;Y) = I(Y;X)$ and $I(X;Y) = 0$ if and only if X and Y are independent.

Definition 8 (Conditional mutual information)**.** *For three discrete random variables X, Y, and Z the conditional mutual information of X and Y given Z is defined as*

$$I(X;Y|Z) := \sum_{z \in \mathcal{Z}} P_Z(z) D_{KL}\left(P_{X,Y|Z}(x,y|z) \| P_{X|Z}(x|z)P_{Y|Z}(y|z)\right). \tag{1.11}$$

Definition 9 (Markov Chain). *Random variables X, Y, Z form a Markov chain denoted by $X \to Y \to Z$ if*

$$P_{Z|X,Y}(z|x,y) = P_{Z|Y}(z|y) \quad \forall x, y, z, \tag{1.12}$$

or equivalently,

- *X and Z are conditionally independent given Y.*
- *$P_{X,Y,Z}(x,y,z) = P_X(x)P_{Y|X}(y|x)P_{Z|Y}(z|y) = P_Z(z)P_{Y|Z}(y|z)P_{X|Y}(x|y)$, for all x, y, z.*

Note that $X \to Y \to Z$ implies $Z \to Y \to X$, thus the Markov chain can also be expressed as $X \leftrightarrow Y \leftrightarrow Z$.

Chapter 2

Neural Networks

Artificial neural networks are computing systems inspired by the way the human brain processes information. They consist of interconnected layers of nodes called artificial neurons, with the terms *node*, *artificial neuron*, and *neuron* used interchangeably. They consist of several interconnected layers of nodes, with the first layer called the *input layer*, the last layer called the *output layer*, and layers in between called *hidden layers.* Information, for example the pixel values of an image, is passed into the input layer then processed through the network until reaching the output layer.

A *deep neural network* or *DNN* is one that has multiple hidden layers whereas a *shallow neural network* has only one hidden layer. A *feedforward* neural network, also called a *multilayer perceptron* or *MLP*, is the simplest type where information flows in only one direction from the input layer to the output layer, as opposed to other types of neural networks called *recurrent neural networks* that contain loops back to previous layers; see Figure 2.1 for an illustration of a feedforward neural network. The structure of a neural network is called its *architecture* and the network itself is often referred to as a machine learning *model.* In theory, there is no limit to how many layers a neural network can contain; this is limited only by computing power.

It can be shown that a neural network can approximate any continuous function [59].

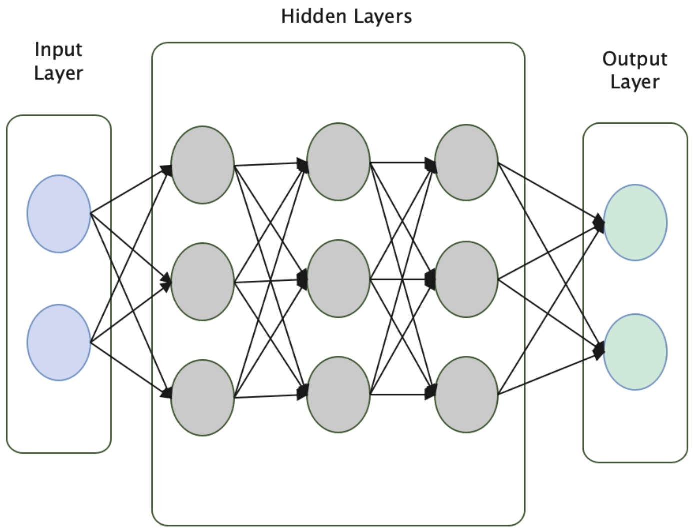


Figure 2.1: A feedforward deep neural network with three hidden layers.

An artificial neuron is a mathematical function that takes a weighted sum of its inputs, adds a bias term, and then passes the sum into a non-linear *activation function*:

Definition 10 (Neuron)**.** *A neuron is a function* $f : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ *such that for inputs* $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, *weights* $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$, *and bias* $b \in \mathbb{R}$,

$$f(\boldsymbol{x}, \boldsymbol{w}, b) = \sigma\left(\sum_{k=1}^{n} x_k w_k + b\right), \tag{2.1}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ *is an activation function.*

Each input x_i has an associated weight w_i which can be thought of as the strength of the connection to the neuron, with a higher weight meaning that the associated input has a higher importance and greater effect on the neuron.

Each neuron has a bias term b which can be thought of as the negative of a threshold that has to be exceeded for the neuron to activate, with an active neuron being one that has an output greater than 0. The higher the bias, the easier for the neuron to activate. Figure 2.2 shows a representation of a neuron.

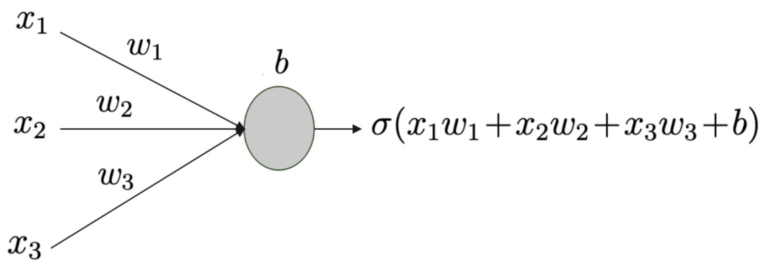


Figure 2.2: A neuron with three inputs.

The weighted sum of the inputs with added bias is then passed into the activation function, denoted $\sigma : \mathbb{R} \to \mathbb{R}$, which is a continuous function that is typically non-linear, monotonic, and continuously differentiable. The activation function determines whether the neuron is active and its main purpose is to introduce non-linearity to the network in addition to bounding the neuron's output within a threshold.

2.1 Activation Functions

There exist many activation functions, each with different advantages and disadvantages. We herein describe some of the most commonly used ones:

- **Sigmoid Function:** $\sigma : \mathbb{R} \to (0, 1)$ defined by:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.2}$$

This is a smoothed out version of a binary step function that gives an output of 1 for a large input and an output of 0 for a small output. It approaches 1 as $x \to \infty$ and approaches 0 as $x \to -\infty$. It has the disadvantage of *saturating* when the argument becomes a very positive or very negative value, becoming flat and insensitive to small changes of the argument.

- **Tanh Function:** $\sigma : \mathbb{R} \to (-1, 1)$ defined by:

$$\sigma(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{2.3}$$

This is a scaled version of the sigmoid function with a steeper slope and output centred at 0.

- **Rectified Linear Unit (ReLU):** $\sigma : \mathbb{R} \to [0, \infty)$ defined by:

$$\sigma(x) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases} \tag{2.4}$$

This is currently one of the most commonly used activation functions and has the advantage of being less computationally expensive than the sigmoid and tanh functions.

- **Leaky ReLU:** $\sigma : \mathbb{R} \to [0, \infty)$ defined by:

$$\sigma(x) = \begin{cases} x, & \text{if } x \geq 0, \\ -\epsilon x, & \text{if } x < 0, \end{cases} \tag{2.5}$$

where $\epsilon \in \mathbb{R}_{>0}$. This is a variation of the ReLU function designed to address the *dying ReLU* problem where too many negative inputs can cause a neuron to become permanently inactive. For a negative input x, the function has a small negative slope ϵ (typically 0.01, though this number is somewhat arbitrary) rather than 0, preventing the neuron from "dying."

- **Softmax:** The softmax function is a special activation function that is typically only used for the neurons in the final output layer, with the output of the softmax being the final output of the neural network. In addition to adding non-linearity like the other activation functions, the purpose of the softmax is to normalize the network's output to a probability distribution, ensuring the outputs of all the nodes in the final layer sum to 1.

  Let $\boldsymbol{z} \in \mathbb{R}^{n_l}$ denote the vector of all weighted inputs to a layer l of a neural network, with n_l denoting the number of nodes in layer l, and z_i the weighted input to the ith node of layer l, where

$$z_i = \sum_{k=1}^{n_{l-1}} x_k w_{ik} + b_i. \tag{2.6}$$

  Here n_{l-1} is the number of nodes in the previous layer $l-1$, $\boldsymbol{x} \in \mathbb{R}^{n_{l-1}}$ is the outputs of layer $l-1$ and inputs to current layer l, $\boldsymbol{w_i} \in \mathbb{R}^{n_{l-1}}$ is the weights between all the elements of $\boldsymbol{x}$ and node z_i, and $b_i \in \mathbb{R}$ is the bias of node z_i.

  Then the softmax function $\sigma : \mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$ is defined element-wise by

$$\sigma\left(\boldsymbol{z}\right)_i = \frac{e^{z_i}}{\sum_{j=1}^{n_l} e^{z_j}} \qquad \text{for } i \in \left\{1, \ldots, n_l\right\}. \tag{2.7}$$

The weighted input of every neuron in the layer is summed in the denominator. This results in all the outputs of the function summing to 1:

$$\sum_{i=1}^{n_L} \sigma(\boldsymbol{z})_i = \sum_{i=1}^{n_L} \frac{e^{z_i}}{\sum_{j=1}^{n_l} e^{z_j}} = \frac{\sum_{j=1}^{n_L} e^{z_i}}{\sum_{j=1}^{n_l} e^{z_j}} = 1. \tag{2.8}$$

Also, $0 < \sigma(\boldsymbol{z})_i \leq 1$, for all $i \in \{1, \dots, n_l\}$, as the denominator is greater than or equal to the numerator. This results in $\sigma(\boldsymbol{z})$ forming what can be thought of as a probability distribution.

2.2 Training a Neural Network

For supervised learning, the goal of using a neural network is to produce a desired output from an input. This output is called the network's *prediction*. For example, an input can be an image of a digit and the desired output can be the digit represented in the image. The network has to be trained to predict this desired output, with the terms *training* or *learning* meaning adjusting all the weights and biases of the network to minimize the difference between the network's prediction (actual output) and the desired output.

Definition 11. *An example is a pair* $(\boldsymbol{x}, \boldsymbol{y})$ *where* $\boldsymbol{x} \in \mathbb{R}^n$ *is the input to the network and* $\boldsymbol{y} \in \mathbb{R}^m$ *is the desired output.*

An example is also called a *data point* or *sample*. The input $\boldsymbol{x}$ is also called a collection of *features*, where each entry x_i is a feature, and the desired output $\boldsymbol{y}$ is also called a *label*, *ground truth label*, or simply the *ground truth*. This label is typically encoded as a *one-hot encoding*.

Definition 12 (One-hot encoding)**.** *A one-hot encoding* $\boldsymbol{y} \in \mathbb{R}^m$ *is a vector where one component is equal to 1, its index representing a value, and every other component is equal to 0.*

For example, for the basic task of classifying digits ranging from 0-9, the label 5 is encoded as $\boldsymbol{y} = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)^T$, with $y_5 = 1$, using indices starting at 0. For this task, since there are 10 different digits to be classified, the network's output layer will have 10 nodes, with each node representing 1 of the 10 classes.

Definition 13 (Training set)**.** *A training set* $\mathcal{D}\left\{\left(\boldsymbol{x}^{\{i\}}, \boldsymbol{y}^{\{i\}}\right)\right\}_{i=1}^{N}$ *is a set of training examples, where* N *is the number of examples.*

Examples are called *training examples* when they are used for training as part of a training set. The training process consists of iterating over a training set and adjusting the network's weights and biases each time. The goal is for the neural network to approximate an unknown function f that maps the input $\boldsymbol{x}$ to the desired output $\boldsymbol{y}$. Letting $\boldsymbol{\theta} \in \mathbb{R}^s$ denote all the weights and biases in the neural network (also called the network's *parameters)*, learning consists of finding the optimal $\boldsymbol{\theta}$ that produces the best approximation $\hat{f}$ of f such that $\hat{\boldsymbol{y}} = \hat{f}(\boldsymbol{x}; \boldsymbol{\theta})$, where $\hat{\boldsymbol{y}}$ is the neural network's prediction. Here $\hat{f}(\boldsymbol{x}; \boldsymbol{\theta})$ is produced by the neural network, and as expected is a function of both the input and the weights and biases.

2.2.1 Cost Functions

To determine how close the approximation is, a cost function is used (also called a *loss* function or *objective* function). A cost function is a continuous function $C : \mathbb{R}^s \to \mathbb{R}$ which measures the average difference between the network's output $\hat{\boldsymbol{y}} = \hat{f}(\boldsymbol{x}; \boldsymbol{\theta})$ and the desired output $\boldsymbol{y}$ over all samples in the training set. Cost functions should satisfy two properties:

1. $C(\boldsymbol{\theta}) \geq 0$, for all $\boldsymbol{\theta}$.
2. $C(\boldsymbol{\theta}) \to 0$ as $\hat{\boldsymbol{y}} \to \boldsymbol{y}$, and $C(\boldsymbol{\theta}) = 0$ when $\hat{\boldsymbol{y}} = \boldsymbol{y}$.

A typical choice for the cost function is the average loss given by

$$C(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\hat{p}_{\text{data}}} L(\hat{f}(\boldsymbol{x};\boldsymbol{\theta}), \boldsymbol{y}), \tag{2.9}$$

where $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is the loss of an individual example and the expectation is over all samples in the empirical distribution of the dataset, $\hat{p}_{\text{data}}$. Note that the cost function is sometimes written as a function of the output and desired output, $C(\hat{\boldsymbol{y}}, \boldsymbol{y})$, or the output, desired output, and parameters, $C(\hat{\boldsymbol{y}}, \boldsymbol{y}, \boldsymbol{\theta})$, but here we regard the training points as fixed and simply write the cost function as a function of the weights and biases $\boldsymbol{\theta}$.

Two functions commonly used to find the loss of each example are the *quadratic* (or *mean squared error*) loss and the *cross-entropy* (or *negative log-likelihood*) loss:

- **Quadratic Loss:**
$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = ||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2^2. \tag{2.10}$$

- **Cross-entropy Loss:**
$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_{j=1}^{m} y_j \log \hat{y}_j \tag{2.11}$$

  where the summation is over each element j of the vectors $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$ of length m.

Using these per-example loss functions as part of a cost function, we get:

- **Quadratic Cost:**

$$C(\boldsymbol{\theta}) = \frac{1}{2N}\sum_{i=1}^{N} ||\hat{f}(\boldsymbol{x}^{\{i\}};\boldsymbol{\theta}), \boldsymbol{y}^{\{i\}}||_2^2 \tag{2.12}$$

  where $(\boldsymbol{x}^{\{i\}}, \boldsymbol{y}^{\{i\}})$ is one training example out of N total. The factor of $\frac{1}{2}$ is not necessary but is often included for convenience as it simplifies matters when differentiating during backpropagation, which we discuss later.

- **Cross entropy Cost:**

$$C(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} y_j^{\{i\}} \log \hat{f}(\boldsymbol{x}^{\{i\}};\boldsymbol{\theta})_j. \tag{2.13}$$

  The cross entropy cost is related to cross entropy; see Definition 4. The cross entropy cost function is typically used in conjunction with the softmax activation function, meaning that all the elements of the output vector $\hat{\boldsymbol{y}} = \hat{f}(\boldsymbol{x};\boldsymbol{\theta})$ sum to 1 and $0 \le \hat{y}_i \le 1$ for all $i \in \{1, \dots, m\}$, and thus $\hat{\boldsymbol{y}}$ can be thought of as a probability distribution. Similarly, as the vector $\boldsymbol{y}$ is typically a one-hot encoding, one of its components is a 1 and the rest are 0, and so it also satisfies the previous two properties of a probability distribution, although it is a deterministic one. Thus, the vectors $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ can be thought of as representing the distributions P_X and Q_X, respectively, in the original definition of cross entropy. Training the neural network can thus be thought of as finding the $\boldsymbol{\theta}$ that results in the $\hat{\boldsymbol{y}}$ that minimizes cross entropy.

Example 1. *For a task classifying digits from 0-9 where we train using a single* *training example consisting of an image of the digit 7 and the label 7, assuming the network predicts the digit is a 7 with a probability of 0.30872, the cross entropy loss is*

$$C(\boldsymbol{\theta}) = -\sum_{j=0}^{9} y_j \log \hat{y}_j = -\log \hat{y}_7 = -\log(0.30872) = 1.1753. \tag{2.14}$$

The label 7 is one-hot encoded as $\boldsymbol{y} = (0, 0, 0, 0, 0, 0, 0, 1, 0, 0)^T$ and thus all the terms in the summation where $j \neq 7$ are 0.

Example 2. *Assume the same task as the previous example except this time there is a second image, an image of the digit 2 that is classified as a 2 with a probability of 0.31604; see Figure 2.3 for an illustration of this example. Then the cross entropy loss is*

$$C(\boldsymbol{\theta}) = -\frac{1}{2}\sum_{i=1}^{2}\sum_{j=0}^{9} y_j^{\{i\}} \log \hat{y}_j^{\{i\}} = -\frac{1}{2}\left(\log(0.23086) + \log(0.31604)\right) = 1.3089. \tag{2.15}$$

2.3 Gradient Descent

As the cost function is typically very high dimensional and non-convex, finding the global minimum is typically non-feasible. Instead, an iterative optimization algorithm or *optimizer* is used, typically *gradient descent* or a related variation.

For the most basic version of gradient descent, sometimes called *vanilla gradient descent*, the network's parameters $\boldsymbol{\theta}$ are iteratively adjusted in the negative direction of the cost function's gradient. Letting $t \in \mathbb{Z}_{>0}$ be a timestep, $\nabla_{\boldsymbol{\theta}_t} C(\boldsymbol{\theta}_t)$ be the gradient of the cost function at timestep t with respect to the parameters $\boldsymbol{\theta}$, and $\eta \in \mathbb{R}_{>0}$ be the *learning rate* (also known as the *step size*), an iteration of the

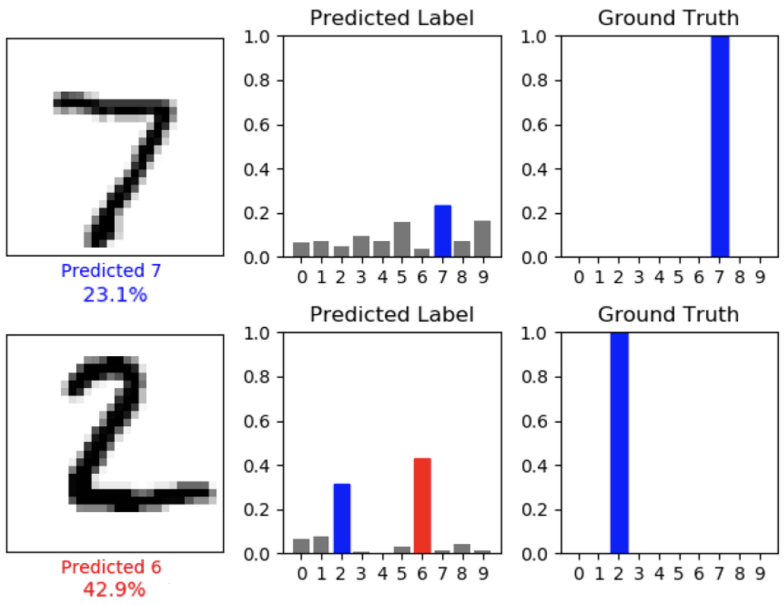


Figure 2.3: Images of the digits of 7 and 2 from the MNIST database [16] (left), predictions $\hat{\boldsymbol{y}}$ from a neural network (centre), and desired output $\boldsymbol{y}$ (right). The network has 10 output nodes, each representing a digit from 0 to 9, and the digit corresponding to the node with the highest value is the network's prediction; ideally, that node should ouput a 1 and all others output a 0. For the digit of a 7 (top row), the network is correct in its prediction, as node corresponding to 7 has the highest value of 0.231, whereas for the digit of a 2 (bottom row) the network is incorrect as the node corresponding to 6 has a value of 0.429, higher than the node corresponding to the 2.

gradient descent algorithm is given by:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}_t} C(\boldsymbol{\theta}_t). \tag{2.16}$$

The process continues until a stopping condition representing a desired level of accuracy is reached. Note that for a η small enough, gradient descent is guaranteed to decrease the cost [70]. If we adjust $\boldsymbol{\theta}$ by a small amount, denoted $\Delta\boldsymbol{\theta}$, then the change in $C(\boldsymbol{\theta})$, denoted $\Delta C(\boldsymbol{\theta})$, can be approximated by

$$\Delta C(\boldsymbol{\theta}) \approx \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_1}\Delta\theta_1 + \cdots + \frac{\partial C(\boldsymbol{\theta})}{\partial \theta_m}\Delta\theta_m = \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) \cdot \Delta\boldsymbol{\theta}. \tag{2.17}$$

Letting $\Delta\boldsymbol{\theta} = -\eta\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$, for an η small enough that the approximation is valid,

$$\Delta C(\boldsymbol{\theta}) \approx -\eta\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) \cdot \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) = -\eta\|\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta})\|^2 \leq 0. \tag{2.18}$$

Thus, as long as η is small enough, the algorithm will converge to a minimum or saddle point where the gradient is 0. It is however not guaranteed to reach a global minimum and the possibility of reaching a saddle point is a potential problem. While this could be avoided by computing the Hessian $\nabla^2_{\boldsymbol{\theta}} C(\boldsymbol{\theta})$, this is usually not done in practice due to the increased computational requirements. Although it is not guaranteed to converge to a useful minimum, the gradient descent algorithm typically gives good results and is often used in practice.

2.3.1 Stochastic Gradient Descent

Gradient descent has the disadvantage of requiring gradients to be computed for all N training examples:

$$\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} L\left(\hat{f}\left(\boldsymbol{x}^{\{i\}};\boldsymbol{\theta}\right), \boldsymbol{y}^{\{i\}}\right). \tag{2.19}$$

As N is typically a large number, this is computationally expensive. In practice, *stochastic gradient descent (SGD)* is typically used instead. SGD, also sometimes called *mini-batch gradient descent*, is a stochastic approximation of gradient descent that uses an estimate of the gradient computed from a randomly selected subset of the dataset.

For SGD, first the dataset is randomly shuffled and then subdivided into subsets called *mini-batches.* The gradient is then computed over all training examples in the selected minibatch, and used as an estimate of the true gradient over all training examples. The same iterative algorithm as for gradient descent is used as in (2.16), except the gradient of the cost is estimated as

$$\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}) \approx \frac{1}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\theta}} L\left(\hat{f}\left(\boldsymbol{x}^{\{i\}}; \boldsymbol{\theta}\right), \boldsymbol{y}^{\{i\}}\right), \tag{2.20}$$

where $M \leq N$ is a positive integer, resulting in one iteration of SGD being given by:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\eta}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\theta_t}} L\left(\hat{f}\left(\boldsymbol{x}^{\{i\}}; \boldsymbol{\theta_t}\right), \boldsymbol{y}^{\{i\}}\right). \tag{2.21}$$

Typically $N = kM$ for some $k \in \mathbb{Z}_{>0}$, and the training set is divided into k mini-batches. Running the SGD algorithm for k iterations, passing through all the examples in the training set one time, is called completing one *epoch* of training. After an epoch is completed, the training set is randomly shuffled again, split into different mini-batches, and the process continues.

Note that sometimes *stochastic gradient descent* is used to refer to only a mini-batch size of $M = 1$, with *mini-batch stochastic gradient descent* referring to a mini-batch size where $1 < M < N$. Here we take stochastic gradient descent to mean any mini-batch size where $M < N$.

There currently does not exist a universally accepted way to calculate an optimal mini-batch size. In practice, mini-batch sizes are usually powers of 2, with typical sizes ranging from 32 to 256. This is done to decrease computational time as some hardware devices, especially graphics processing units (GPUs), offer better runtime on batch sizes that are powers of 2. While it would be expected that a larger batch size is better as it provides a more accurate estimate of the gradient, this is not necessarily the case and some, such as [65], argue that smaller mini-batch sizes from 2 to 32 yield best performance. One possible reason is that smaller batches can offer a regularization effect due to the noise they add to the training process, helping reduce generalization error [96], something we discuss later.

2.3.2 Momentum

There exist many other optimization algorithms that are variations of stochastic gradient descent. One is called *momentum* (or gradient descent with momentum). Momentum is a technique designed to accelerate learning, leading to faster convergence. This algorithm accumulates an exponentially decaying moving average of past gradients.

We introduce a new update vector, $\boldsymbol{m} \in \mathbb{R}^s$ which keeps track of past gradients, as well as a hyperparameter $\beta \in [0, 1)$ which controls how quickly the past gradients decay exponentially. An iteration of momentum is then given by:

$$\boldsymbol{m}_{t+1} \leftarrow \beta \boldsymbol{m}_t + (1 - \beta) \nabla_{\boldsymbol{\theta}_t} C(\boldsymbol{\theta}_t), \tag{2.22}$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \boldsymbol{m}_{t+1}. \tag{2.23}$$

This can be thought of as giving stochastic gradient descent memory. An analogy used is pushing a ball down a hill that accumulates momentum as it rolls downhill, increasing its velocity on the way [82]. The gradient can be though of as something that changes the ball's velocity rather than its position as in SGD.

If $\beta = 0$, this results in regular stochastic gradient descent. But for higher values of β, higher weight is placed on the previous gradients to determine the current direction. In practice, large values of β are typically used, such as $\beta = 0.9$ or $\beta = 0.99$, which results in what's called *acceleration*. Momentum results in reducing oscillations and typically results in faster convergence, giving up to a quadratic boost in speed for many functions compared to SGD [31].

2.3.3 AdaGrad

The learning rate η has a very large effect on the learning process. While the previously discussed optimizers update all the parameters $\boldsymbol{\theta}$ at the same rate, there exist adaptive algorithms that use a different learning rate for each parameter θ_i at each timestep t. AdaGrad [19], short for adaptive gradient, is one such algorithm.

To simplify notation, we let $g_{t,i}$ be the gradient of the cost function with respect to the parameter θ_i at timestep t,

$$g_{t,i} = \frac{\partial C(\boldsymbol{\theta}_t)}{\partial \theta_{t,i}}. \tag{2.24}$$

Then one iteration of AdaGrad is given by:

$$\theta_{t+1,i} \leftarrow \theta_{t,i} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} \cdot g_{t,i}, \qquad i = 1, \ldots, d. \tag{2.25}$$

This results in smaller updates being performed for parameters with a history of large gradients, and larger updates for parameters with small gradients. This is especially helpful for dealing with sparse data, as more weight can be shifted onto parameters associated with inputs that occur infrequently. The learning rate decreases faster for parameters that change frequently and slows down for parameters that only change occasionally [99]. AdaGrad also has the benefit that there is no need to manually tune η with most implementations using a default value of 0.01 with no changes.

However, AdaGrad has a weakness in the accumulation of squared gradients in the denominator. As every added term is positive, the sum keeps growing at every iteration t, eventually causing the learning rate to become infinitesimally small and the algorithm no longer making any updates to $\boldsymbol{\theta}$.

To combat this weakness, RMSProp was developed, an unpublished algorithm proposed by Geoffery Hinton in Lecture 6e of a Coursera class [45]. It replaces AdaGrad's summation of squared gradients in the denominator with an exponentially decaying average of squared gradients.

We let $\boldsymbol{v} \in \mathbb{R}^s$ be this exponentially decaying average, and $\beta \in [0, 1)$ control the rate of decay. To simplify notation, we drop the i subscript and vectorize the equation, with all operations performed element-wise. Then an iteration of RMSProp is given by:

$$\boldsymbol{v}_{t+1} \leftarrow \beta \boldsymbol{v}_t + (1-\beta)\boldsymbol{g}_{t+1}^2, \tag{2.26}$$

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\boldsymbol{v}_{t+1}}}\boldsymbol{g}_{t+1}. \tag{2.27}$$

2.3.4 Adam Optimizer

The Adam optimizer [52] is currently one of the most commonly used optimizers. It is a continuation of the previously discussed techniques, and can be thought of as a combination of RMSProp with momentum, with bias correction added.

It involves two exponential moving averages, one of the gradient, similar to SGD with momentum, and one of the gradient squared, similar to RMSProp. As before, we use $\boldsymbol{m}$ for the running average of the gradient with the rate of decay controlled by $\beta_1 \in [0, 1)$, and $\boldsymbol{v}$ for the running average of the squared gradient, with rate controlled by hyperparameter $\beta_2 \in [0, 1)$.

First, we initialize $\boldsymbol{m}$ and $\boldsymbol{v}$ to the all-zero vector, $\boldsymbol{m} = \mathbf{0}$ and $\boldsymbol{v} = \mathbf{0}$. Then the first stage of the algorithm is to compute the running averages of the gradient and squared gradient:

$$\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t, \tag{2.28}$$

$$\boldsymbol{v}_t \leftarrow \beta_2 \cdot \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2. \tag{2.29}$$

These two running averages are estimates of the first and second moment of the gradient. However, since both $\boldsymbol{m}$ and $\boldsymbol{v}$ are initialized as $\mathbf{0}$ the estimate will be biased toward 0. The bias is especially high during the initial timesteps and with small decay rates where the two β values are close to 1.

The second stage of the algorithm is to use a bias-correction technique to counteract this initialization bias. We use $\hat{\boldsymbol{m}}$ and $\hat{\boldsymbol{v}}$ to denote the bias-corrected exponential moving averages of the gradient and squared gradient. This is done by:

$$\hat{\boldsymbol{m}}_t \leftarrow \frac{\boldsymbol{m}_t}{(1 - \beta_1^t)}, \tag{2.30}$$

$$\hat{\boldsymbol{v}}_t \leftarrow \frac{\boldsymbol{v}_t}{(1 - \beta_2^t)}. \tag{2.31}$$

A proof of how this bias correction technique works is given later.

Finally, we use these two bias corrected quantities to give one iteration of the Adam optimizer:[1]

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \boldsymbol{\epsilon}}. \tag{2.32}$$

The parameters are updated by the learning rate η multiplied by the bias corrected exponential moving average of the gradient divided by the square root of the bias corrected exponential moving average of the squared gradient. We also introduce a new term of $\boldsymbol{\epsilon} \in \mathbb{R}^s_{>0}$ in the denominator, which is chosen to be close to the zero vector and is included simply to prevent division by 0. Default values of $\epsilon_i = 10^{-8}$ for all components of $\boldsymbol{\epsilon}$ are suggested. Combining these steps, we get the full algorithm for the Adam optimizer, given below as Algorithm 1. The authors of the original work [52] recommend default values of $\eta = 0.001, \beta_1 = 0.9$, and $\beta_2 = 0.999$, and these values are rarely changed in practice.

Next we show that $\hat{\boldsymbol{m}}_t$ and $\hat{\boldsymbol{v}}_t$ are unbiased estimators of $\boldsymbol{g}_t$ and $\boldsymbol{g}_t^2$, respectively, assuming $\mathbb{E}[\boldsymbol{g}_t]$ and $\mathbb{E}[\boldsymbol{g}_t^2]$ are constant for all timesteps $t \in \{1, \ldots, T\}$. If this condition is not met, the estimators are biased by a small error term $\boldsymbol{\zeta} \in \mathbb{R}^s$.

Theorem 2. *$\mathbb{E}[\hat{\boldsymbol{m}}_t] = \mathbb{E}[\boldsymbol{g}_t] + \boldsymbol{\zeta_1}$ and $\mathbb{E}[\hat{\boldsymbol{v}}_t] = \mathbb{E}[\boldsymbol{g}_t^2] + \boldsymbol{\zeta_2}$, where $\boldsymbol{\zeta_1} = \mathbf{0}$ and $\boldsymbol{\zeta_2} = \mathbf{0}$ if $\mathbb{E}[\boldsymbol{g}_t] = \boldsymbol{a}$ and $\mathbb{E}[\boldsymbol{g}_t^2] = \boldsymbol{b}$ for all timesteps $t \in \{1, \ldots, T\}$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are constant vectors.*

Proof. First, note that one can inductively conclude that

$$\boldsymbol{m}_t = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} \boldsymbol{g}_i. \tag{2.33}$$

[1]Note that all operations in the algorithm including the division and square root are performed element-wise.

Algorithm 1 Adam optimizer

```
Require: $\eta$                                   ▷ Learning rate (stepsize)
Require: $\beta_1, \beta_2 \in [0, 1)$            ▷ Exponential decay rates for moment estimates
Require: $C(\boldsymbol{\theta})$                 ▷ Objective function with parameters $\boldsymbol{\theta}$
Require: $\boldsymbol{\theta}_0$                  ▷ Initial parameter vector
 1: $\boldsymbol{m}_0 \leftarrow 0$               ▷ First moment initialized to 0
 2: $\boldsymbol{v}_0 \leftarrow 0$               ▷ Second moment initialized to 0
 3: $t \leftarrow 0$                              ▷ Timestep initialized to 0
 4: while $t \in \{0, \dots, T\}$ do
 5:     $t \leftarrow t + 1$                      ▷ Increment timestep
 6:     $\boldsymbol{g}_t \leftarrow \nabla_{\boldsymbol{\theta}_{t-1}} C_t(\boldsymbol{\theta}_{t-1})$   ▷ Calculate gradient of objective at timestep $t$
 7:     $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$   ▷ Update exponential moving average of gradient
 8:     $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$   ▷ Update exponential moving average of squared
    gradient
 9:     $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$   ▷ Correct for bias
10:     $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$   ▷ Correct for bias
11:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \cdot \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \boldsymbol{\epsilon})$   ▷ Update parameters
12: end while
13: return $\boldsymbol{\theta}_t$                ▷ Return parameters that minimize objective $C(\boldsymbol{\theta})$
```

Using this formula,

$$\mathbb{E}\left[\hat{\boldsymbol{m}}_t\right] = \mathbb{E}\left[\frac{1-\beta_1}{1-\beta_1^t}\sum_{i=1}^{t}\beta_1^{t-i}\boldsymbol{g}_i\right]. \tag{2.34}$$

Next we make an approximation and assume that $\boldsymbol{g}_i = \boldsymbol{g}_t$ for all $t \in \{1, \dots, T\}$. This approximation introduces the error term $\boldsymbol{\zeta_1}$ which is 0 when the true moment $\mathbb{E}[\boldsymbol{g}_i]$ is stationary. As β_1 is typically large, with a default value of $\beta_1 = 0.9$, the gradient terms for values of $i < t$ will be small, and thus the approximation error $\boldsymbol{\zeta_1}$ will also be small. Making this approximation allows us to move the gradient outside the summation as it no longer depends on i:

$$\mathbb{E}\left[\hat{\boldsymbol{m}}_t\right] = \mathbb{E}\left[\boldsymbol{g}_t\right] \cdot \frac{1-\beta_1}{1-\beta_1^t}\sum_{i=1}^{t}\beta_1^{t-i} + \boldsymbol{\zeta_1}. \tag{2.35}$$

Finally, we use the formula for a sum of a finite geometric series to rewrite the summation:

$$\begin{aligned}\mathbb{E}\left[\hat{\boldsymbol{m}}_t\right] &= \mathbb{E}\left[\boldsymbol{g}_t\right] \cdot \frac{1-\beta_1}{1-\beta_1^t} \frac{\beta_1^t \beta_1^{-1}(1-\beta_1^{-t})}{1-\beta_1^{-1}} + \boldsymbol{\zeta_1} \\ &= \mathbb{E}\left[\boldsymbol{g}_t\right] \cdot \frac{1-\beta_1}{1-\beta_1^t} \frac{\beta_1^t - 1}{\beta_1 - 1} + \boldsymbol{\zeta_1} \\ &= \mathbb{E}\left[\boldsymbol{g}_t\right] + \boldsymbol{\zeta_1}.\end{aligned} \tag{2.36}$$

To show that $\mathbb{E}[\hat{\boldsymbol{v}}_t] = \mathbb{E}[\boldsymbol{g}_t^2] + \boldsymbol{\zeta_2}$, an expression for $\boldsymbol{v}_t$ can be found in the same way as for $\boldsymbol{m}_t$:

$$\boldsymbol{v}_t = (1-\beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \boldsymbol{g}_i^2, \tag{2.37}$$

and the proof is entirely analogous.

$\square$

2.4 Backpropagation

To compute the gradient $\nabla C(\boldsymbol{\theta})$, the *backpropagation* algorithm is used. The purpose of backpropagation is to efficiently compute all the partial derivatives of the cost function with respect to every weight and bias, $\frac{\partial C}{\partial \theta_i}$ for every $i \in \{1, \ldots, s\}$, showing how quickly the cost changes based on changing a weight or bias. To simplify notation, we write the cost function as C. Before we give the equations of backpropagation, we introduce some new notation.

Suppose a neural network has L layers indexed by l, with $l = 1$ the input layer and $l = L$ the output layer. Let $a_j^{(l)}$ denote the activation of the jth neuron in the lth layer (where activation is the neuron's output), $b_j^{(l)}$ denote the bias of the jth neuron in the lth layer, and $w_{jk}^{(l)}$ denote the weight between the jth neuron in layer l and a kth neuron in previous layer $l-1$. Let n_l be the number of neurons in layer l and σ

be an activation function as previously defined. Then

$$a_j^{(l)} = \sigma\left(\sum_{k=1}^{n_{l-1}} w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}\right), \quad l \in \{2, \dots, L\} \tag{2.38}$$

where the sum is over all n_{l-1} neurons k in layer $l-1$.

This expression can also be written in matrix form. Let $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ be a matrix containing all the weights for layer l, with each row j containing the weights of the jth neuron in layer l with all n_{l-1} neurons in layer $l-1$. Let $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ be a vector containing the biases of all n_l neurons in layer l and $\boldsymbol{a}^{(l)} \in \mathbb{R}^{n_l}$ be a vector containing the activations of all n_l neurons in layer l. Next, we vectorize the activation function σ as $\sigma(\boldsymbol{v})$: $\mathbb{R}^{n_l} \to \mathbb{R}^{n_l}$, where σ is applied to each component of $\boldsymbol{v}$, i.e. $\sigma(\boldsymbol{v})_j = \sigma(v_j)$. Then

$$\boldsymbol{a}^{(l)} = \sigma\left((\boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}\right), \quad l \in \{2, \dots, L\}. \tag{2.39}$$

We use $\boldsymbol{z}^{(l)} \in \mathbb{R}^{n_l}$ to denote the vector of all weighted inputs to nodes in layer l,

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}, \quad l \in \{2, \dots, L\} \tag{2.40}$$

Next we define a new intermediate quantity.

Definition 14 (Error [44]). *The error of the jth neuron in layer l is given by*

$$\delta_j^{(l)} \coloneqq \frac{\partial C}{\partial z_j^{(l)}}, \textit{ for } 1 \le j \le n_l,\ 2 \le l \le L. \tag{2.41}$$

The error quantifies how sensitive the cost function is to its corresponding neuron. Note that the usage of the term error is somewhat misleading since it is unclear how much each individual neuron is responsible for misclassifications that occur in the network's final output layer. This usage has arisen since the cost function can only reach a minimum when all partial derivatives are 0, so it is desired to achieve an error of 0 for all neurons [44]. We express the errors as a vector, where $\boldsymbol{\delta}^{(l)} \in \mathbb{R}^{n_l}$ is the vector containing all the errors in layer l.

We also define the Hadamard product:

Definition 15 (Hadamard product)**.** *Given two vectors $\boldsymbol{x}$, $\boldsymbol{y} \in \mathbb{R}^n$, then $\boldsymbol{x} \odot \boldsymbol{y} \in \mathbb{R}^n$ is defined element-wise by:*

$$(x \odot y)_i = x_i y_i. \tag{2.42}$$

Using this notation, we introduce four equations of backpropagation which are consequences of the chain rule.

Theorem 3. *The backpropagation equations of a neural network are*

$$\begin{aligned}
\boldsymbol{\delta}^{(L)} &= \nabla C_{\boldsymbol{a}^{(L)}} \odot \sigma'(\boldsymbol{z}^{(L)}), && && (2.43)\\
\boldsymbol{\delta}^{(l)} &= (\boldsymbol{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)} \odot \sigma'(\boldsymbol{z}^{(l)}), && l \in \{2, \ldots, L-1\}, && (2.44)\\
\frac{\partial C}{\partial b_j^{(l)}} &= \delta_j^{(l)}, && l \in \{2, \ldots, L\}, && (2.45)\\
\frac{\partial C}{\partial w_{jk}^{(l)}} &= \delta_j^{(l)} a_k^{(l-1)}, && l \in \{2, \ldots, L\}, && (2.46)
\end{aligned}$$

where $\nabla C_{\boldsymbol{a}^{(L)}}$ is the gradient vector of C with respect to the activations of layer L.

Proof. We begin by proving (2.43). Recall that the cost function compares the difference between the network's desired output $\boldsymbol{y}$ and true output $\hat{\boldsymbol{y}}$. The true output is the activation of the neurons in the final layer,

$$\hat{\boldsymbol{y}} = \boldsymbol{a}^{(L)} = \sigma(\boldsymbol{z}^{(L)}). \tag{2.47}$$

Then

$$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = \sigma'\left(z_j^{(L)}\right), \tag{2.48}$$

and since C depends on $\boldsymbol{z}$ through $\boldsymbol{a}$, by the chain rule,

$$\delta_j^{(L)} = \frac{\partial C}{\partial z_j^{(L)}} = \sum_{k=1}^{n_L} \frac{\partial C}{\partial a_k^{(L)}} \frac{\partial a_k^{(L)}}{\partial z_j^{(L)}} = \frac{\partial C}{\partial a_j^{(L)}} \sigma'(z_j^{(L)}), \tag{2.49}$$

where the final equality follows since $\frac{\partial a_k^{(L)}}{\partial z_j^{(L)}} = 0$ when $k \neq j$. This is the component-wise form of (2.43).

To prove (2.44), we start by rewriting $\delta_j^{(l)}$ in terms of $\delta_k^{(l+1)}$ using the chain rule:

$$\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}. \tag{2.50}$$

Recall that

$$z_j^{(l)} = \sum_{k=1}^{n_{l-1}} w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)} = \sum_{k=1}^{n_{l-1}} w_{jk}^{(l)} \sigma(z_k^{(l-1)}) + b_j^{(l)}, \tag{2.51}$$

$$z_k^{(l+1)} = \sum_{j=1}^{n_l} w_{kj}^{(l+1)} \sigma(z_j^{(l)}) + b_k^{(l+1)}, \tag{2.52}$$

and thus

$$\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = w_{kj}^{(l+1)} \sigma'(z_j^{(l)}). \tag{2.53}$$

Substituting (2.53) back into (2.50), we get

$$\delta_j^{(l)} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} \sigma'(z_j^{(l)}), \tag{2.54}$$

the component-wise form of (2.44). To prove (2.45), by (2.52),

$$\frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = 1, \tag{2.55}$$

and thus by the chain rule,

$$\frac{\partial C}{\partial b_j^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}(1) = \frac{\partial C}{\partial z_j^{(l)}} = \delta_j^{(l)}. \tag{2.56}$$

Similarly, to prove (2.46), by (2.52),

$$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = a_k^{(l-1)}, \tag{2.57}$$

and by the chain rule,

$$\frac{\partial C}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = \frac{\partial C}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} a_k^{(l-1)} = \delta_j^{(l)} a_k^{(l-1)}. \tag{2.58}$$

□

Using these four fundamental equations, all partial derivatives of C are computed. First, for each training point, a *forward pass* is taken through the network, computing the activations and weighted inputs starting from the first layer and progressing toward the last layer, in the order $\boldsymbol{a}^{(1)}, \boldsymbol{z}^{(2)}, \boldsymbol{a}^{(2)}, \boldsymbol{z}^{(3)}, \boldsymbol{a}^{(3)}, \ldots, \boldsymbol{a}^{(L)}$. Next, a *backward pass* is taken, computing partial derivatives and finding the error terms starting from the last layer and progressing toward the first layer, in the order $\boldsymbol{\delta}^{(L)}, \boldsymbol{\delta}^{(L-1)}, \ldots, \boldsymbol{\delta}^{(2)}$. The first error $\boldsymbol{\delta}^{(L)}$ is given by (2.43) and the remaining errors are then found using (2.44). Finally, the partial derivatives of the gradient vector are found by (2.45) and (2.46).

This backpropagation technique combined with mini-batch stochastic gradient descent or other related optimizers form the essence of how neural networks work.

2.5 Evaluation and Overfitting

The goal of training a neural network is to not only achieve good accuracy on the training set but also to "generalize", i.e. to achieve good accuracy on new data not in the training set, making it more difficult than in optimization settings. To evaluate how the network performs on unseen data, after training on the training set is completed, the network is evaluated on a *test set.*

Definition 16 (Test set)**.** *A test set* $\mathcal{T}\left\{\left(\boldsymbol{x}^{\{i\}}, \boldsymbol{y}^{\{i\}}\right)\right\}_{i=1}^{M}$ *is a set of examples, where* M *is the number of examples. The examples are taken from the same distribution as those in the training set.*

The training set and test can be thought of as two disjoint partitions of a larger set containing examples $\left(\boldsymbol{x}^{\{i\}}, \boldsymbol{y}^{\{i\}}\right)$. The *i.i.d. assumptions* are typically made on these datasets, with it being assumed that all samples are independent from each other and drawn from the same probability distribution. Typically about 70% of examples are used for the training set and 30% for the test set. Ultimately, the goal of machine learning is to find the weights and biases $\boldsymbol{\theta}$ that minimize the error on the test set (maximize accuracy).

Definition 17. *Error is the percentage of examples classified incorrectly,*

$$Error = \frac{1}{M} \sum_{i=1}^{M} \left[\hat{\boldsymbol{y}}^{\{i\}} \neq \boldsymbol{y}^{\{i\}} \right]. \tag{2.59}$$

Definition 18. *Accuracy is the percentage of examples classified correctly,*

$$Accuracy = \frac{1}{M} \sum_{i=1}^{M} \left[\hat{\boldsymbol{y}}^{\{i\}} = \boldsymbol{y}^{\{i\}} \right]. \tag{2.60}$$

The expected test error will always be greater than or equal to the training error. In order to attain the overall goal of minimizing the test error, the neural network must achieve two goals:

1. Minimize the training error.

2. Minimize the gap between the test error and training error.

These two critera correspond to two possible problems:

1. *Underfitting:* The training error is too large.

2. *Overfitting:* The gap between test and training error is too large.

Overfitting is a more common problem. Since neural networks can be very powerful with millions of parameters, it is possible for them to simply memorize the training data, allowing them to achieve high accuracy on the training set but generalize poorly. There exist several techniques to reduce overfitting; one is to simply increase the amount of training data, assuming the data is randomly sampled and there is no repetition of samples. However, datasets are often limited and this is not always possible. Another way is to modify the algorithm by adding a *regularization* term to the cost function. This term is designed to constrain the model to something more simple, penalizing overly complex models. There exist many different regularization methods including ridge regression and lasso regression.

2.5.1 Early Stopping

One of the most common ways to reduce overfitting is called *early stopping.* As training progresses over time, the loss and error rate on the training set typically continues to decrease but at some point the loss and error on the test set begins to increase as the network begins to overfit. This is illustrated in Figure 2.4.

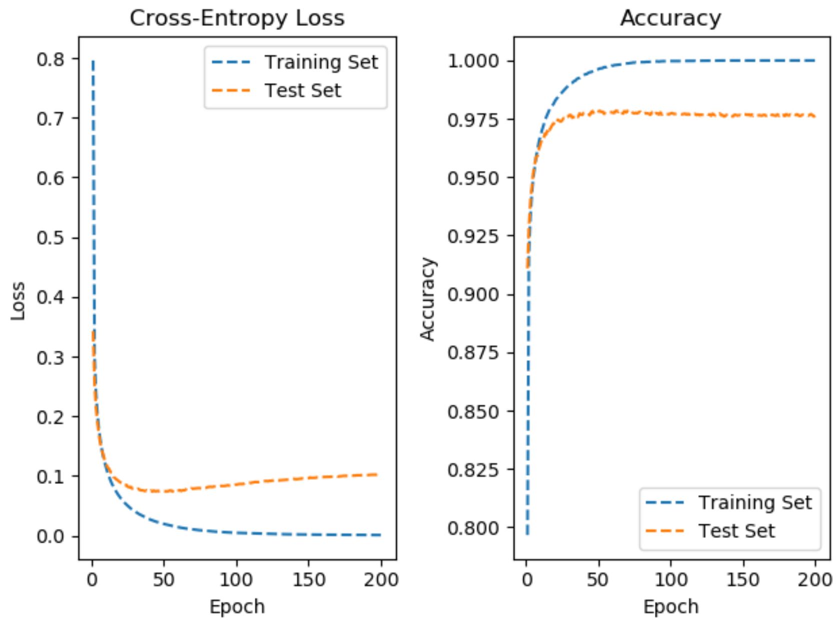


Figure 2.4: Loss (left) and accuracy (right) classifying images from the MNIST dataset [16]. On the training set, the loss continues to decrease and accuracy continues to increase as training progresses. On the test set, loss decreases and accuracy increases during the initial epochs of training, but after a certain number of epochs performance begins to worsen.

The goal of early stopping is to find the point in time where overfitting begins and to stop training. This is done by continuously evaluating the performance of the network (typically loss or error rate) after each epoch or a certain number of epochs of training. If the performance stops improving, training is stopped. This evaluation is done on a new, separate partition of the data set called the *validation set*.

A validation set $\mathcal{V}\left\{\left(\boldsymbol{x}^{\{i\}}, \boldsymbol{y}^{\{i\}}\right)\right\}_{i=1}^{P}$, where P is the number of examples, is a third partition of the dataset in addition to the training set and test set. Adding a validation set has the disadvantage of reducing the amount of data available for the training and test sets but is necessary in order to use the network's performance on unseen data to adjust training. Like the test set, the validation set is used to evaluate how the model performs on new, unseen data, but in this case the data influences training, so it is best practice to use a validation set completely disjoint with the test set for the evaluation on the test set to be fair. In general the training set will contain the most data, followed by the test set and then the validation set. The is no optimal split percentage but a common combination is 70% for the training set, 20% for the test set, and 10% for the validation set.

The main parameter of the early stopping algorithm is the `patience`: the number of epochs with no performance improvement before training is ended. For example, a patience of 5 means that 5 epochs with no improvement triggers stopping. These epochs can be non-consecutive. Another is `min_delta`: the minimum change in the monitored quantity between epochs to qualify as an improvement. By default, `min_delta` $= 0$, meaning that no change will not count towards triggering early stopping, but a decrease in performance will. Increasing `min_delta` allows early stopping to also trigger when very minor, negligible improvements occur.

2.6 Preprocessing

Before input data is fed into a neural network, the data usually requires some *preprocessing.* It is important that all input values have the same scale and a small variance. Extremely large or small values and values with a large variance can hurt training.

Images consist of pixel values that are typically 8 bits, meaning each pixel has $2^8 = 256$ possible values. Thus when $\boldsymbol{x} \in \mathbb{R}^n$ is an input image, each component of the vector represents a pixel value, $x_i \in \{0, \dots, 255\}$, for all $i \in \{1, \dots, n\}$. The most simple preprocessing method is to scale the data by normalizing it so that the pixel values are in the range $[0, 1]$. This is done simply by dividing each value by 255. Letting $\boldsymbol{x}^*$ be the scaled data,

$$\boldsymbol{x}^* = \frac{1}{255}\boldsymbol{x}. \tag{2.61}$$

More generally, any input data can be scaled to any range $[l, u]$ by

$$x_i^* = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}(u - l) + l, \quad \forall i \in \{1, \dots, n\}, \tag{2.62}$$

where $x_{\min}$ is the smallest component of $\boldsymbol{x}$, $x_{\max}$ is the largest component of $\boldsymbol{x}$, u is the upper value of the new range, and l is the lower value of the new range. The range $[-1, 1]$ is a common alternative to $[0, 1]$.

Another way to scale the data is to standardize it so that its mean is 0 and standard deviation is 1. This is done by

$$x_i^* = \frac{x_i - \mu}{\sigma}, \quad \forall i \in \{1, \dots, n\}. \tag{2.63}$$

Note that here we overload notation and let $\sigma \in \mathbb{R}_{>0}$ be the standard deviation (rather than an activation function). We let $\mu \in \mathbb{R}$ be the mean, calculated as

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i, \tag{2.64}$$

and calculate the standard deviation

$$\sigma = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)^2}. \tag{2.65}$$

Scaling the data is by far the most important form of preprocessing but there exist some other, such as cropping and centering images to ensure they have the same size and aspect ratio.

2.7 Weight Initialization

At the very beginning of training, all weights and biases in the network are initialized at random. There are many different initialization schemes with different advantages and disadvantages. Weights are typically chosen from either a normal or a uniform distribution. One criteria that should be fulfilled is that all inputs to a node should be given a different initial weight. Initialization is important and can greatly effect the time it takes for the network to converge. Weights that are too small can cause an input to "vanish" or become so small it is no longer useful, while weights that are too large can cause the input to "explode" or become so large it is no longer useful.

One of the most common initialization schemes is Xavier initialization [30], also called Glorot initialization (named after Xavier Glorot who proposed the method). The goal of this method is initialize weights such that the variance of inputs is the same in each layer, $Var(z_i^{(l-1)}) = Var(z_j^{(l)}), \forall i \in \{1, \cdots, n_{l-1}\},\ j \in \{1, \cdots, n_l\}$,

$l \in \{2, \cdots, L\}$. This constant variance helps prevent gradients from vanishing or exploding. There is both a *Xavier Normal* and *Xavier Uniform* version of this initialization scheme, which uses a normal distribution and a uniform distribution, respectively. Xavier Normal initialization consists of initializing weights in a network's layer l using a normal distribution with 0 mean and a variance of $\frac{2}{n_{l-1}+n_l}$,

$$w_{ij}^{(l)} \sim \mathcal{N}\left(0, \frac{2}{n_{l-1}+n_l}\right), \tag{2.66}$$

recalling that n_l is the number of nodes in layer l. Xavier Uniform initialization consists of initializing these weights using a uniform distribution with a variance of $\frac{2}{n_{l-1}+n_l}$,

$$w_{ij}^{(l)} \sim \text{Uniform}\left[-\frac{\sqrt{6}}{\sqrt{n_{l-1}+n_l}}, \frac{\sqrt{6}}{\sqrt{n_{l-1}+n_l}}\right]. \tag{2.67}$$

The key idea behind these initialization schemes is that the variance of the distribution is $\frac{2}{n_{l-1}+n_l}$; the normal and uniform versions are used interchangeably in practice and it remains unclear if one is better than the other.

A second common initialization scheme is He initialization [41], also called Kaiming initialization (named after its creator Kaiming He). This scheme has the same goal as Xavier initialization of keeping the variances of inputs constant across layers but is better adapted to the ReLU activation function, while Xavier initialization is better adapted to the tanh or sigmoid activation function. Like Xavier initialization, there exists both a *He Normal* and *He Uniform* version of this initialization scheme that uses a normal distribution and a uniform distribution. Both distributions have the same variance of $\frac{2}{n_{l-1}}$. These two initialization schemes, respectively, are

$$w_{ij}^{(l)} \sim \mathcal{N}\left(0, \frac{2}{n_{l-1}}\right), \tag{2.68}$$

$$w_{ij}^{(l)} \sim \text{Uniform}\left[-\sqrt{\frac{6}{n_{l-1}}}, \sqrt{\frac{6}{n_{l-1}}}\right]. \tag{2.69}$$

2.8 Convolutional Neural Networks

For classifying images or other data in grid-like patterns, convolutional neural networks (CNNs) are typically used rather than the fully-connected neural networks previously discussed. In fully connected networks, each node is connected to every node in the next layer with a unique weight, and regular matrix multiplication takes place between an input matrix and a weight matrix. By contrast, convolutional neural networks contain layers where instead a convolution operation is performed between the input matrix and weight matrix. As we are working with discrete image data, we consider discrete convolution.

Definition 19 (Discrete convolution)**.** *For two functions $x(t)$ and $w(t)$ defined on a discrete integer t, the discrete convolution $(x * w)(t)$ is defined as*

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a), \tag{2.70}$$

where $a \in \mathbb{R}^n$ and the summation is over all dimensions.

The first argument x to the convolution operation is called the *input* and the second argument w is called the *kernel* or the *filter*. The input x consists of image pixel values while w consists of weights.

For a two dimensional image I as the input, a two dimensional kernel K is typically used, and the convolution is

$$(I * K)(i, j) = \sum_m \sum_n I(m, n) K(i - m, j - n), \tag{2.71}$$

or equivalently

$$(I * K)(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n) \tag{2.72}$$

as convolution is commutative.

Typically the kernel is much smaller than the input, which reduces memory usage and computational power compared to fully-connected networks. Just like with fully-connected networks, the output of the convolution is passed through a non-linear activation function and the weights of the kernel are learned with gradient descent and backpropagation. This final output out of a convolutional layer is known as a *feature map.* Convolution can be viewed as multiplication by a matrix where several entries are constrained to be equal to other entries, such as a *Toeplitz matrix* or *circulant matrix* [33].

Typically *pooling* layers are used in conjunction with convolutional layers. A pooling layer takes in the output of a convolutional layer and gives averages across different rectangular neighbourhoods of this output. Popular pooling layers include the *max pooling* layer which returns the maximal pixel value in each neighbourhood, the *average pooling* layer which returns the mean of the values, and the L^2 *pooling* layer which returns the L^2 norm of the values in the neighbourhood. Max pooling is the one used most commonly.

In addition to reducing the size of the output which reduces memory usage and computational power since the next layer has less inputs to process, pooling has the benefit of making the convolutional operation approximately *invariant* to small translations of the input. This is useful for applications where the exact location of a feature can slightly vary. For example, when classifying faces, the location of the eye can vary by several pixels on each image. The network needs to determine whether or not the eye is present in the image but the exact location is not important [33].

2.9 ResNet

One of the most widely used type of neural network is ResNet [42], short for *residual neural network*. ResNet is a variation of convolutional neural networks. ResNet adds *shortcut* connections between layers, also known as *skip* connections. These connections consist of the identity mapping, and their output is added to a layer further in the network, allowing input data to "skip" several layers.

Definition 20 (Residual function)**.** *If $\boldsymbol{x}$ is the input to a neural network's layer and f is the function that the several layers of that network are fitting, then $h(\boldsymbol{x}) = f(\boldsymbol{x}) - \boldsymbol{x}$ is the residual function.*

The original function is then calculated as $f(\boldsymbol{x}) = h(\boldsymbol{x}) + \boldsymbol{x}$. The network learns the residual function $h(\boldsymbol{x})$ and $\boldsymbol{x}$ is added as the skip connection rather than the network directly learning $f(\boldsymbol{x})$. In the original work [42], the authors hypothesize that it is easier to optimize the residual function compared to the original mapping. For example, in the case where the optimal mapping is the identity, it is easier for neural network to learn the 0 mapping for $h(\boldsymbol{x})$ rather than directly learning the identity mapping for $f(\boldsymbol{x})$. Figure 2.5 shows an example of two layers in a residual neural network with a skip connection.

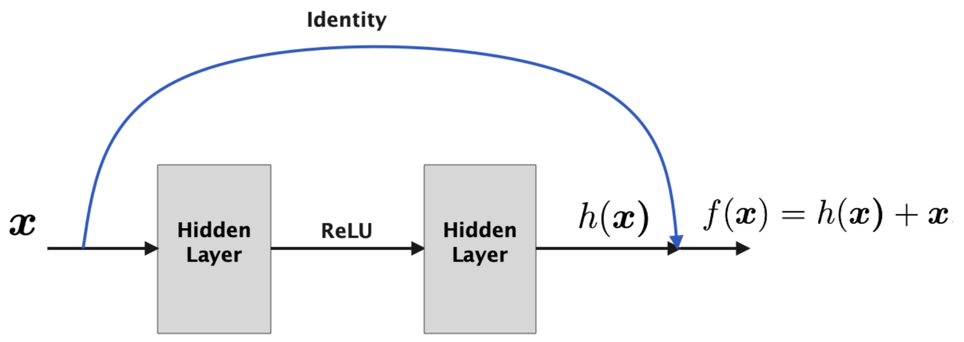


Figure 2.5: The two hidden layers of the network learn the residual mapping $h(\boldsymbol{x})$ and then $\boldsymbol{x}$ is added to the output, with the identity mapping skip connection bypassing the two layers. This results in a final output of $h(\boldsymbol{x}) + \boldsymbol{x}$.

Note that the dimension of $\boldsymbol{x}$ must be equal to the dimension of $h(\boldsymbol{x})$. If this is not the case, a linear projection $\boldsymbol{W_s}$ is performed on the identity skip connection so that the dimensions match, and the output of the block is given by

$$\boldsymbol{z} = h(\boldsymbol{x}) + \boldsymbol{W_s x}. \tag{2.73}$$

Typically the skip connection is used over two or three layers, though it is possible to use it over a higher number of layers. There is no advantage to using it over a single layer. ResNet was shown to outperform other methods, achieving state of the art results on ImageNet [15], a dataset containing more than 14 million images in over 20,000 categories that is used in computer vision competitions [83].

While residual neural networks can have any number of layers, ResNet typically refers to specific network architectures described in the original ResNet paper [42]. These are known as ResNet18, ResNet34, ResNet50, ResNet101, and ResNet152, where the number is the total number of layers in the network. Each of these networks consist of several blocks of convolutional layers with skip connections. The best accuracy was achieved on the largest networks with the most layers; however using a larger network comes at the expense of requiring significantly more computational power and accuracy improvements become minor amongst the largest networks.

2.9.1 Transfer Learning

Rather than learning a model from scratch, it is possible to begin with a ResNet model pretrained on ImageNet and then just perform a minor amount of training to adapt the model to a specific application. This is a form of *transfer learning*, applying knowledge learned from one setting to a different setting [33]. The idea behind transfer learning is that some features are common in all images so a neural network trained to recognize them does not need to be retrained for new datasets. It is thought that initial layers of the network learn more generic features of images and the final layers learn more specific features. Thus, when presented with a new dataset, it is possible to take a pretrained ResNet network, remove the last several layers and replace them with new ones, and then only train these layers rather than training the entire network. This has the benefit of significantly reducing computational time as only a few layers need to be trained.

Chapter 3

Variational Autoencoders

Variational Autoencoders (VAEs) [53, 54] are a type of deep learning architecture that encodes data into a new representation and then decodes it to generate data similar to the original. Along with Generative Adversarial Networks (GANs) [34], VAEs are one of the most popular generative models that can be used to create new, realistic content such as images, text, video, or sound.

3.1 Variational Autoencoder Objective

The variational autoencoder is a *latent variable* model.

Definition 21 (Latent variable)**.** *A latent variable* $\boldsymbol{z} \in \mathbb{R}^m$ *is a hidden variable that is part of the model but is not directly observed and not part of the dataset.*

As before, we assume we have a dataset of input data $\mathcal{D}\left\{\boldsymbol{x}^{\{i\}}\right\}_{i=1}^{N}$ containing N i.i.d. samples of a random variable $\boldsymbol{x}$. We also assume the data is generated by a random process involving an unobserved latent random variable $\boldsymbol{z}$, and all distributions are parameterized by $\boldsymbol{\theta}$. Note that as before, when we represent a distribution by a neural network then $\boldsymbol{\theta}$ is the network's weights and biases, but we overload notation to have $\boldsymbol{z}$ be a latent variable rather than a weighted input to a neural network's layer.

The generative process can be described as follows. For each datapoint $i \in \{1, \ldots, N\}$,

- We draw the latent variable $\boldsymbol{z}^{\{i\}} \sim P_Z(\boldsymbol{z}; \boldsymbol{\theta})$.
- We then draw an input datapoint $\boldsymbol{x}^{\{i\}} \sim P_{X|Z}(\boldsymbol{x}|\boldsymbol{z}; \boldsymbol{\theta})$.

Both these distributions are unknown and we parameterize them with neural networks. The distribution $P_X(\boldsymbol{x}; \boldsymbol{\theta})$ is given by

$$P_X(\boldsymbol{x}; \boldsymbol{\theta}) = \int P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) d\boldsymbol{z} = \int P_{X|Z}(\boldsymbol{x}|\boldsymbol{z}; \boldsymbol{\theta}) P_Z(\boldsymbol{z}; \boldsymbol{\theta}) d\boldsymbol{z}. \tag{3.1}$$

When taken as a function of $\boldsymbol{\theta}$, this distribution is also known as the *marginal likelihood* or the *model evidence.*

The variational autoencoder aims to solve several different problems:

1. We wish to perform maximum likelihood estimation of the parameters $\boldsymbol{\theta}$. Letting Θ be the parameter space of $\boldsymbol{\theta}$, we want to find the $\hat{\boldsymbol{\theta}}$ that maximizes the average (or equivalently the sum) of the log-probabilities assigned to the data by the model:
$$\hat{\boldsymbol{\theta}} = \max_{\boldsymbol{\theta} \in \Theta} \sum_{i=1}^{N} \log P_X(\boldsymbol{x}^{\{i\}}; \boldsymbol{\theta}). \tag{3.2}$$
This allows us to generate new, artificial input datapoints, mimicking the hidden random process.

2. We wish to find a procedure to encode an input datapoint $\boldsymbol{x}$ into $\boldsymbol{z}$ for a fixed value of $\boldsymbol{\theta}$. This requires us to find $P_{Z|X}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\theta})$.

This has to be done without simplifying assumptions about the marginal or posterior probabilities:

- We consider the integral of the marginal likelihood

$$P_X(\boldsymbol{x};\boldsymbol{\theta}) = \int P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})P_Z(\boldsymbol{z};\boldsymbol{\theta})d\boldsymbol{z} \tag{3.3}$$

  to be intractable, so the marginal likelihood cannot be evaluated.

- We also consider the true posterior density $P_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta})$ to be intractable. As

$$P_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta}) = \frac{P_{Z,X}(\boldsymbol{z},\boldsymbol{x};\boldsymbol{\theta})}{P_X(\boldsymbol{x};\boldsymbol{\theta})}, \tag{3.4}$$

  and $P_{Z,X}(\boldsymbol{z},\boldsymbol{x};\boldsymbol{\theta})$ is tractable, $P_X(\boldsymbol{x};\boldsymbol{\theta})$ being intractable leads to $P_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta})$ being intractable, and vice-versa.

To solve this problem, we use variational inference and introduce a parametric inference model $Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})$ where $\boldsymbol{\phi}$ are the parameters of the model, also called the *variational parameters.* The distribution $Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})$ is known as the *encoder*, as the input $\boldsymbol{x}$ is encoded into a new representation $\boldsymbol{z}$ (which can be thought of as a code), while $P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})$ is known as the *decoder*, as it decodes $\boldsymbol{z}$ back into the original data $\boldsymbol{x}$. The distributions $Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})$ and $P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})$ are each represented by a neural network, and $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$ are the weights and biases of those networks.

We hope to use $Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})$ as an approximation of $P_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta})$, i.e., to find $\boldsymbol{\phi}$ such that $Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi}) \approx P_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\theta})$. Given this approximation, and using Jensen's Inequality, which we recall below, we derive a variational lower bound on $\log P_X(\boldsymbol{x};\boldsymbol{\theta})$, c.f. Theorem 5.

Theorem 4 (Jensen's Inequality)**.** *Let $\mathcal{X} \subset \mathbb{R}^n$ be a convex subset of $\mathbb{R}^n$, and let X be a random variable with alphabet in $\mathcal{X}$. If $f : \mathcal{X} \to \mathbb{R}$ is concave over $\mathcal{X}$ then*

$$\mathbb{E}[f(X)] \leq f(\mathbb{E}[X]). \tag{3.5}$$

Theorem 5. *The following is a lower bound on $\log P_X(\boldsymbol{x};\boldsymbol{\theta})$:*

$$\log P_X(\boldsymbol{x};\boldsymbol{\theta}) \geq \mathbb{E}_{Q_{Z|X}}\left[\log P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})\right] - D_{KL}\left(Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})||P_Z(\boldsymbol{z};\boldsymbol{\theta})\right). \tag{3.6}$$

Proof. We have

$$\begin{aligned}
\log P_X(\boldsymbol{x};\boldsymbol{\theta}) &= \log \int P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})P_Z(\boldsymbol{z};\boldsymbol{\theta})d\boldsymbol{z} \qquad (3.7)\\
&= \log \int \frac{Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})}{Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})} P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})P_Z(\boldsymbol{z};\boldsymbol{\theta})d\boldsymbol{z}\\
&= \log \mathbb{E}_{Q_{Z|X}}\left[\frac{P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})P_Z(\boldsymbol{z};\boldsymbol{\theta})}{Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})}\right]\\
&\geq \mathbb{E}_{Q_{Z|X}}\left[\log \frac{P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})P_Z(\boldsymbol{z};\boldsymbol{\theta})}{Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})}\right]\\
&= \mathbb{E}_{Q_{Z|X}}\left[\log P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})\right] + \mathbb{E}_{Q_{Z|X}}\left[\log \frac{P_Z(\boldsymbol{z};\boldsymbol{\theta})}{Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})}\right]\\
&= \mathbb{E}_{Q_{Z|X}}\left[\log P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})\right] - D_{KL}\left(Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})||P_Z(\boldsymbol{z};\boldsymbol{\theta})\right),
\end{aligned}$$

where the inequality follows from Jensen's inequality as the logarithmic function is concave. □

This lower bound is known as the Evidence Lower Bound (ELBO) [54].

Definition 22. *The Evidence Lower Bound (ELBO) is defined as*

$$\mathcal{L}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \mathbb{E}_{Q_{Z|X}}\left[\log P_{X|Z}(\boldsymbol{x}|\boldsymbol{z};\boldsymbol{\theta})\right] - D_{KL}\left(Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})||P_Z(\boldsymbol{z};\boldsymbol{\theta})\right) \tag{3.8}$$

$$= \mathbb{E}_{Q_{Z|X}}\left[\log P_{X,Z}(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})\right]. \tag{3.9}$$

The objective of the variational autoencoder is to maximize the ELBO over both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. This is done as an average for all input datapoints in the dataset - we find values for $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ that solve

$$\max_{\boldsymbol{\theta},\boldsymbol{\phi}} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{x}^{\{i\}};\boldsymbol{\theta},\boldsymbol{\phi}). \tag{3.10}$$

3.2 Computing Gradients

To optimize this objective using stochastic gradient descent, we need to take gradients with respect to both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. To do this with respect to $\boldsymbol{\theta}$ is possible:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{x};\boldsymbol{\theta},\boldsymbol{\phi}) = \nabla_{\boldsymbol{\theta}}\mathbb{E}_{Q_{Z|X}}\left[\log P_{X,Z}(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})\right] \tag{3.11}$$

$$= \mathbb{E}_{Q_{Z|X}}\left[\nabla_{\boldsymbol{\theta}}\left[\log P_{X,Z}(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})\right]\right] \tag{3.12}$$

$$\simeq \nabla_{\boldsymbol{\theta}}\left[\log P_{X,Z}(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})\right] \tag{3.13}$$

$$= \nabla_{\boldsymbol{\theta}}\left[\log P_{X,Z}(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta})\right]. \tag{3.14}$$

For the third line, we estimate the expectation using a Monte Carlo estimate, given by

$$\mathbb{E}_{Q_{Z|X}}\left[\nabla_{\boldsymbol{\theta}}\left[\log P_{X,Z}(\boldsymbol{x},\boldsymbol{z};\boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x};\boldsymbol{\phi})\right]\right] \tag{3.15}$$

$$\simeq \frac{1}{L}\sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}} \left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}^{\{l\}}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}^{\{l\}}|\boldsymbol{x}; \boldsymbol{\phi})\right], \tag{3.16}$$

where $\boldsymbol{z}^{\{l\}} \sim Q_{Z|X}$. Typically a single sample is used; i.e., $L = 1$.

However, to take gradients with respect to $\boldsymbol{\phi}$ is not possible since the expectation is taken with respect to a function of $\boldsymbol{\phi}$. In general,

$$\nabla_{\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}}\mathbb{E}_{Q_{Z|X}} \left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\phi})\right] \tag{3.17}$$

$$\neq \mathbb{E}_{Q_{Z|X}} \left[\nabla_{\boldsymbol{\phi}} \left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\phi})\right]\right]. \tag{3.18}$$

Therefore, Monte Carlo estimation cannot be used to compute the gradient. For this to be possible, a change of variables is used called the *reparameterization trick* [53].

3.3 Reparameterization Trick

Definition 23 (Reparameterization Trick)**.** *The reparameterization trick is to express $\boldsymbol{z} \sim Q_{Z|X}$ as a deterministic function f of another random variable $\boldsymbol{\epsilon} \sim P_E$ as well as $\boldsymbol{x}$ and $\boldsymbol{\phi}$, and where the distribution of P_E is independent of $\boldsymbol{x}$ and $\boldsymbol{\phi}$:*

$$\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\epsilon}; \boldsymbol{\phi}). \tag{3.19}$$

Example 3. *Let $\boldsymbol{z} \in \mathbb{R}^m$ be a Gaussian random variable $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{\phi}), diag(\boldsymbol{\sigma}^2(\boldsymbol{x}, \boldsymbol{\phi})))$, where $\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{\phi}) \in \mathbb{R}^m$ is the mean and $diag(\boldsymbol{\sigma}^2(\boldsymbol{x}, \boldsymbol{\phi})) \in \mathbb{R}^{m\times m}$ is a diagonal covariance matrix, where the mean and variance $\boldsymbol{\sigma}^2(\boldsymbol{x}, \boldsymbol{\phi}) \in \mathbb{R}^m$ are the outputs of a neural network with parameters $\boldsymbol{\phi}$ and input $\boldsymbol{x}$. Then this can be reparameterized as $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\epsilon}; \boldsymbol{\phi}) = \boldsymbol{\sigma}(\boldsymbol{x}, \boldsymbol{\phi}) \odot \boldsymbol{\epsilon} + \boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{\phi})$, where f is a deterministic function and*

$\boldsymbol{\epsilon} \in \mathbb{R}^m$ is a spherical Gaussian random variable, i.e., $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_m)$, where $\mathbf{0}$ is the m-dimensional zero vector and $\boldsymbol{I}_m$ the m-dimensional identity matrix.

The reparameterization trick allows us to take the gradient of the ELBO (3.9) with respect to $\boldsymbol{\phi}$ as the expectation can be rewritten in terms of P_E, where P_E is a distribution that is independent of $\boldsymbol{x}$ or $\boldsymbol{\phi}$, as follows:

$$\begin{aligned}
&\mathbb{E}_{Q_{Z|X}}\left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\phi})\right] && (3.20)\\
&= \mathbb{E}_{P_E}\left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\epsilon}; \boldsymbol{\phi}); \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\epsilon}; \boldsymbol{\phi})|\boldsymbol{x}; \boldsymbol{\phi})\right]. && (3.21)
\end{aligned}$$

This allows the expectation and gradient operators to be commutative, allowing us to perform Monte Carlo estimation. Letting $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\epsilon}; \boldsymbol{\phi})$, we have

$$\begin{aligned}
\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) &= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{Q_{Z|X}}\left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\phi})\right] && (3.22)\\
&= \nabla_{\boldsymbol{\phi}} \mathbb{E}_{P_E}\left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\phi})\right] && (3.23)\\
&= \mathbb{E}_{P_E}\left[\nabla_{\boldsymbol{\phi}}\left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\phi})\right]\right] && (3.24)\\
&\simeq \frac{1}{L}\sum_{l=1}^{L}\left[\nabla_{\boldsymbol{\phi}}\left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}^{\{l\}}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}^{\{l\}}|\boldsymbol{x}; \boldsymbol{\phi})\right]\right], && (3.25)
\end{aligned}$$

where $\boldsymbol{z}^{\{l\}}$ is calculated as the following:

- $\boldsymbol{\epsilon}^{\{l\}}$ is sampled as $\boldsymbol{\epsilon}^{\{l\}} \sim P_E$.
- $\boldsymbol{z}^{\{l\}}$ is calculated with the deterministic function $\boldsymbol{z}^{\{l\}} = f(\boldsymbol{x}, \boldsymbol{\epsilon}^{\{l\}}; \boldsymbol{\phi})$.

As typically a single sample is used for the Monte Carlo estimation, $L = 1$, and (3.22) simply becomes

$$\nabla_{\boldsymbol{\phi}} \mathcal{L}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) \simeq \nabla_{\boldsymbol{\phi}} \left[\log P_{X,Z}(\boldsymbol{x}, \boldsymbol{z}; \boldsymbol{\theta}) - \log Q_{Z|X}(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{\phi}) \right]. \tag{3.26}$$

Chapter 4

Information Bottleneck Methods

The Information Bottleneck (IB) Method, originally developed by Tishby *et. al.* [90], is an information-theoretic technique to compress a random variable as much as possible while maintaining information relevant about another random variable. It has been hypothesized that the IB method provides a theoretical explanation of how deep learning works [91]. In this section we discuss the IB method and how it can be applied to neural networks.

4.1 Information Bottleneck Method

Considering two dependent random variables X and Y, the IB method aims to find a new code Z that maximally compresses X while simultaneously retaining the information from X necessary to predict Y. This is expressed in terms of mutual information: it is desired to find a Z such that $I(Z;X)$ is minimized while $I(Z;Y)$ is maximized. This can be thought of as “squeezing” the information that X contains about Y out through a bottleneck formed by a limited set of codewords for Z.

There will always exist a trade-off between the amount of compression and the amount of relevant information preserved about Y. This is captured through the

Information Bottleneck Lagrangian, which is minimized over $P_{Z|X}$:

$$\mathcal{L} = I(Z;X) - \beta I(Z;Y). \tag{4.1}$$

Here β is a positive Lagrange multiplier that controls the trade-off between compression and the retention of relevant information about Y.

- As $\beta \to 0$, the constraint on preserving relevant information is removed, resulting in maximal compression with no relevant information retained.
- As $\beta \to \infty$, the constraint becomes that all information about Y has to be preserved, resulting in no compression with the trivial encoding $Z = X$.

By varying β it is possible to examine the trade-off between compression and preserved relevant information.

4.2 Deep Variational Information Bottleneck

Alemi *et. al.* [2] developed a variational approximation of the original IB method and applied it to neural networks. The original IB method requires computing mutual information which is computationally expensive in general, with the exception of scenarios where the variables are discrete or jointly Gaussian. The Deep Variational Information Bottleneck uses variational approximations to compute the mutual information terms, allowing the method to be used with continuous random variables.

The Deep Variational IB method can be used with neural networks: we consider X to be the original, high dimensional input to a neural network such as an image, while Y is the target that the network tries to predict from X. The method aims find a compressed, lower-dimensional encoding of X that still allows Y to be predicted accurately. This encoding Z is defined by the distribution $P_{Z|X}$, and the distribution is modelled by a neural network. A second neural network then predicts Y based on Z, modelling the distribution $P_{Y|Z}$. As all the information about Y is contained in X, Y implicitly determines what information in X should be retained, and the Markov chain $Y \to X \to Z$ is formed.

One potential practical benefit of using this method rather than directly predicting Y from X is a potential accuracy improvement due to improved generalization. It has been theorized that compressing X into Z acts as a regularization method, preventing the neural network from memorizing too much information about X and reducing overfitting. It has also been claimed that this method increases robustness to adversarial attack [2]. Adversarial attacks involve purposefully changing inputs to a neural network by a small amount to trick the network into misclassifying the input [35].

4.2.1 Variational Bounds

Variational bounds are computed by using the non-negativity of KL divergence similarly to what was done for the variational autoencoder. To compute an upper bound on the $I(Z;X)$ term to be minimized, the distribution P_Z is replaced with a variational approximation Q_Z, as in the following.

$$I(Z;X) = \sum_{(\boldsymbol{z},\boldsymbol{x}) \in \mathcal{Z} \times \mathcal{X}} P_{Z,X}(\boldsymbol{z},\boldsymbol{x}) \log \frac{P_{Z|X}(\boldsymbol{z}|\boldsymbol{x})}{P_Z(\boldsymbol{z})}$$

$$\begin{aligned}
&= \sum_{(\boldsymbol{z},\boldsymbol{x})\in\mathcal{Z}\times\mathcal{X}} P_{Z,X}(\boldsymbol{z},\boldsymbol{x}) \log P_{Z|X}(\boldsymbol{z}|\boldsymbol{x}) \\
&\quad - D_{KL}(P_Z \| Q_Z) - \sum_{\boldsymbol{z}\in\mathcal{Z}} P_Z(\boldsymbol{z}) \log Q_Z(\boldsymbol{z}) \\
&\leq \sum_{(\boldsymbol{z},\boldsymbol{x})\in\mathcal{Z}\times\mathcal{X}} P_{Z,X}(\boldsymbol{z},\boldsymbol{x}) \log \frac{P_{Z|X}(\boldsymbol{z}|\boldsymbol{x})}{Q_Z(\boldsymbol{z})} \\
&= \mathbb{E}_{P_X} D_{KL}\left(P_{Z|X} \| Q_Z\right). \qquad (4.2)
\end{aligned}$$

Note that $P_{Z|X}$ is modeled by a neural network with parameters $\boldsymbol{\theta}$, but we omit this to simplify notation. We choose a distribution for Q_Z (typically a spherical Gaussian is used).

To compute a lower bound on the $I(Z;Y)$ term to be maximized, the distribution $P_{Y|Z}$ is replaced with a variational approximation $Q_{Y|Z}$ as we demonstrate next.

$$\begin{aligned}
I(Z;Y) &= H(Y) - H(Y|Z) \\
&= H(Y) + \sum_{(y,\boldsymbol{z})\in\times\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log P_{Y|Z}(y|\boldsymbol{z}) \\
&= H(Y) + \sum_{(y,\boldsymbol{z})\in\times\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log P_{Y|Z}(y|\boldsymbol{z}) \\
&\qquad - \sum_{(y,\boldsymbol{z})\in\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log Q_{Y|Z}(y|\boldsymbol{z}) \\
&\qquad + \sum_{(y,\boldsymbol{z})\in\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log Q_{Y|Z}(y|\boldsymbol{z}) \\
&= H(Y) + \mathbb{E}_{P_Z} D(P_{Y|Z}||Q_{Y|Z}) + \sum_{(y,\boldsymbol{z})\in\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log Q_{Y|Z}(y|\boldsymbol{z}) \\
&\geq H(Y) + \sum_{(\boldsymbol{x},y,\boldsymbol{z})\in\mathcal{X}\times\mathcal{Y}\times\mathcal{Z}} P_{Z|X}(\boldsymbol{z}|\boldsymbol{x}) P_{X,Y}(\boldsymbol{x},y) \log Q_{Y|Z}(y|\boldsymbol{z}). \qquad (4.3)
\end{aligned}$$

The distribution $Q_{Y|Z}$ is modelled by another neural network parameterized by $\boldsymbol{\phi}$

which we also omit to simplify notation. As $H(Y)$ is independent of the parameterization, it can be ignored for the optimization. In the final line, $P_{X,Y,Z}(\boldsymbol{x}, y, \boldsymbol{z}) = P_{Z|X}(\boldsymbol{z}|\boldsymbol{x})P_{X,Y}(\boldsymbol{x}, y)$ because of the Markov chain $Y \to X \to Z$.

Combining these two bounds results in

$$\begin{aligned} &I(Z;X) - \beta I(Z;Y) \\ &\leq \mathbb{E}_{P_X} D_{KL}\left(P_{Z|X}||Q_Z\right) - \beta \sum_{(\boldsymbol{x},y,\boldsymbol{z})\in\mathcal{X}\times\mathcal{Y}\times\mathcal{Z}} P_{Z|X}(\boldsymbol{z}|\boldsymbol{x})P_{X,Y}(\boldsymbol{x}, y)\log Q_{Y|Z}(y|\boldsymbol{z}) = \mathcal{L}. \end{aligned} \tag{4.4}$$

To compute this bound, first we approximate P_X and $P_{X,Y}$ using their empirical densities. The distributions are approximated $P_{X,Y}(\boldsymbol{x}, y) \approx \frac{1}{N}\sum_{i=1}^{N} \delta\left(\boldsymbol{x}^{\{i\}}, \boldsymbol{x}\right)\delta\left(y^{\{i\}}, y\right)$, using the N samples in a batch from the training set, where $\delta\left(\boldsymbol{x}^{\{i\}}, \boldsymbol{x}\right)$ is the Kronecker delta function,

$$\delta\left(\boldsymbol{x}^{\{i\}}, \boldsymbol{x}\right) = \begin{cases} 0 & \text{if } \boldsymbol{x}^{\{i\}} \neq \boldsymbol{x} \\ 1 & \text{if } \boldsymbol{x}^{\{i\}} = \boldsymbol{x} \end{cases}. \tag{4.5}$$

Considering a batch with N samples $\mathcal{D} = \{\boldsymbol{x}^{\{i\}}, y^{\{i\}}\}_{i=1}^{N}$, this results in

$$\mathcal{L} \approx \frac{1}{N}\sum_{i=1}^{N}\left[D_{KL}\left(P_{Z|X}||Q_Z\right) - \beta\sum_{\boldsymbol{z}\in\mathcal{Z}} P_{Z|X}(\boldsymbol{z}|\boldsymbol{x}^{\{i\}})\log Q_{Y|Z}(y^{\{i\}}|\boldsymbol{z})\right]. \tag{4.6}$$

Finally, we use the reparameterization trick. As with the variational autoencoder, we let $\boldsymbol{z}$ be a deterministic function of $\boldsymbol{x}$ and a Gaussian random variable $\boldsymbol{\epsilon}$, $\boldsymbol{z} = f(\boldsymbol{x}, \boldsymbol{\epsilon})$, $\boldsymbol{\epsilon} \sim P_E$, and replace $P_{Z|X}$ with P_E. This results in a final cost function to be minimized of

$$J_{IB} = \sum_{i=1}^{N} \left[D_{KL} \left(P_{Z|X=\boldsymbol{x}^{\{i\}}} || Q_Z \right) - \beta \mathbb{E}_{P_E} \log Q_{Y|Z}(y^{\{i\}} | f(\boldsymbol{x}^{\{i\}}, \boldsymbol{\epsilon})) \right]. \tag{4.7}$$

Note that there is a large similarity between the Deep Variational Information Bottleneck and the Variational Autoencoder. The cost function of the variational autoencoder to be minimized can be written as

$$\mathcal{L} = \sum_{i=1}^{N} \left[D_{KL} \left(Q_{Z|X=\boldsymbol{x}^{\{i\}}} || P_Z \right) - \mathbb{E}_{P_E} \log P_{X|Z}(\boldsymbol{x}^{\{i\}} | f(\boldsymbol{x}^{\{i\}}, \boldsymbol{\epsilon})) \right]. \tag{4.8}$$

In both cost functions, there is a KL divergence term that acts as a regularization term. Typically the distribution of Z is taken as a spherical Gaussian. Minimizing the KL divergence forces the distribution of Z learned by the neural network to be closer to the spherical Gaussian, hence limiting its richness.

The main difference between the two cost functions is that for the VAE, the network reconstructs X from Z, with $\mathbb{E}_{P_E} \log P_{X|Z}(\boldsymbol{x}^{\{i\}} | f(\boldsymbol{x}^{\{i\}}, \boldsymbol{\epsilon})$ measuring the reconstruction loss, while for IB, the network predicts Y from Z, with the quantity $\mathbb{E}_{P_E} \log Q_{Y|Z}(y^{\{i\}} | f(\boldsymbol{x}^{\{i\}}, \boldsymbol{\epsilon}))$ measuring the prediction error. A second difference is that the IB has the β hyperparameter to control the trade-off between the two terms in the cost function that the original VAE does not. However, there exists a variant of the VAE called the β-VAE [43] that also adds the β hyperparameter to the variational autoencoder, functioning the same way as in the IB method, with $\beta = 1$ corresponding to the original VAE.

4.3 Other IB Variations

There exist several variants of the Variational Information Bottleneck, such as the nonlinear information bottleneck [57] and conditional entropy bottleneck [24], as well as other generalizations [64, 89, 46, 3, 94]. The nonlinear information bottleneck squares the compression term for an objective of

$$\mathcal{L} = I(Z;X)^2 - \beta I(Z;Y). \tag{4.9}$$

The conditional entropy bottleneck adds an additional constraint on the compression term, by adding conditioning on Y to preserve more relevant information about Y in the compressed representation. This is done by replacing $I(Z;X)$ with $I(Z;X|Y)$ for an objective given by

$$\mathcal{L} = I(Z;X|Y) - \beta I(Z;Y). \tag{4.10}$$

Chapter 5

Fairness

In this section we discuss the problem of fairness in machine learning and present our fairness method, Rényi Fair Information Bottleneck (RFIB).

The problem of fairness in machine learning is to obtain accurate predictions of a target of interest while remaining free of bias due to sensitive information such as gender, race, age, or other similar attributes. Representing input data as random variable $X \in \mathcal{X}$, prediction target as random variable $Y \in \mathcal{Y}$, and sensitive information as random variable $S \in \mathcal{S}$, the goal is to predict Y from X in a way that is uninfluenced by S.

5.1 Definitions

5.1.1 Fairness Definitions

We begin by providing several definitions of fairness; there exist many different definitions but the following are some of the most commonly used. Let X be the input variable, Y be the target variable (true label), $\hat{Y}$ be the network's prediction, and S be the protected (sensitive) variable. We assume that Y, $\hat{Y}$, and S are all binary.

Definition 24 (Demographic Parity (DP)). *A classifier satisfies demographic parity if $\hat{Y}$ and S are independent:*

$$P(\hat{Y} = \hat{y}) = P(\hat{Y} = \hat{y}|S = s), \forall s, \hat{y}. \tag{5.1}$$

Demographic parity is also known as *statistical parity.* One disadvantage of demographic parity is that it does not allow the ideal predictor $\hat{Y} = Y$ when Y is correlated with S. In these cases, it can damage utility and cause undesirable effects in some applications, such as unqualified individuals being accepted.

To address these concerns, equalized odds can be used instead, which allows $\hat{Y}$ to depend on S but only through the target variable Y.

Definition 25 (Equalized Odds). *A classifier satisfies equalized odds if $\hat{Y}$ and S are conditionally independent given Y:*

$$P(\hat{Y} = \hat{y}|Y = y) = P(\hat{Y} = \hat{y}|S = s, Y = y), \forall s, y, \hat{y}. \tag{5.2}$$

Equalized odds is also known as *positive rate parity.* For binary Y, it requires both equal *true positive rates* and equal *false positive rates* across all values of S. It allows the existence of perfect classifiers.

Definition 26 (Equality of Opportunity). *A classifier satisfies equality of opportunity if $\hat{Y}$ and S are conditionally independent given a particular y:*

$$P(\hat{Y} = \hat{y}|Y = y) = P(\hat{Y} = \hat{y}|S = s, Y = y), \forall s, \hat{y}. \tag{5.3}$$

This is similar to equalized odds but is a weaker constraint. For binary Y, it requires an equal *true positive rate* across all values of S but can have an unequal *false negative rate.* It also allows the existence of perfect classifiers.

Definition 27 (Accuracy Parity)**.** *A classifier achieves accuracy parity if its accuracy is equal across all subgroups:*

$$P(\hat{Y} = Y|S = 0) = P(\hat{Y} = Y|S = 1). \tag{5.4}$$

It also allows the existence of perfect classifiers, making it suitable for applications where Y is correlated with S.

5.1.2 Metrics

The following are several metrics used to evaluate the performance of classifiers, considering a dataset with N samples $\mathcal{D} = \{\boldsymbol{x}^{\{i\}}, s^{\{i\}}, y^{\{i\}}\}_{i=1}^{N}$. We let $\hat{y}^{\{i\}}$ be the network's prediction of target $y^{\{i\}}$.

Definition 28 (Accuracy [98])**.** *The accuracy is the fraction of correct classifications on a dataset:*

$$\mathit{Accuracy} = P(\hat{Y} = Y) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{\hat{y}^{\{i\}} = y^{\{i\}}\}. \tag{5.5}$$

Definition 29 (Error)**.** *The error is the fraction of incorrect classifications on a dataset:*

$$\mathit{Error} = P(\hat{Y} \neq Y) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}\{\hat{y}^{\{i\}} \neq y^{\{i\}}\} = 1 - \mathit{Accuracy}. \tag{5.6}$$

Definition 30 (Demographic Parity Gap [98])**.** *The demographic parity gap, also known as the* discrimination gap*, measures how close the classifier is to achieving* *demographic parity. It is defined as*

$$\begin{aligned} dp_{\text{gap}} &= \left| P(\hat{Y} = 1 | S = 0) - P(\hat{Y} = 1 | S = 1) \right| \\ &= \left| \frac{\sum_{i:s^{\{i\}}=0} \hat{y}^{\{i\}}}{\sum_{i:s^{\{i\}}=0} 1} - \frac{\sum_{i:s^{\{i\}}=1} \hat{y}^{\{i\}}}{\sum_{i:s^{\{i\}}=1} 1} \right|. \end{aligned} \tag{5.7}$$

Definition 31 (Equalized Odds Gap [103])**.** *The equalized odds gap measures how close the classifier is to achieving equalized odds. It is defined as*

$$\begin{aligned} eqodds_{\text{gap}} &= \max_{y \in \{0,1\}} \left| P(\hat{Y} = 1 | S = 0, Y = y) - P(\hat{Y} = 1 | S = 1, Y = y) \right| \\ &= \max_{y \in \{0,1\}} \left| \frac{\sum_{i:s^{\{i\}}=0, y^{\{i\}}=y} \hat{y}^{\{i\}}}{\sum_{i:s^{\{i\}}=0, y^{\{i\}}=y} 1} - \frac{\sum_{i:s^{\{i\}}=1, y^{\{i\}}=y} \hat{y}^{\{i\}}}{\sum_{i:s^{\{i\}}=1, y^{\{i\}}=y} 1} \right|. \end{aligned} \tag{5.8}$$

Definition 32 (Error Gap [103])**.** *The error gap is the difference in error rate between the two subpopulations of the sensitive variable. It is defined as*

$$\begin{aligned} \textit{Error gap} &= \left| P(\hat{Y} \neq Y | S = 0) - P(\hat{Y} \neq Y | S = 1) \right| \\ &= \left| \frac{\sum_{i:s^{\{i\}}=0} \mathbb{1}\{\hat{y}^{\{i\}} \neq y^{\{i\}}\}}{\sum_{i:s^{\{i\}}=0} 1} - \frac{\sum_{i:s^{\{i\}}=1} \mathbb{1}\{\hat{y}^{\{i\}} \neq y^{\{i\}}\}}{\sum_{i:s^{\{i\}}=1} 1} \right|. \end{aligned} \tag{5.9}$$

Definition 33 (Accuracy Gap)**.** *The accuracy gap is the difference in accuracy between the two subpopulations of the sensitive variable. It is defined as*

$$\begin{aligned} \textit{Accuracy gap} &= \left| P(\hat{Y} = Y | S = 0) - P(\hat{Y} = Y | S = 1) \right| \\ &= \left| \frac{\sum_{i:s^{\{i\}}=0} \mathbb{1}\{\hat{y}^{\{i\}} = y^{\{i\}}\}}{\sum_{i:s^{\{i\}}=0} 1} - \frac{\sum_{i:s^{\{i\}}=1} \mathbb{1}\{\hat{y}^{\{i\}} = y^{\{i\}}\}}{\sum_{i:s^{\{i\}}=1} 1} \right|. \end{aligned} \tag{5.10}$$

As S is binary, the accuracy gap is equal to the error gap.

5.2 Method

We present a fair machine learning method which we call the *Rényi Fair Information Bottleneck* (RFIB). Our method consists of learning *fair representations*, finding an intermediate representation $Z \in \mathcal{Z}$ that can then be used instead of the original data X with existing machine learning architectures to make predictions. The new representation Z must simultaneously preserve information from X relevant to predicting Y while removing sensitive information that could lead to bias.

To reach this objective, we adopt a variational approach to encode X into Z. In light of our model, we assume that the Markov chain $(Y, S) \rightarrow X \rightarrow Z$ holds. To simplify notation, we assume in this section that all random variables are discrete, though a similar derivation holds for a mix of continuous and discrete random variables.

5.2.1 Fairness Defined

Among the three principal definitions of fairness – *demographic parity*, *equalized odds*, and *equality of opportunity* – we focus on addressing both demographic parity and equalized odds since a) equalized odds is related to (but a stronger constraint than) equality of opportunity, and b) demographic parity, also called statistical parity, is an altogether different type of constraint compared to the former two constraints in that the requirement of independence does not involve the actual target label value.

For demographic parity, the goal is for the model's prediction $\hat{Y}$ to be independent of the sensitive variable S, as shown in (5.1), while for equalized odds the goal is to achieve this independence by conditioning on the actual target Y, as in (5.2).

5.2.2 Lagrangian Formulation

To encourage equalized odds, we minimize $I(Z;S|Y)$; i.e., the average amount of information that Z has about S given Y. To both obtain good classification accuracy and help promote demographic parity, we maximize $I(Z;Y|S)$. Maximizing mutual information between Z and Y ensures the representation will be expressive about its target while the conditioning on S ensures that Z does not keep information shared by S, encouraging demographic parity.

In addition, we minimize $I(Z;X|S,Y)$, a compression term similar to one from the IB problem [2]. This minimization further encourages Z to discard information irrelevant for drawing predictions about Y, hence improving generalization capability and reducing the risk of keeping nuisances. Finally, we maximize the utility term $I(Z;Y)$; this optimization, similar to the IB problem, solely ensures the representation is maximally expressive of the target Y.

Combining these terms leads to a Lagrangian, $\mathcal{L}$, that we seek to minimize over the encoding conditional distribution $P_{Z|X}$. The Lagrangian is given by

$$\mathcal{L} = I(Z;S|Y) + I(Z;X|S,Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S), \tag{5.11}$$

where λ_1 and λ_2 are hyperparameters. Developing this Lagrangian, we show that it is equivalent to a simpler expression with fewer terms.

Theorem 6. *Minimizing the Lagrangian in (5.11) over $P_{Z|X}$ is equivalent to minimizing the following expression*

$$\mathcal{L} = I(Z;X) - \beta_1 I(Z;Y) - \beta_2 I(Z;Y|S), \tag{5.12}$$

where $\beta_1 = \lambda_1 + 1 \geq 0$ *and* $\beta_2 = \lambda_2 \geq 0$.

Proof. We have

$$\begin{aligned}
\mathcal{L} &= H(Z|Y) - H(Z|S,Y) + H(Z|S,Y) - H(Z|X,S,Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\
&= H(Z|Y) - H(Z|X) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\
&= H(X) - H(Z,X) - [H(Y) - H(Z,Y)] - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\
&= I(Z;X) - I(Z;Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\
&= I(Z;X) - (\lambda_1 + 1) I(Z;Y) - \lambda_2 I(Z;Y|S), \qquad (5.13)
\end{aligned}$$

where the second equality follows from the Markov chain assumption $(Y, S) \to X \to Z$. □

The simpler Lagrangian in (5.12) is easier to compute while exactly maintaining the properties of the original one. It also reveals a direct relation of the original Lagrangian with the first two terms being exactly equivalent to the "classical IB" formulation. The two hyperparameters β_1 and β_2 control trade-offs between accuracy (or "utility") and fairness, with higher β values corresponding to a higher priority on accuracy and lower β values giving more influence to the compression term $I(Z;X)$ that discards unwanted information, potentially improving fairness at the expense of accuracy. As $I(Z;Y)$ is partially derived from the $I(Z;S|Y)$ term designed to improve equalized odds, using a higher β_1 over β_2 should give more priority to improving equalized odds, whereas a higher β_2 should result in improved demographic parity. This allows for more nuanced outcomes compared to other methods that focus rigidly on a single fairness metric. It is also possibly an interesting tool for policy makers to translate those more balanced and nuanced versions of fairness into an "engineered system."

5.2.3 Variational Bounds

We use a variational approach to develop bounds on the three terms in the Lagrangian in (5.12), finding lower bounds for the terms to be maximized and an upper bound for the term to be minimized. The Markov chain property $(Y, S) \to X \to Z$ results in the joint distribution P_{SYXZ} factoring as as $P_{SYX}P_{Z|X}$.

The distribution $P_{Z|X}$ is a parametric stochastic encoder to be designed while all other distributions are fully determined by the joint data distribution $P_{S,X,Y}$, the encoder, and the Markov chain constraint. To simplify notation, we simply write $P_{Z|X}$ rather than including the parameter $P_{Z|X,\boldsymbol{\theta}}$, with $\boldsymbol{\theta}$ denoting network weights. Computing the mutual information terms requires the usually intractable distributions $P_{Y|S,Z}$, $P_{Y|Z}$, and P_Z; we thus replace them with variational approximations $Q_{Y|S,Z}$, $Q_{Y|Z}$ and Q_Z, respectively. We next derive an upper bound for $I(Z; X)$ with the novel use of Rényi divergence:

$$\begin{aligned}
I(Z;X) &= \sum_{(\boldsymbol{z},\boldsymbol{x})\in\mathcal{Z}\times\mathcal{X}} P_{Z,X}(\boldsymbol{z},\boldsymbol{x}) \log \frac{P_{Z|X}(\boldsymbol{z}|\boldsymbol{x})}{P_Z(\boldsymbol{z})} \\
&= \sum_{(\boldsymbol{z},\boldsymbol{x})\in\mathcal{Z}\times\mathcal{X}} P_{Z,X}(\boldsymbol{z},\boldsymbol{x}) \log P_{Z|X}(\boldsymbol{z}|\boldsymbol{x}) \\
&\quad - D_{KL}(P_Z\|Q_Z) - \sum_{\boldsymbol{z}\in\mathcal{Z}} P_Z(\boldsymbol{z}) \log Q_Z(\boldsymbol{z}) \\
&\le \sum_{(\boldsymbol{z},\boldsymbol{x})\in\mathcal{Z}\times\mathcal{X}} P_{Z,X}(\boldsymbol{z},\boldsymbol{x}) \log \frac{P_{Z|X}(\boldsymbol{z}|\boldsymbol{x})}{Q_Z(\boldsymbol{z})} \\
&= \mathbb{E}_{P_X} D_{KL}\left(P_{Z|X}\|Q_Z\right) \\
&\le \mathbb{E}_{P_X} D_{\alpha}\left(P_{Z|X}\|Q_Z\right), \qquad (5.14)
\end{aligned}$$

for $\alpha > 1$. The first inequality follows from the non-negativity of Kullback-Leibler (KL) divergence, similar to [2, 88, 80]. For the final step, we take the Rényi divergence $D_\alpha(\cdot\|\cdot)$ of order α (e.g., see [92]), defined in Definition 6, rather than the KL divergence as typically done in the literature. Using Rényi divergence gives an extra degree of freedom and allows more control over the compression term $I(X;Z)$. As the Rényi divergence is non-decreasing with α, a higher α will more strongly force the distribution $P_{Z|X}$ closer to Q_Z, resulting in more compression.

The upper bound in (5.14) holds for $\alpha > 1$ since D_α is non-decreasing in α and $\lim_{\alpha\to 1} D_\alpha(P\|Q) = D_{KL}(P\|Q)$.[1] When $\alpha < 1$, then $\mathbb{E}_{P_X} D_\alpha\big(P_{Z|X}\|Q_Z\big)$ is no longer an upper bound on $I(Z;X)$; but it can be considered as a potentially useful approximation that is tunable by varying α.

We can similarly leverage the non-negativity of the KL divergence to get lower bounds on $I(Z;Y)$ and $I(Z;Y|S)$:

$$
\begin{aligned}
I(Z;Y) &= H(Y) - H(Y|Z) \\
&= H(Y) + \sum_{(y,\boldsymbol{z})\in\times\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log P_{Y|Z}(y|\boldsymbol{z}) \\
&= H(Y) + \sum_{(y,\boldsymbol{z})\in\times\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log P_{Y|Z}(y|\boldsymbol{z}) \\
&\qquad - \sum_{(y,\boldsymbol{z})\in\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log Q_{Y|Z}(y|\boldsymbol{z}) \\
&\qquad + \sum_{(y,\boldsymbol{z})\in\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log Q_{Y|Z}(y|\boldsymbol{z}) \\
&= H(Y) + \mathbb{E}_{P_Z} D(P_{Y|Z}||Q_{Y|Z}) + \sum_{(y,\boldsymbol{z})\in\mathcal{Y}\times\mathcal{Z}} P_{Y,Z}(y,\boldsymbol{z}) \log Q_{Y|Z}(y|\boldsymbol{z})
\end{aligned}
$$

[1]For simplicity and by the continuity property of D_α in α, we define its extended orders at $\alpha = 1$ and $\alpha = 0$ [92] as $D_1(P\|Q) := D_{KL}(P\|Q)$ and $D_0(P\|Q) := \lim_{\alpha\to 0} D_\alpha(P\|Q) = -\log Q(x : P(x) > 0)$, which is equal to 0 when P and Q share a common support.

$$\geq H(Y) + \mathbb{E}_{P_{Y,Z}} \left[\log Q_{Y|Z}(Y|Z)\right], \tag{5.15}$$

and

$$\begin{aligned}
I(Z;Y|S) &= H(Y|S) - H(Y|Z,S) \\
&= H(Y|S) + \sum_{(y,\boldsymbol{z},s)\in\times\mathcal{Y}\times\mathcal{Z}\times\mathcal{S}} P_{Y,Z,S}(y,\boldsymbol{z},s)\log P_{Y|Z,S}(y|\boldsymbol{z},s) \\
&= H(Y|S) + \sum_{(y,\boldsymbol{z},s)\in\times\mathcal{Y}\times\mathcal{Z}\times\mathcal{S}} P_{Y,Z,S}(y,\boldsymbol{z},s)\log P_{Y|Z,S}(y|\boldsymbol{z},s) \\
&\qquad - \sum_{(y,\boldsymbol{z},s)\in\mathcal{Y}\times\mathcal{Z}} P_{Y,Z,S}(y,\boldsymbol{z},s)\log Q_{Y|Z,S}(y|\boldsymbol{z},s) \\
&\qquad + \sum_{(y,\boldsymbol{z},s)\in\mathcal{Y}\times\mathcal{Z}\times\mathcal{S}} P_{Y,Z,S}(y,\boldsymbol{z},s)\log Q_{Y|Z,S}(y|\boldsymbol{z},s) \\
&= H(Y|S) + \mathbb{E}_{P_{Z,S}} D(P_{Y|Z,S}||Q_{Y|Z,S}) \\
&\qquad + \sum_{(y,\boldsymbol{z},s)\in\mathcal{Y}\times\mathcal{Z}\times\mathcal{S}} P_{Y,Z,S}(y,\boldsymbol{z},s)\log Q_{Y|Z,S}(y|\boldsymbol{z},s) \\
&\geq H(Y|S) + \mathbb{E}_{P_{Y,Z,S}}\left[\log Q_{Y|Z,S}(Y|Z,S)\right].
\end{aligned} \tag{5.16}$$

As the entropy $H(Y)$ and conditional entropy $H(Y|S)$ of the labels do not depend on the parameterization they can be ignored for the optimization.

5.2.4 Computing the Bounds

To compute the bounds in practice we use the reparameterization trick, as explained in Section 3.3. Modeling $P_{Z|X}$ as a density, we let $P_{Z|X}dZ = P_E dE$, where E is a random variable and $Z = f(X,E)$ is a deterministic function, allowing us to back-propagate gradients and optimize the parameter via gradient descent. We use the data's empirical densities to estimate $P_{X,S}$ and $P_{X,Y,S}$.

Considering a batch $D = \{\boldsymbol{x}^{\{i\}}, s^{\{i\}}, y^{\{i\}}\}_{i=1}^{N}$ this finally leads to the following RFIB cost function to minimize over $P_{Z|X}$:

$$\begin{aligned} J_{\text{RFIB}} = \frac{1}{N}\sum_{i=1}^{N} \Big[& D_\alpha(P_{Z|X=\boldsymbol{x}^{\{i\}}} \| Q_Z) \\ & - \beta_1 \mathbb{E}_E \left[\log\left(Q_{Y|Z}\left(y^{\{i\}} | f(\boldsymbol{x}^{\{i\}}, E)\right)\right)\right] \\ & - \beta_2 \mathbb{E}_E \left[\log\left(Q_{Y|S,Z}\left(y^{\{i\}} | s^{\{i\}}, f(\boldsymbol{x}^{\{i\}}, E)\right)\right)\right] \Big], \end{aligned} \tag{5.17}$$

where we estimate the expectation over E using a single Monte Carlo sample.

Remark 1. *We note that depending on the choice of α, β_1, and β_2, from our method we can recover both the* IB *[2] and* conditional fairness bottleneck (CFB) *[80] schemes to which we compare our results. Letting $\alpha = 1$ and $\beta_2 = 0$ corresponds to IB, while setting $\alpha = 1$ and $\beta_1 = 0$ corresponds to CFB.*

5.2.5 Derivation of Rényi Divergence for Gaussians

Here we calculate the Rényi divergence term $D_\alpha\left(P_{Z|X} \| Q_Z\right)$ of our cost function when $P_{Z|X}$ and Q_Z are multivariate Gaussian distributions with $P_{Z|X}$ having a diagonal covariance structure while Q_Z being a spherical Gaussian. More specifically, we let

$$\begin{aligned} P_{Z|X} &= \mathcal{N}\left(Z | \boldsymbol{\mu}_{\text{enc}}(X), \operatorname{diag}(\boldsymbol{\sigma}^2_{enc}(X))\right) \\ Q_Z &= \mathcal{N}(Z | \mathbf{0}, \gamma^2 \boldsymbol{I}_d), \end{aligned}$$

where $\boldsymbol{\mu}_{\text{enc}}$ and $\boldsymbol{\sigma}^2_{\text{enc}}$ are d-dimensional mean and variance vectors (that depend on X), $\operatorname{diag}(\boldsymbol{\sigma}^2_{enc}(X))$ is a $d \times d$ diagonal matrix with the entries of $\boldsymbol{\sigma}^2_{enc}$ on the diagonal, $\mathbf{0}$ is the all-zero vector of size d, γ is a positive scalar, and $\boldsymbol{I}_d$ is the d-dimensional identity matrix. For simplicity of notation, in the rest of this chapter we write $\boldsymbol{\mu}_{enc}(X)$ as $\boldsymbol{\mu}_{enc}$ and $\mathrm{diag}(\boldsymbol{\sigma}^2_{enc}(X))$ as $\boldsymbol{\sigma}^2_{enc}\boldsymbol{I}_d$.

Starting with the closed-form expression of the Rényi divergence of order α ($\alpha > 0$, $\alpha \neq 1$) derived in [28, 11], we have

$$D_\alpha\left(P_{Z|X} \| Q_Z\right) = \frac{\alpha}{2}\left(\boldsymbol{\mu}'_{enc}[(\Sigma_\alpha)^*]^{-1}\boldsymbol{\mu}_{enc}\right) - \frac{1}{2(\alpha-1)} \ln \frac{|(\Sigma_\alpha)^*|}{|\boldsymbol{\sigma}^2_{enc}\boldsymbol{I}_d|^{1-\alpha}\,|\gamma^2\boldsymbol{I}_d|^{\alpha}}$$

where $\boldsymbol{\mu}'_{enc}$ is the transpose of $\boldsymbol{\mu}_{enc}$,

$$(\Sigma_\alpha)^* = \alpha\gamma^2\boldsymbol{I}_d + (1-\alpha)\boldsymbol{\sigma}^2_{enc}\boldsymbol{I}_d,$$

and $\alpha[\boldsymbol{\sigma}^2_{enc}\boldsymbol{I}_d]^{-1} + (1-\alpha)[\gamma^2\boldsymbol{I}_d]^{-1}$ is positive definite. Then

$$\frac{\alpha}{2}\left(\boldsymbol{\mu}'_{enc}[(\Sigma_\alpha)^*]^{-1}\boldsymbol{\mu}_{enc}\right) = \frac{\alpha}{2}\sum_{i=1}^{d} \frac{\mu_i^2}{\alpha\gamma^2 + (1-\alpha)\sigma_i^2}, \tag{5.18}$$

where μ_i and σ_i are the ith components of μ_{enc} and σ_{enc} and

$$\begin{aligned}
&\frac{1}{2(\alpha-1)} \ln \frac{|(\Sigma_\alpha)^*|}{|\boldsymbol{\sigma}^2_{enc}\boldsymbol{I}_d|^{1-\alpha}\,|\gamma^2\boldsymbol{I}_d|^{\alpha}} \\
&= \frac{1}{2(\alpha-1)} \ln \frac{\begin{vmatrix} \alpha\gamma^2 + (1-\alpha)\sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \alpha\gamma^2 + (1-\alpha)\sigma_d^2 \end{vmatrix}}{\begin{vmatrix} \sigma_1^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_d^2 \end{vmatrix}^{1-\alpha} \begin{vmatrix} \gamma^2 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \gamma^2 \end{vmatrix}^{\alpha}}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2(\alpha-1)} \ln \frac{\prod_{i=1}^{d} [\alpha\gamma^2 + (1-\alpha)\sigma_i^2]}{(\prod_{i=1}^{d} \sigma_i^2)^{1-\alpha} (\prod_{i=1}^{d} \gamma^2)^{\alpha}} \\
&= \frac{1}{2(\alpha-1)} \sum_{i=1}^{d} \ln \frac{\alpha\gamma^2 + (1-\alpha)\sigma_i^2}{\sigma_i^{2(1-\alpha)} \gamma^{2\alpha}}, \qquad (5.19)
\end{aligned}
$$

yielding that

$$
D_\alpha\left(P_{Z|X} \| Q_Z\right) = \frac{\alpha}{2} \sum_{i=1}^{d} \frac{\mu_i^2}{\alpha\gamma^2 + (1-\alpha)\sigma_i^2} - \frac{1}{2(\alpha-1)} \sum_{i=1}^{d} \ln \frac{\alpha\gamma^2 + (1-\alpha)\sigma_i^2}{\sigma_i^{2(1-\alpha)} \gamma^{2\alpha}}. \quad (5.20)
$$

Since we require the matrix $\alpha[\sigma_{enc}^2 \boldsymbol{I}_d]^{-1} + (1-\alpha)[\gamma^2 \boldsymbol{I}_d]^{-1}$ to be positive definite, the above Rényi divergence expression is valid for

$$
\alpha\gamma^2 + (1-\alpha)\sigma_i^2 > 0, \qquad i = 1, \ldots, d,
$$

or equivalently (recalling that $\alpha > 0$, $\alpha \neq 1$) for

$$
\begin{aligned}
&0 < \alpha < 1,\ \sigma_i^2 > 0, \qquad i = 1, \ldots, d, \\
&\alpha > 1, \sigma_i^2 < \frac{\alpha\gamma^2}{\alpha - 1}, \qquad i = 1, \ldots, d. \qquad (5.21)
\end{aligned}
$$

We finish this section with a remark.

Remark 2 (Limit as $\alpha \to 1$). *Taking the limit of the Rényi divergence as $\alpha \to 1$ we recover, as expected, the KL divergence expression between Gaussians:*

$$
\begin{aligned}
\lim_{\alpha \to 1} D_\alpha(P_{Z|X} \| Q_Z) &= \lim_{\alpha \to 1} \left[\frac{\alpha}{2} \sum_{i=1}^{d} \frac{\mu_i^2}{\alpha\gamma^2 + (1-\alpha)\sigma_i^2} - \frac{1}{2(\alpha-1)} \sum_{i=1}^{d} \ln \frac{\alpha\gamma^2 + (1-\alpha)\sigma_i^2}{\sigma_i^{2(1-\alpha)} \gamma^{2\alpha}} \right] \\
&= \frac{1}{2} \sum_{i=1}^{d} \frac{\mu_i^2}{\gamma^2} - \lim_{\alpha \to 1} \frac{1}{2} \left[\sum_{i=1}^{d} \frac{\gamma^2 - \sigma_i^2}{\alpha\gamma^2 + (1-\alpha)\sigma_i^2} + 2 \ln \sigma_i - \ln \gamma^2 \right]
\end{aligned}
$$

$$
\begin{aligned}
&= -\frac{1}{2}\sum_{i=1}^{d}\left[\ln\sigma_i^2 - \ln\gamma^2 + 1 - \frac{\sigma_i^2}{\gamma^2} - \frac{\mu_i^2}{\gamma^2}\right] \\
&= D_{KL}(P_{Z|X}\|Q_Z). \qquad (5.22)
\end{aligned}
$$

5.3 Experiments

We conduct experiments on three different image datasets: CelebA, FairFace, and EyePACS. In this section, we detail the implementation steps of our method, describe the metrics and the datasets we used, and explain how the experiments were performed and present results. We also present a uniform manifold approximation and projection (UMAP) clustering analysis that visualizes the effects of the hyper-parameter α.

5.3.1 Additional Implementation Details

For all experiments, we use an isotropic Gaussian distribution for the encoder with mean and variance learned by a neural network, $P_{Z|X} = \mathcal{N}(Z|\boldsymbol{\mu}_{\text{enc}}, \boldsymbol{\sigma}^2_{\text{enc}}\boldsymbol{I}_d)$, using the same notation as described previously. Leveraging the reparameterization trick, we compute our representation Z as $Z = \boldsymbol{\mu}_{\text{enc}} + \boldsymbol{\sigma}_{\text{enc}} \odot E$, where $\odot$ is the Hadamard product and $E \sim \mathcal{N}(0, \boldsymbol{I}_d)$.

We model the approximation of the representation's marginal as a d-dimensional spherical Gaussian, $Q_Z = \mathcal{N}(Z|\mathbf{0}, \gamma^2\boldsymbol{I}_d)$ and calculate the Rényi divergence in (5.17) between the multivariate Gaussians $P_{Z|X}$ and Q_Z using (5.20). For all experiments, we use a value of $\gamma = 1$. For $\alpha < 1$, by (5.21) there is no restriction on the values of $\boldsymbol{\sigma}^2_{\text{enc}}$ so we let them be any positive number, while for $\alpha > 1$ we add an additional sigmoid function to the encoder to limit outputs such that all values of $\boldsymbol{\sigma}^2_{\text{enc}}$ are

< 1. Finally, as we only use binary values for Y, we model $Q_{Y|Z}$ with Bernoulli distributions, $Q_{Y|Z} = \text{Bernoulli}(Y; f(Z))$ and $Q_{Y|Z,S} = \text{Bernoulli}(Y; g(Z, S))$ where f and g are auxillary fully connected networks.

The encoder network $P_{Z|X}$ is a ResNet50 (Section 2.9) classifier pretrained on ImageNet with the final linear layer replaced by two randomly initialized layers with output dimension d equal to the dimension of the representation. The two decoder networks f and g are each two fully connected layers with 100 units and a sigmoid layer.

After creating the representation Z, we use a logistic regression classifier with default settings from `sci-kit learn` [73] to predict Y from Z. We evaluate accuracy and fairness on these predictions. Fig. 5.1 shows the architecture of our model.

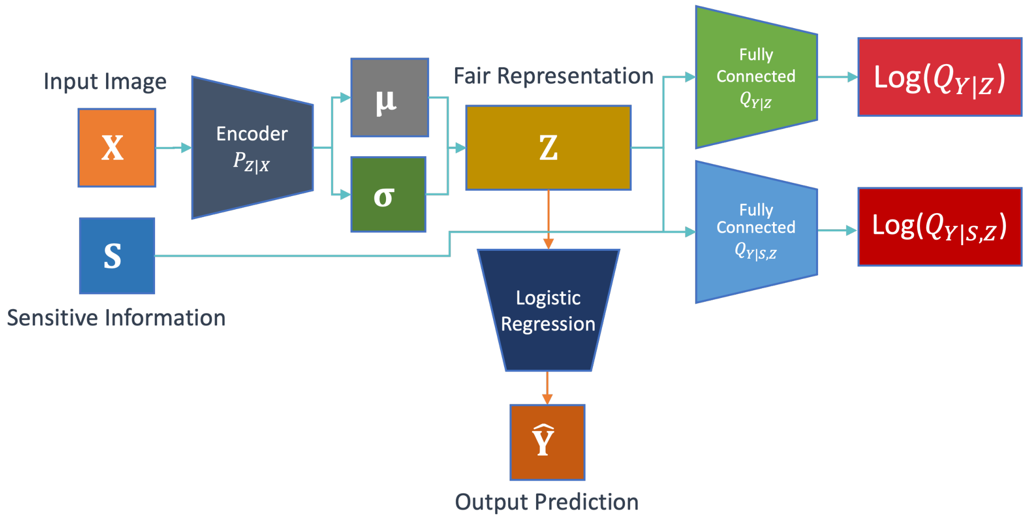


Figure 5.1: Architecture of the model. The input image X is given to the encoder $P_{Z|X}$ that generates the mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ for the distribution of the fair representation Z, $\mathcal{N}(Z|\boldsymbol{\mu}_{\text{enc}}, \boldsymbol{\sigma}^2_{\text{enc}}\boldsymbol{I}_d)$. Z is then given to two fully connected networks, one of which is also given the sensitive information S. After training, Z can be used as input to other existing architectures for fair prediction. For our experiments, we use logistic regression to predict Y from Z.

We preprocess images by taking a 128 by 128 pixel centre crop of the 218 by 178 pixel CelebA images, and a 256 pixel by 256 pixel centre crop of EyePACS images. For FairFace we use the full 224 pixel by 224 pixel images. We then split the training data into a training set and validation set, using 20% of the data for validation, and use the validation set for early stopping. We train for up to 20 epochs, with early stopping triggering when there is no decrease in validation loss for 5 epochs, using a `min_delta` value of 0.

For all experiments, we train using PyTorch on a NVIDIA GP100 GPU. We use $d = 32$ as the dimension of our representation Z, a batch size of 64, and the Adam optimizer with a learning rate of 0.001. For most experiments, we tune hyperparameters by varying α linearly from 0 to 1, with $\alpha = 0$ signifying

$$D_0(P_{Z|X} \| Q_Z) = -\log Q_Z(z : P_{Z|X}(z) > 0) = 0$$

and with $\alpha = 1$ corresponding to $D_\alpha(P_{Z|X} \| Q_Z)$ being given by the KL divergence $D_{KL}(P_{Z|X} \| Q_Z)$ (see Footnote 1), and by varying β_1 and β_2 linearly from 1 to 50. We also perform some additional experiments with α values > 1.

5.3.2 Metrics

We use the following metrics to evaluate how well the model performs:

Measure of Utility

We use the overall classification accuracy (later denoted *acc*).

Measures of Fairness

We measure this in multiple ways:

1. Using the gap in accuracy (denoted acc_{gap}) between favoured and protected subpopulations (defined in Definition 33);

2. Reporting the minimum accuracy across subpopulations (denoted as acc_{min}), which is based on the Rawlsian principle of achieving fairness by maximizing acc_{min} [78];

3. Measuring the adherence to demographic parity via its gap dp_{gap}, following [80];

4. Measuring the adherence using equalized odds via its gap $eqodds_{\text{gap}}$.

The latter two metrics are already defined in Definition 30 and Definition 31, respectively.

Joint Utility-Fairness Measure

Echoing and comparing with the work in [72], we use a single metric that jointly captures utility and fairness, the Conjunctive Accuracy Improvement (CAI_λ) measure:

$$\text{CAI}_\lambda = \lambda(acc^b_{\text{gap}} - acc^d_{\text{gap}}) + (1-\lambda)(acc^d - acc^b) \tag{5.23}$$

where $0 \leq \lambda \leq 1$, and acc^b and acc^d are the accuracy for baseline and debiased algorithms, respectively, while acc^b_{gap} and acc^d_{gap} are gap in accuracy for the baseline and debiased algorithms. In practice, one can use $\lambda = 0.5$ for an equal balance between utility and fairness or a higher value such as $\lambda = 0.75$ to emphasize fairness.

5.3.3 Data

The datasets we use (sample images shown in Fig. 5.2) include the following.

CelebA

The CelebA dataset [61] contains 202,599 celebrity faces that each have 40 binary attributes. We use age as our prediction target Y and are interested in skin tone as the sensitive attribute S. As this is not included in the dataset we instead use the Individual Topology Angle (ITA) [95] as a proxy, which was found to correlate with the Melanin Index, frequently used in dermatology to classify human skin on the Fitzpatrick scale. As in [68, 72], we compute ITA via

$$\text{ITA} = \frac{180}{\pi} \arctan\left(\frac{L-50}{b}\right), \tag{5.24}$$

where L is luminescence and b is yellowness in CIE-Lab space. We then binarize ITA where an ITA of ≤ 28 is taken to mean dark skin, matching category thresholds used in [55, 72].

FairFace

The FairFace dataset [49] consists of 108,501 face images labeled with race, gender, and age. We consider age as the target Y and race as sensitive information S. The dataset contains several different categories for race and age and we binarize them into Black and non-Black for race and ≥ 30 years old and < 30 years old for age.

EyePACS

The EyePACS dataset [23] is sourced from the Kaggle Diabetic Retinopathy challenge. It consists of 88,692 retinal fundus images of individuals potentially suffering from diabetic retinopathy (DR), an eye disease associated with diabetes that is one of the leading causes of visual impairment worldwide. The dataset contains 5 categories of images based on the severity of the disease, with 0 being completely healthy and 4 being the most severe form of the disease. Similar to [72], we binarize this label into our prediction target Y, with $Y = 1$ corresponding to categories 2-4, considered a positive, referable case for DR and $Y = 0$ corresponding to categories 0-1, considered healthy.

In our experiments we consider skin tone as the sensitive attribute. As with the CelebA dataset, we use ITA as a proxy for skin tone, with S a binary variable that denotes whether or not the ITA of the fundus is ≤ 19, denoting that the individual has dark skin, as done by [72]. This has the advantage of being significantly easier to determine compared to the expensive and time consuming process of having a clinician manually annotate images. We compare these results with additional experiments we run using a smaller partition of the dataset with the race labels manually added by a ophthalmologist as done in [12], determined based on factors that might correlate with race such as the darkness of pigmentation in the fundus, thickness of blood vessels, and ratio of the optic cup size to optic disk size.

5.3.4 Experiment Details and Results

We perform experiments on the EyePACS, CelebA, and FairFace datasets, considering the challenging case of severe data imbalance where training data is completely

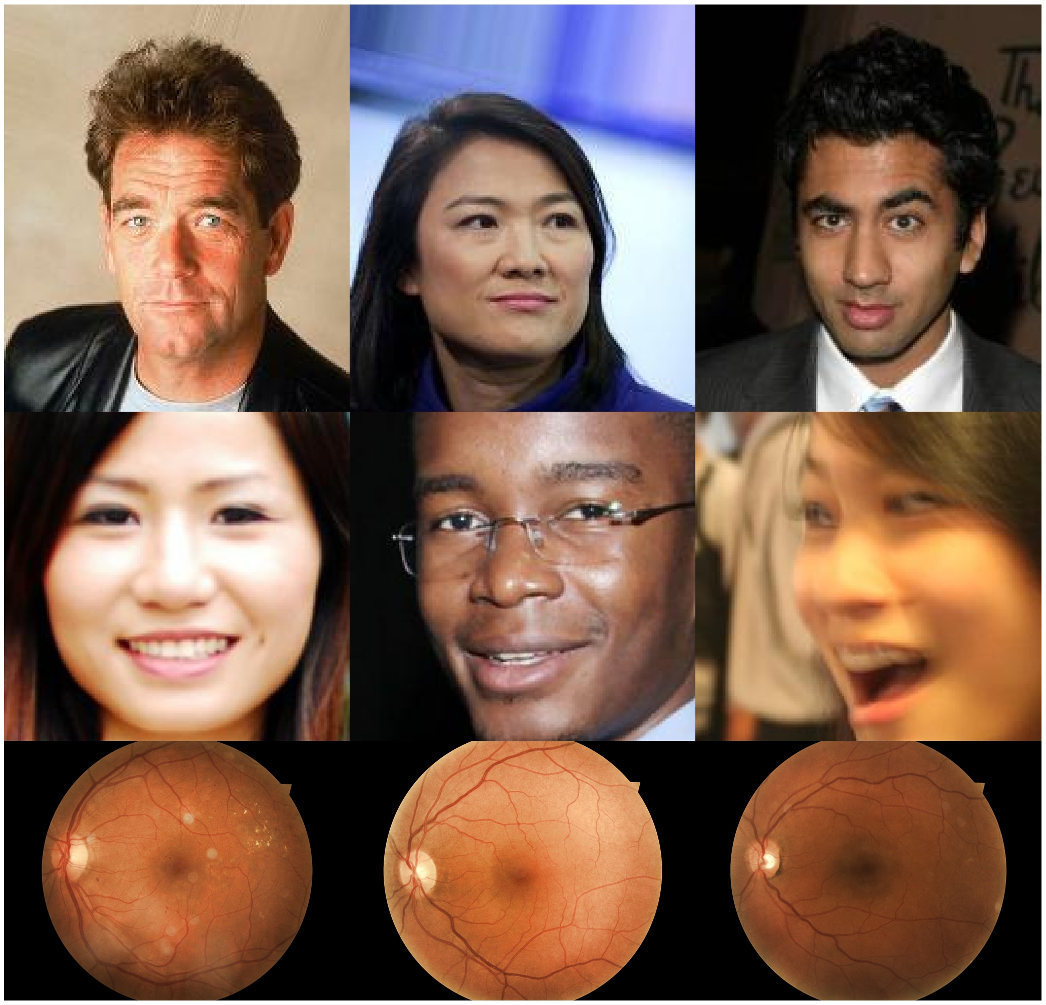


Figure 5.2: Examples of images taken respectively from CelebA (top row), FairFace (middle row), and EyePACS (bottom row) datasets. For the EyePACS images, for the left image $(Y, S) = (1, 0)$, for the middle $(Y, S) = (0, 0)$, and for the right $(Y, S) = (0, 1)$.

missing for one protected subgroup (e.g., diseased dark skin individuals for EyePACS).

Hyper-parameter Tuning

For all datasets, we conduct a hyper-parameter sweep, using various combinations of hyperparameters β_1 and β_2 varied linearly from 1 to 50 and α varied linearly from 0 to

1, where $\alpha = 1$ signifies KL divergence instead of Rényi divergence. While we mostly focus on values of $\alpha < 1$, as these correspond to an approximation of a tighter upper bound that we expect would lead to better results, we also run several experiments with values of $\alpha > 1$.

As our RFIB method subsumes IB and CFB, with each of their hyper-parameters corresponding to one of RFIB's hyper-parameters, we compare RFIB with those two methods. Setting $\alpha = 1$ and $\beta_1 = 0$ or $\beta_2 = 0$ results in RFIB being equivalent to CFB or IB, respectively, as the latter two systems are based on KL divergence and only have a single β hyper-parameter. We do separate comparisons of CFB, IB with RFIB, picking a commonly used value that performs well for CFB, IB and setting RFIB's corresponding hyperparmeter to the same value. We then use grid search to tune the two additional hyper-parameters that our method introduces; to compare with IB we fix a value of β_1 and tune α and β_2, while to compare with CFB we fix a value of β_2 and then tune α and β_1.

We implement the IB and CFB methods ourselves and also compare RFIB with two methods used by [72]: adversarial independence, referred to as adversarial debiasing (AD), that minimizes conditional dependence of predictions on sensitive attributes with an adversarial two player game and intelligent augmentation (IA) that generates synthetic data for underrepresented populations and performs data augmentation to train a less biased model. To compare with AD and IA, we take results from [72] and report their original CAI scores calculated with respect to their baseline, while for IB and CFB we implement the methods ourselves and calculate CAI scores with respect to results from our own baseline, a ResNet50 network [42].

EyePACS Results

We predict Y = DR Status while using S = ITA as the sensitive attribute (note that throughout this thesis, we use this abuse of notation to denote which attribute each variable represents). We consider a scenario where training data is completely missing for the subgroup of $(Y, S) = (1, 1)$, individuals referable for DR who have dark skin. The goal of our method is for predictions on the missing subgroup to be just as accurate as on the group with adequate training data, which is a problem of both fairness and also *domain adaptation*, achieving high performance on a group not present in the original dataset. This scenario matches an important real world problem where data for a minority subgroup such as dark skinned individuals is lacking.

We create a training partition containing both images referable and non-referable for DR of light skin individuals but only non-referable images of dark skin individuals. We use the same partition as in [72] to compare with their method, using a training set that consists of 10,346 light skin images referable for diabetic retinopathy (DR = 1, ITA = 0), 5,173 non-referable light skin images (DR = 0, ITA = 0), and 5,173 non-referable dark skin images (DR = 0, ITA = 1). Then for a fair assessment of our method's performance we evaluate on a balanced test set with an equal number of positive and negative examples for both dark and light skin individuals, with the set containing 2,400 images equally balanced across DR and ITA.

As shown in Table 5.1, our method outperforms all other methods, showing improvements in accuracy and fairness across all metrics. We show a result with a value of $\alpha > 1$ in the bottom row, showing that it is still possible to achieve promising performance with higher α values, although best results for most metrics were achieved with values of $\alpha < 1$, as expected since these values correspond to an approximation of a tighter upper bound on the $I(Z;X)$ mutual information term (that is being minimized). Usual caution should be exercised in interpretations since – despite our aligning with data partitioning in [72] – other variations may exist with [72, 2, 80] due to non-determinism, parameter setting or other factors.

Table 5.1: Results for debiasing methods on EyePACS predicting $Y=$ DR Status, trained on partitioning with respect to $S=$ ITA, and evaluated on a test set balanced across DR status and ITA. For metrics with an $\uparrow$ higher is better whereas for $\downarrow$ lower is better. Subpopulation is the one that corresponds to the minimum accuracy, with (D) indicating dark skin and (L) light skin. Metrics are given as percentages.

Methods	$acc \uparrow$	$acc_{\text{gap}} \downarrow$	$acc_{\text{min}} \uparrow$ (subpop.)	$\text{CAI}_{0.5} \uparrow$	$\text{CAI}_{0.75} \uparrow$	$dp_{\text{gap}} \downarrow$	$eqodds_{\text{gap}} \downarrow$
Baseline (from [72])	70.0	3.5	68.3	-	-	NA	NA
AD ($\beta = 0.5$) ([72])	76.1	2.4	74.9 (L)	3.6	2.3	NA	NA
IA ([72])	71.5	1.5	70.7 (D)	1.8	1.9	NA	NA
Baseline (ours)	73.4	8.1	69.3 (D)	-	-	28.3	36.3
IB (β_1=30) ([2])	74.1	2.1	73.1 (D)	3.4	4.7	18.6	20.7
CFB (β_2=30) ([80])	77.8	1.7	77.0 (L)	5.4	5.9	10.8	12.5
RFIB (ours) ($\alpha = 0.8, \beta_1 = 36, \beta_2 = 30$)	79.4	0.5	**79.2 (L)**	**6.8**	**7.2**	16.2	16.7
RFIB (ours) ($\alpha = 0.3$, $\beta_1 = 30$, $\beta_2 = 50$)	**79.7**	1.8	78.8 (L)	6.3	6.3	**9.8**	**11.5**
RFIB (ours) ($\alpha = 1.8, \beta_1 = 30, \beta_2 = 17$)	78.4	**0.2**	78.3 (L)	6.4	7.1	15.6	15.8

We perform a second experiment where we use the same networks from before trained using $S=$ ITA but test on a balanced test set where $S=$ Ethnicity (defined as in [72]), with labels coming from a human clinician. The test set consists of 400 images equally balanced across DR and ethnicity. We show these experimental results in Table 5.2, where we outperform the other methods across most metrics, including the most important CAI scores. As we achieve similar results using ITA and race labels from a human clinician, we show that ITA can successfully be used as an easily obtained alternative to manual label annotation, further supporting conclusions reached by [72]. These results also demonstrate the ability of our method to perform well in this type of protected factor domain adaptation problem where a different protected factor is used after initial training, which is important in settings where the actual protected factor is not revealed for privacy reasons.

Table 5.2: Performance results for debiasing methods on EyePACS predicting $Y=$ DR Status, trained on partitioning with respect to $S=$ ITA, and evaluated on a test set balanced across DR status and ethnicity (defined as in [35]). For metrics with an $\uparrow$ higher is better whereas for $\downarrow$ lower is better. Subpopulation is the one that corresponds to the minimum accuracy, with (W) indicating white individuals and (B) Black individuals. Metrics are given as percentages.

Methods	$acc \uparrow$	$acc_{\text{gap}} \downarrow$	$acc_{\min} \uparrow$ (subpop.)	$\text{CAI}_{0.5} \uparrow$	$\text{CAI}_{0.75} \uparrow$	$dp_{\text{gap}} \downarrow$	$eqodds_{\text{gap}} \downarrow$
Baseline	76.0	13.0	69.5 (B)	-	-	3.0	16.0
IB (β_2=30) ([2])	78.8	**3.5**	77.0 (B)	6.1	**7.8**	**0.5**	**4.0**
CFB (β_2=30) ([80])	75.0	6.0	72.0 (B)	3.0	5.0	5.0	11.0
RFIB (ours) ($\alpha=0.8, \beta_1=36, \beta_2=30$)	81.5	6.0	78.5 (B)	6.3	6.6	3.0	9.0
RFIB (ours) ($\alpha=0.3, \beta_1=30, \beta_2=50$)	**81.8**	4.5	**79.5 (B)**	**7.1**	**7.8**	11.5	16.0

CelebA Results

We predict age using ITA as the sensitive attribute. Again, we consider the domain adaptation/fairness problem where part of the data is completely missing for a protected subgroup. In this case, older light skin images are missing and we use a training set of 48,000 images, consisting of 24,000 older dark skin images, 12,000 younger light skin images, and 12,000 older dark skin images. We use a test set of 8,000 images equally balanced across age and ITA, using the same partition as [72] and varying hyperparameters and comparing methods the same way as for EyePACS. Our results are shown in Table 5.3, showing our method performs the best across all metrics.

Table 5.3: Results for debiasing methods on CelebA predicting Y= Age, trained on partitioning with respect to S = ITA, and evaluated on a test set balanced across age and ITA. For metrics with an ↑ higher is better whereas for ↓ lower is better. Subpopulation is the one that corresponds to the minimum accuracy, with (D) indicating dark skin and (L) light skin. Metrics are given as percentages.

Methods	$acc \uparrow$	$acc_{\text{gap}} \downarrow$	$acc_{\text{min}} \uparrow$ (subpop.)	$\text{CAI}_{0.5} \uparrow$	$\text{CAI}_{0.75} \uparrow$	$dp_{\text{gap}} \downarrow$	$eqodds_{\text{gap}} \downarrow$
Baseline (from [72])	74.4	13.9	67.5	-	-	NA	NA
AD ($\beta = 0.5$) ([72])	76.4	9.6	71.6 (D)	3.2	3.7	NA	NA
IA ([72])	75.3	9.2	70.7 (D)	2.8	1.6	NA	NA
Baseline (ours)	70.6	16.6	62.3 (D)	-	-	43.8	60.4
IB (β_1=30) ([2])	71.8	14.9	64.3 (D)	1.4	4.7	36.6	51.5
CFB (β_2=30) ([80])	71.8	13.3	65.1 (D)	2.2	2.8	38.3	51.6
RFIB (ours) ($\alpha = 0.3, \beta_1 = 30$, $\beta_2 = 43$)	75.0	10.9	69.6 (D)	5.0	5.3	29.0	40.0
RFIB (ours) ($\alpha = 0.4, \beta_1 = 1$, $\beta_2 = 30$)	**76.9**	**5.1**	**74.4 (D)**	**8.9**	**10.2**	**2.7**	**7.8**

FairFace Results

We predict Y = Gender with S = Race as the sensitive attribute, testing on a test set balanced across gender and race. As before, we exclude one population subgroup and remove Black females from the training data, matching the common real-world scenario where data is lacking for this group. Unlike the previous experiments, here we can obtain race directly from the dataset rather than using ITA as a proxy for skin tone. We binarize the race labels into two groups, a smaller Black group and a larger non-Black group. We use a training set of 16,500 images, consisting of 5,500 male Black images, 5,500 male white images, and 5,500 female white images. We test on a test set of 3,000 images equally balanced across gender and race. Our results, given in Table 5.4, show that we outperform both the IB and CFB methods both on accuracy and on all fairness metrics.

Table 5.4: Results for debiasing methods on FairFace predicting Y= Gender, trained on partitioning with respect to S = Race, and evaluated on a test set balanced across Gender and ITA. For metrics with an ↑ higher is better whereas for ↓ lower is better. Subpopulation is the one that corresponds to the minimum accuracy, with (B) indicating Black. Metrics are given as percentages.

Methods	$acc \uparrow$	$acc_{\text{gap}} \downarrow$	$acc_{\text{min}} \uparrow$ (subpop.)	$\text{CAI}_{0.5} \uparrow$	$\text{CAI}_{0.75} \uparrow$	$dp_{\text{gap}} \downarrow$	$eqodds_{\text{gap}} \downarrow$
Baseline	74.0	16.5	65.7 (B)	-	-	27.5	44.0
IB (β_2=30) ([2])	73.9	14.9	68.6 (B)	0.7	1.1	30.0	44.9
CFB (β_2=30) ([80])	75.6	14.0	65.2 (B)	2.0	2.3	26.3	40.3
RFIB (ours) ($\alpha = 0.2, \beta_1 = 30$, $\beta_2 = 29$)	**83.6**	8.5	**79.3 (B)**	**8.8**	8.4	19.5	28.0
RFIB (ours) ($\alpha = 0.2, \beta_1 = 1$, $\beta_2 = 30$)	82.1	**7.3**	78.4 (B)	8.6	**8.9**	**19.2**	**26.4**

5.3.5 UMAP Clustering Analysis and Influence of Rényi Parameter

We use UMAP [67] to provide a visual illustration of the effect of the Rényi parameter α. In Fig. 5.3, we show 2-dimensional UMAP vectors of our representation Z, colouring the points both based on the label Y and sensitive attribute S. Here experiments were done on the EyePACS dataset with Y = DR and S = ITA. The goal is for the representation to preserve information about Y, allowing the two classes of Y to be easily separated, while removing the sensitive information S and preventing its two classes from being distinguished.

A value of $\alpha = 0$ corresponds to no compression with $I(Z;X) = 0$, which preserves maximum accuracy but does not help fairness, with the classes of both Y and S being easily separated as shown in Fig. 5.3. Increasing α gradually results in the different points getting more mixed together. While this is seen for both Y and S, the $I(Z;Y)$ and $I(Z;Y|S)$ terms help ensure that information about Y is still preserved, allowing the classes of Y to still separate fairly well even with higher compression, whereas the classes of S get mixed together as desired.

We note that an intermediate value of α can potentially provide the best compromise between fairness and accuracy, as a more moderate amount of compression can be enough to sufficiently remove sensitive information and further compression might only harm accuracy. This is illustrated in Fig. 5.3 where a value of $\alpha = 0.5$ was sufficient to mix together the classes of S while still preserving an obvious separation of the two classes of Y. Further increasing α worsened the separation of Y more than it added additional benefit for S. This is also supported by our experimental results where best overall accuracy-fairness trade-offs were typically obtained for intermediate values of α.

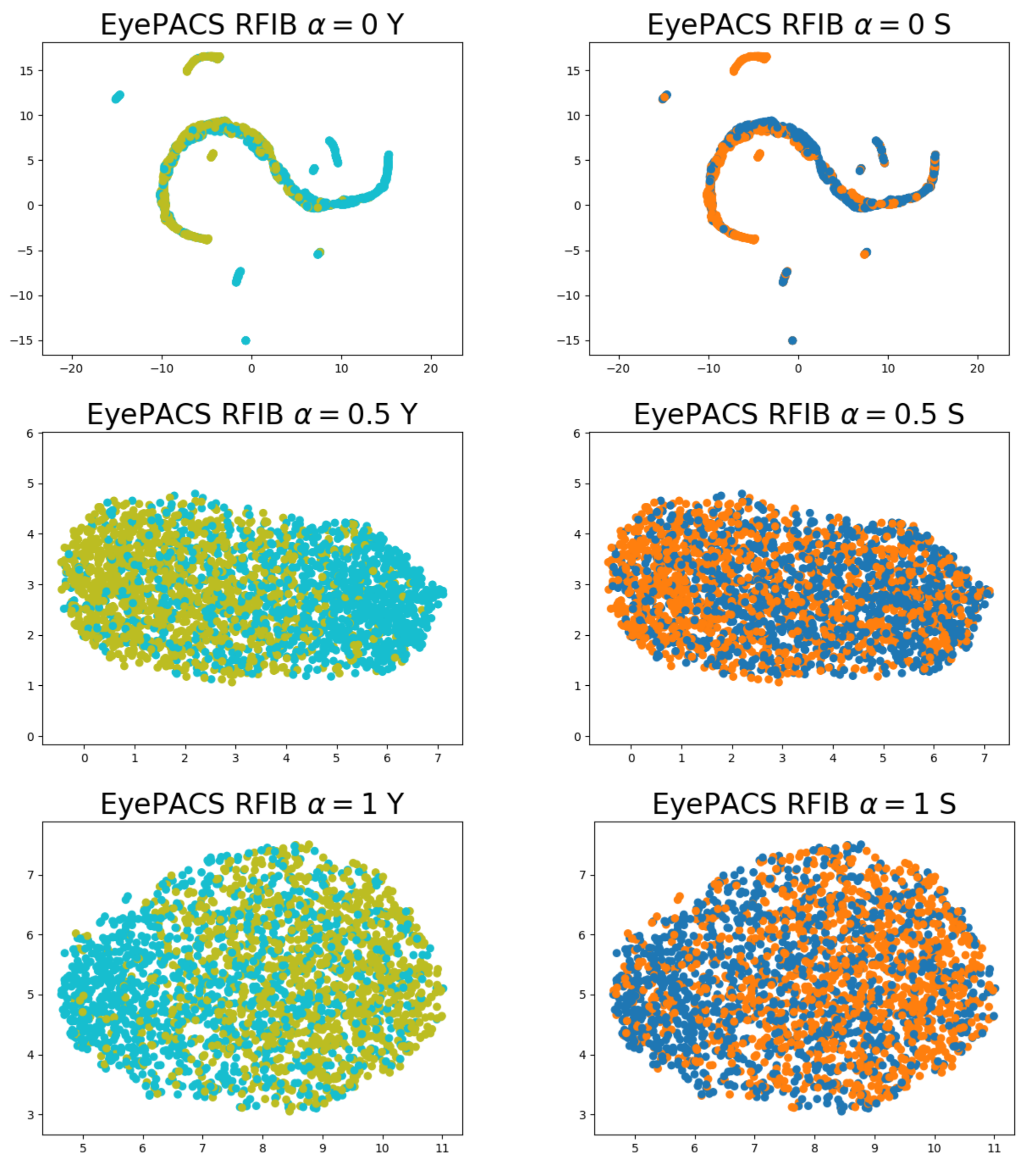


Figure 5.3: UMAP dimensionality reduction of Z for the EyePACS dataset. The left column shows the two classes of Y (light blue representing the positive examples and light green the negative) while the right column shows the two classes of S (orange representing the positive examples and blue the negative). Going from top to bottom α is increased, resulting in more compression and causing a decrease in separation of the two classes. While separation occurs for both Y and S, it occurs to a greater extent for S as desired.

Chapter 6

Fairness and Privacy Combined

In this section, we discuss the problem of privacy, how it relates to fairness, and present our combined method to improve fairness and privacy, termed Rényi Fair and Private Information Bottleneck (RFPIB).

The problem of privacy is to encode data in such a way that a private attribute $P \in \mathcal{P}$ cannot be inferred from it. Here, P can include factors such as race, gender, or age, similarly to the sensitive variable S. While the problem of fairness involves predicting Y in a way uninfluenced by S, the problem of privacy does not involve the target prediction Y.

6.1 Method

We present the RFPIB method to jointly improve fairness and privacy while maintaining utility. Our goal is to encode the input data X into a representation Z that is both *fair* and *private*. We assume that the Markov chains $P \rightarrow X \rightarrow Z$ and $(Y, S) \rightarrow X \rightarrow Z$ both hold. To simplify notation, we assume that all random variables are discrete, though a similar derivation holds for a mix of continuous and discrete random variables. We take Y, P, and S to all be binary.

6.1.1 Lagrangian Formulation

For the representation to be private, we want to minimize the amount of information Z contains about P, namely $I(Z;P)$. However, we also must retain information about X in Z to ensure utility. We want to only retain the information that is not shared by P, so we naturally condition on P and maximize $I(Z;X|P)$.

For fairness, we take a similar approach as for the RFIB method examined in Chapter 5. We want to improve both equalized odds and demographic parity, defined in Definition 24 and Definition 26, respectively, while also retaining information about Y to ensure utility. We minimize $I(Z;S|Y)$ to increase equalized odds by encouraging conditional independence of Z and S given Y, and also minimize $I(Z;X|S,Y)$ to encourage the representation to be compact. We maximize $I(Z;Y)$ to ensure the representation is maximally informative about Y, and also $I(Z;Y|S)$ which retains information about Y but only if it is not shared by S, helping to encourage demographic parity.

We combine these terms into one Lagrangian, adding non-negative Lagrange multipliers to the terms to be maximized. This results in the following Lagrangian $\mathcal{L}$ to be minimized over the encoding conditional distribution $P_{Z|X}$:

$$\mathcal{L} = I(Z;P) - \lambda_3 I(Z;X|P) + I(Z;S|Y) + I(Z;X|S,Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S). \tag{6.1}$$

Next, we rewrite this Lagrangian into a simpler expression that has fewer terms, making it easier to work with, while still maintaining similar properties to the original one.

Theorem 7. *Minimizing the Lagrangian in (6.1) over $P_{Z|X}$ is equivalent to minimizing the following expression*

$$\mathcal{L} = 2I(Z;X) - \beta_1 I(Z;Y) - \beta_2 I(Z;Y|S) - \beta_3 I(Z;X|P), \tag{6.2}$$

where $\beta_1 = \lambda_1 + 1$, $\beta_2 = \lambda_2$, and $\beta_3 = \lambda_3 + 1$.

Proof. First, we give an equivalent expression for $I(Z;P) - \lambda_3 I(Z;X|P)$:

$$\begin{aligned} I(Z;P) - \lambda_3 I(Z;X|P) &= H(Z) - H(Z|P) - \lambda_3 I(Z;X|P) \\ &= H(Z) - H(Z|X) - H(Z|P) + H(Z|X) - \lambda_3 I(Z;X|P) \\ &= I(Z;X) - [H(Z|P) - H(Z|X,P)] - \lambda_3 I(Z;X|P) \\ &= I(Z;X) - I(Z;X|P) - \lambda_3 I(Z;X|P) \\ &= I(Z;X) - (\lambda_3 + 1) I(Z;X|P), \end{aligned} \tag{6.3}$$

where the third equality follows from the Markov chain assumption $P \to X \to Z$. Next, we give an equivalent expression for $I(Z;S|Y) + I(Z;X|S,Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S)$:

$$\begin{aligned} &I(Z;S|Y) + I(Z;X|S,Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\ &= H(Z|Y) - H(Z|S,Y) + H(Z|S,Y) - H(Z|X,S,Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\ &= H(Z|Y) - H(Z|X) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\ &= H(X) - H(Z,X) - [H(Y) - H(Z,Y)] - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\ &= H(Z) + H(X) - H(Z,X) - [H(Z) + H(Y) - H(Z,Y)] - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\ &= I(Z;X) - I(Z;Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \end{aligned}$$

$$= I(Z;X) - (\lambda_1 + 1)I(Z;Y) - \lambda_2 I(Z;Y|S), \tag{6.4}$$

where the second equality follows from the Markov chain assumption $(Y,S) \to X \to Z$. Combining (6.3) and (6.4) we get

$$\begin{aligned} &I(Z;P) - \lambda_3 I(Z;X|P) + I(Z;S|Y) + I(Z;X|S,Y) - \lambda_1 I(Z;Y) - \lambda_2 I(Z;Y|S) \\ &= 2I(Z;X) - (\lambda_3 + 1)I(Z;X|P) - (\lambda_1 + 1)\, I(Z;Y) - \lambda_2 I(Z;Y|S), \end{aligned} \tag{6.5}$$

and thus letting $\beta_1 = \lambda_1 + 1$, $\beta_2 = \lambda_2$, and $\beta_3 = \lambda_3 + 1$ yields (6.2). $\square$

6.1.2 Variational Bounds

We give variational bounds for each of the four terms in the Lagrangian. For the $I(Z;X)$ compression term to be minimized, we use the upper bound derived in Section 5.2.3:

$$I(Z;X) \leq \mathbb{E}_{P_X} D_\alpha \left(P_{Z|X} \| Q_Z \right). \tag{6.6}$$

The distribution $P_{Z|X}$ is modelled by a neural network (we drop the parameter $\boldsymbol{\theta}$ from $P_{Z|X;\boldsymbol{\theta}}$ to simplify notation, with $\boldsymbol{\theta}$ denoting the network's weights and biases). This network is called the *encoder network* as it encodes X into the fair/private representation Z. The distribution Q_Z is a variational approximation to the true distribution of Z, P_Z (in our experiments, we choose Q_Z to be a spherical Gaussian). The divergence is Renyi's divergence with parameter α, as defined in Definition 6 and discussed in Section 5.2.3.

For the $I(Z;Y)$ and $I(Z;Y|S)$ terms to be maximized, we also use lower bounds derived in Section 5.2.3:

$$I(Z;Y) \geq H(Y) + \mathbb{E}_{P_{Y,Z}}\left[\log Q_{Y|Z}(Y|Z)\right], \tag{6.7}$$

$$I(Z;Y|S) \geq H(Y|S) + \mathbb{E}_{P_{S,Y,Z}}\big[\log Q_{Y|S,Z}(Y|S,Z)\big]. \tag{6.8}$$

The variational approximations $Q_{Y|Z}$ and $Q_{Y|Z}$ are each modelled by a neural network (again we do not include the parameters to simplify notation). As the entropy terms $H(Y)$ and $H(Y|S)$ are independent of the parameterization, they can be ignored for the optimization.

Finally, we derive a lower bound on the $I(Z;X|P)$ term to be maximized, where we replace $P_{X|Z,P}$ with a variational approximation $Q_{X|Z,P}$:

$$\begin{aligned}
I(Z;X|P) &= H(X|P) - H(X|Z,P) \\
&= H(X|P) + \sum_{(\boldsymbol{x},\boldsymbol{z},p)\in\times\mathcal{X}\times\mathcal{Z}\times\mathcal{P}} P_{X,Z,P}(\boldsymbol{x},\boldsymbol{z},p)\log P_{X|Z,P}(\boldsymbol{x}|\boldsymbol{z},p) \\
&= H(X|P) + \sum_{(\boldsymbol{x},\boldsymbol{z},p)\in\times\mathcal{X}\times\mathcal{Z}\times\mathcal{P}} P_{X,Z,P}(\boldsymbol{x},\boldsymbol{z},p)\log P_{X|Z,P}(\boldsymbol{x}|\boldsymbol{z},p) \\
&\qquad - \sum_{(\boldsymbol{x},\boldsymbol{z},p)\in\mathcal{X}\times\mathcal{Z}} P_{X,Z,P}(\boldsymbol{x},\boldsymbol{z},p)\log Q_{X|Z,P}(\boldsymbol{x}|\boldsymbol{z},p) \\
&\qquad + \sum_{(\boldsymbol{x},\boldsymbol{z},p)\in\mathcal{X}\times\mathcal{Z}\times\mathcal{P}} P_{X,Z,P}(\boldsymbol{x},\boldsymbol{z},p)\log Q_{X|Z,P}(\boldsymbol{x}|\boldsymbol{z},p) \\
&= H(X|P) + \mathbb{E}_{P_{Z,P}} D(P_{X|Z,P}||Q_{X|Z,P}) \\
&\qquad + \sum_{(\boldsymbol{x},\boldsymbol{z},p)\in\mathcal{X}\times\mathcal{Z}\times\mathcal{P}} P_{X,Z,P}(\boldsymbol{x},\boldsymbol{z},p)\log Q_{X|Z,P}(\boldsymbol{x}|\boldsymbol{z},p) \\
&\geq H(X|P) + \mathbb{E}_{P_{X,Z,P}}\left[\log Q_{X|Z,P}(X|Z,P)\right].
\end{aligned} \tag{6.9}$$

The inequality follows from the non-negativity of KL divergence. The distribution

$Q_{X|Z,P}$ again is modelled by a neural network. We call this network the *decoder network* or *reconstruction network* since it tries to decode Z and use it to reconstruct the original image X. The entropy term $H(X|P)$ is independent of the parameterization so it can be ignored for the optimization.

6.1.3 Computing the Bounds

Similar to the development in Chapter 5, to compute the bound we use the reparameterization trick (Section 3.3). Modeling $P_{Z|X}$ as a density, we let $P_{Z|X}dZ = P_E dE$, where E is a random variable and $Z = f(X, E)$ is a deterministic function, allowing us to backpropagate gradients and optimize the parameters via stochastic gradient descent. We use the data's empirical densities to estimate $P_{X,S}$, $P_{X,Y,S}$, and $P_{X,P}$.

Considering a batch $D = \{\boldsymbol{x}^{\{i\}}, s^{\{i\}}, p^{\{i\}}, y^{\{i\}}\}_{i=1}^{N}$, we minimize the following cost:

$$\begin{aligned} J = \frac{1}{N}\sum_{i=1}^{N}\big[&D_\alpha\left(P_{Z|X=\boldsymbol{x}^{\{i\}}}\|Q_Z\right) - \beta_1\mathbb{E}_E\left[\log\left(Q_{Y|Z}\left(y^{\{i\}} \mid f\left(\boldsymbol{x}^{\{i\}}, E\right)\right)\right)\right] \\ &- \beta_2\mathbb{E}_E\left[\log\left(Q_{Y|S,Z}\left(y^{\{i\}} \mid s^{\{i\}}, f\left(\boldsymbol{x}^{\{i\}}, E\right)\right)\right)\right] \\ &-\beta_3\mathbb{E}_E\left[\log\left(Q_{X|Z,P}\left(\boldsymbol{x}^{\{i\}} \mid p^{\{i\}}, f\left(\boldsymbol{x}^{\{i\}}, E\right)\right)\right)\right]\big]. \end{aligned} \tag{6.10}$$

Remark 3. *Depending on the choice of α, β_1, β_2, and β_3, from this method we can recover the* Information Bottleneck Method *[2] ($\alpha = 1, \beta_2 = 0, \beta_3 = 0$),* Conditional Fairness Bottleneck *[80] ($\alpha = 1, \beta_1 = 0, \beta_3 = 0$), or the* Conditional Privacy Funnel *[80] ($\alpha = 1, \beta_1 = 0, \beta_2 = 0$).*

6.2 Experiment Setup

We explain how we implement the RFPIB system in practice to run experiments and describe the metrics and datasets we use.

6.2.1 Implementation Details

For all experiments, we take the encoder $P_{Z|X}$ to be a multivariate Gaussian distribution with a diagonal covariance matrix where the mean and variance are determined by a neural network:

$$P_{Z|X} = \mathcal{N}\left(Z|\boldsymbol{\mu}_{enc}(X), \text{diag}(\boldsymbol{\sigma}^2_{enc}(X))\right). \tag{6.11}$$

Here, $\boldsymbol{\mu}_{enc}(X)$ and $\boldsymbol{\sigma}^2_{enc}(X)$ are d-dimensional mean and variance vectors, which are outputs of the encoder neural network, and $\text{diag}(\boldsymbol{\sigma}^2_{\text{enc}}(X))$ is a $d \times d$ diagonal matrix with the entries of $\boldsymbol{\sigma}^2_{enc}(X)$ on the diagonal. This network is a ResNet50 classifer (Section 2.9) pretrained on ImageNet with the final linear layer replaced by two randomly initialized linear layers that each have d units. This network takes in the image X and its two final layers output $\boldsymbol{\mu}_{enc}(X)$ and $\boldsymbol{\sigma}^2_{enc}(X)$. Rather than directly drawing samples from $P_{Z|X}$, we leverage the reparameterization trick and draw samples $E \in \mathbb{R}^d$ from a spherical Gaussian distribution, $E \sim \mathcal{N}(0, \boldsymbol{I}_d)$, where $\boldsymbol{I}_d$ is the d-dimensional identity matrix. We then compute Z as

$$Z = \boldsymbol{\mu}_{\text{enc}} + \boldsymbol{\sigma}_{\text{enc}} \odot E. \tag{6.12}$$

This allows us to backpropagate gradients as we only need to draw samples from $\mathcal{N}(0, \boldsymbol{I}_d)$ rather than a distribution that depends on the neural network's parameters.

We model the variational approximation of the representation's marginal as a d-dimensional spherical Gaussian, $Q_Z = \mathcal{N}(Z|\mathbf{0}, \gamma^2 \boldsymbol{I}_d)$, where $\gamma \in \mathbb{R}_{>0}$. We then calculate the Rényi divergence $D_\alpha \left(P_{Z|X} \| Q_Z\right)$ using (5.20), derived in Section 5.2.5. For all experiments, we use a value of $\gamma = 1$.

As we only use binary values for Y, we model $Q_{Y|Z}$ and $Q_{Y|Z,S}$ with Bernoulli distributions, $Q_{Y|Z} = \text{Bernoulli}(Y; f(Z))$ and $Q_{Y|Z,S} = \text{Bernoulli}(Y; g(Z, S))$ where f and g are auxillary fully connected networks. These two networks each consist of one single fully connected linear layer with a sigmoid activation function, where f has d units and g has $d+1$ units.

The reconstruction network $P_{X|Z,P}$ consists of two fully connected linear layers and three convolutional layers. The first layer has $d+1$ units, the second has 768, and the three convolutional layers have 64, 32, and 8 channels, respectively. The three convolutional layers each have a kernel size of $(3, 3)$. The two linear layers and first two convolutional layers use a ReLU activation function while the final layer uses a sigmoid activation function.

Overall, our RFPIB system consists of four neural networks, the encoder $P_{Z|X}$, decoder $Q_{X|Z,P}$, and two networks that predict Y, $Q_{Y|Z}$ and $Q_{Y|Z,S}$. Figure 6.1 shows the architecture of the system. After training is complete, we use a logistic regression classifier with default settings from `sci-kit learn` [73] to predict Y from Z. We evaluate accuracy and fairness on these predictions. We also use a second logistic regression classifier as an adversary to predict P from Z. Our goal is to create Z such that the first classifier will accurately predict Y but the adversary will be unable to accurately predict P.

Before training, we preprocess images by taking a 128 by 128 pixel centre crop

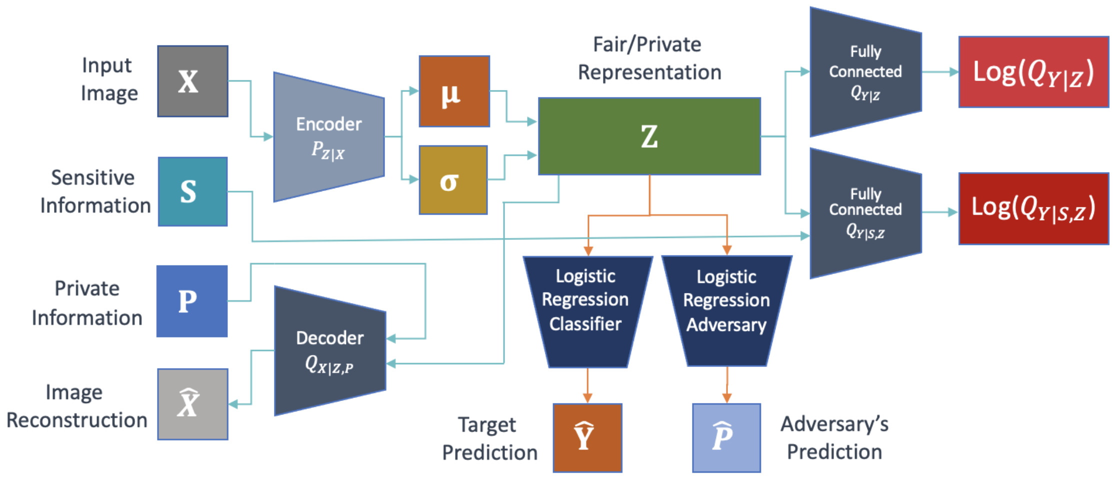


Figure 6.1: Architecture of the RFPIB model. Teal arrows signify elements of our model that are used during training, while orange arrows signify elements that are not part of the model and are used after training is complete. During training, the input image X is given to the encoder $P_{Z|X}$ that generates the mean $\boldsymbol{\mu}_{enc}$ and standard deviation $\boldsymbol{\sigma}_{enc}$ for the distribution of the fair/private representation Z, $\mathcal{N}\left(Z|\boldsymbol{\mu}_{\text{enc}}(X), \text{diag}(\boldsymbol{\sigma}^2_{\text{enc}}(X))\right)$. Then, Z is given to two fully connected networks, one of which is also given the sensitive information S, and Z is also given to the decoder network together with P, which reconstructs the original image as $\hat{X}$. After training, X can be encoded with the encoder the same way as during training and then Z can be used as input to other existing architectures for fair predictions of Y, shown here as a logistic regression classifier that predicts $\hat{Y}$ which we use for our experiments. We also use a second logistic regression classifier as an adversary that outputs $\hat{P}$, attempting to predict P from Z.

of the 218 by 178 pixel CelebA images and a 256 pixel by 256 pixel centre crop of EyePACS images. We then split the training data into a training set and validation set, using 10% of the data for validation, and use the validation set for early stopping. We train for up to 20 epochs, with early stopping triggering when there is no decrease in validation loss for 5 epochs, using a `min_delta` value of 0.

For all experiments, we train using PyTorch on a NVIDIA GP100 GPU. We use $d = 64$ as the dimension of our representation Z, a batch size of 64, and the Adam optimizer with a learning rate of 0.001. We use values of α that vary from 0 to 1, with $\alpha = 0$ signifying

$$D_0(P_{Z|X} \| Q_Z) = -\log Q_Z(z : P_{Z|X}(z) > 0) = 0$$

and with $\alpha = 1$ corresponding to $D_\alpha(P_{Z|X} \| Q_Z)$ being given by the KL divergence $D_{KL}(P_{Z|X} \| Q_Z)$.[1]

6.2.2 Metrics

Measures of Fairness

To evaluate how well our method achieves fairness, we use similar metrics as in Chapter 5. After training is complete, we use a logistic regression classifier to predict Y from Z. We then measure how close we are to achieving accuracy parity, demographic parity, and equalized odds by calculating three different fairness gaps:

- Adherence to accuracy parity is measured by calculating the accuracy gap (Definition 33) and denoted as acc_{gap}.
- Adherence to demographic parity is measured by calculating the demographic parity gap (Definition 30) and denoted as dp_{gap}.
- Adherence to equalized odds is measured by calculating the equalized odds gap (Definition 26) and denoted as $eqodds_{\text{gap}}$.

[1]See Footnote 1 in Chapter 5.

Measure of Utility

To measure the utility of our representation, we report the classification accuracy; see Definition 28. This is done by using a logistic regression classifier to predict Y from Z after training is complete. We denote this accuracy as $acc(Y)$.

Measure of Privacy

To measure how well our representation preserves privacy, we measure how effectively an adversary is able to predict P from Z after our model's training is complete. We use a logistic regression classifier as the adversary and report its classification accuracy as $acc(P)$. We would like to minimize this accuracy, ideally having it be no more than 50%; i.e., performing no better than picking at random. Note that we use an identical process of running a logistic regression classifier on Z to determine both $acc(Y)$ and $acc(P)$; the only difference is that the prediction target is changed from Y to P.

The overall goal of training our RFPIB model is for it to produce Z such that $acc(Y)$ is maximized and $acc(P)$ is minimized, in addition to all three fairness gaps being minimized.

6.2.3 Data

CelebA

We run most experiments on the CelebA dataset [61]. This dataset contains 202,599 images of celebrity faces that each have 40 binary attributes. We use age as our prediction target Y and are interested in both skin tone and gender for the sensitive attribute S and the private attribute P. Age and gender are both included in the dataset; we take them to be binary where $Y = 0$ refers to an age < 30 years old and

$Y = 1$ refers to an age ≥ 30 years old. For gender, we take $S = 0$ or $P = 0$ to mean male and $S = 1$ or $P = 1$ to mean female.

Skin tone is not an attribute included in the dataset; instead we use ITA as a proxy, calculated according to (5.24) as described in Section 5.3.3. We then binarize it, and take $P = 0$ and $S = 0$ to both refer to ITA > 28, taken to mean light skin, and $P = 1$ and $S = 1$ to both refer to an ITA ≤ 28, taken to mean dark skin, matching category thresholds used in [55, 72]. From now on, we use skin tone and ITA interchangeably, with light skin referring to ITA > 28 and dark skin referring to ITA ≤ 28.

We run experiments where $S =$ Skin Tone and $P =$ Gender, and where $S =$ Gender and $P =$ Skin Tone (note that as in Chapter 5, throughout this chapter we use this abuse of notation to denote the attribute each variable represents). We use a training set of 48,000 images, consisting of 24,000 older dark skin images, 12,000 younger light skin images, and 12,000 older dark skin images. Out of these images, 22,311 are male and 25,689 are female. The dataset is fairly balanced over gender but data for the subgroup of older light skin images is missing. When we take S to be skin tone, this corresponds to the challenging fairness problem where there is no data for a subgroup. Evaluations are carried out on a test set of 8,000 images equally balanced across age, skin tone, and gender.

EyePACS

We also run experiments on the EyePACS dataset [23] of retinal fundus images. This dataset consists of 88,692 retinal fundus images of individuals potentially suffering from diabetic retinopathy (DR). As described in Section 5.3.3, we binarize the label of DR status. We then take this label as our prediction target Y, where $Y = 1$ is a positive case of DR and $Y = 0$ is negative. Also as above, we use ITA as a proxy for the patient's skin tone. We use ITA for both P and S, with $P = 0$ or $S = 0$ indicating light skin and $P = 1$ or $S = 1$ indicating dark skin.

We use a training set consisting of 10,346 light skin images referable for diabetic retinopathy (DR = 1, ITA = 0), 5,173 non-referable light skin images (DR = 0, ITA = 0), and 5,173 non-referable dark skin images (DR = 0, ITA = 1). We evaluate performance on a test set containing 2,400 images equally balanced across DR and ITA.

6.3 Experiment Results and Discussion

We present experimental results using RFPIB on the CelebA and EyePACS datasets.

6.3.1 Effects of Rényi Parameter

We begin by studying in isolation how changing the amount of compression affects privacy, fairness, and utility, recalling that minimizing $I(Z;X)$ is what results in compression. The degree of this compression is determined by the Rényi divergence's parameter α, which controls how close $P_{Z|X}$ becomes to Q_Z (a spherical Gaussian). Rényi divergence is non-decreasing in α and thus a higher α results in the $D_\alpha(P_{Z|X} \| Q_Z)$ term having a larger influence on the cost function. As $P_{Z|X}$ is determined by a neural network and Q_Z is fixed, to minimize a larger Rényi divergence the network must choose $P_{Z|X}$ to be closer to Q_Z; hence, $D_\alpha(P_{Z|X} \| Q_Z)$ acts as a regularization term that limits the possible richness of $P_{Z|X}$. A higher α results in more compression while a lower α results in less compression, with $\alpha = 0$ resulting in no compression as $D_0(P_{Z|X} \| Q_Z) = 0$.

We run experiments on the CelebA dataset where we consider a simplified system with only an encoder and reconstruction network, taking $\beta_1 = 0$ and $\beta_2 = 0$, and $\beta_3 = 0.01$. We then vary α linearly from 0 to 1 using 100 different values. For each value of α, after training is complete we encode all images in the test set and then run two logistic regression classifiers on the encoding Z, with one predicting Y and one predicting P. We take Y to be age, and P and S to both be skin tone. We evaluate the utility, fairness, and privacy for each value of α, shown in Figure 6.2.

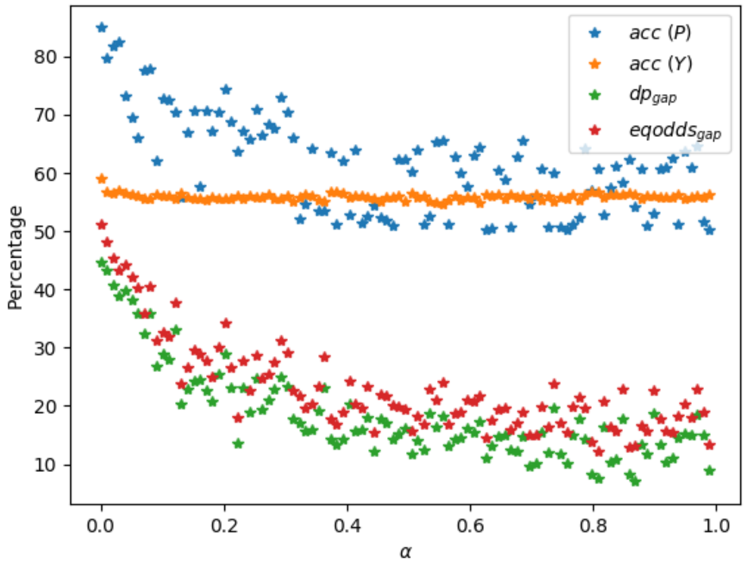


Figure 6.2: Results from 100 different systems trained on the CelebA dataset where $\beta_1 = 0$, $\beta_2 = 0$, $\beta_3 = 0.01$, and α varies from 0 to 1, where Y is age, P is skin tone, and S is skin tone. After training an RFPIB system with the given parameters, we compute the plotted metrics by running two logistic regression classifiers on Z, one to predict Y and one to predict P. For privacy, we want to minimize $acc(P)$ (blue), for fairness we want to minimize dp_{gap} (green) and $eqodds_{\text{gap}}$ (red), and for utility we want to maximize $acc(Y)$ (orange).

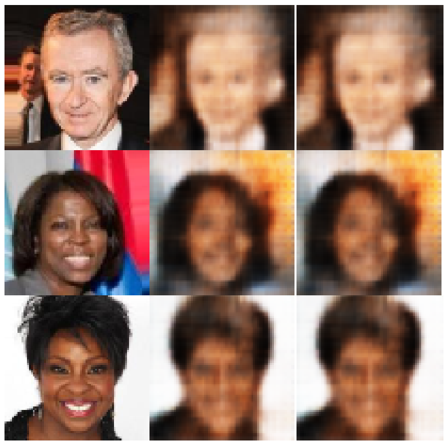
Clearly, as α increases, $acc(P)$ decreases. This is expected since a higher amount of compression results in less information about P being retained in Z. In addition, the two fairness metrics dp_{gap} and $eqodds_{\text{gap}}$ also decrease as α increases. Note that that these results are from systems where $\beta_1 = 0$ and $\beta_2 = 0$, meaning neither Y nor S appears in the cost function. The results show that simply by increasing the amount of compression, we can improve fairness even without directly targeting Y or S, which has implications for applications where these variables are unknown during training. They also illustrate the similarities between the problem of fairness and privacy: in our original Lagrangian formulation (6.1) we aim to minimize $I(Z; P)$ to improve privacy and $I(Z; S|Y)$ to improve fairness. In both cases, we are interested in minimizing the amount of information shared between Z and the private or sensitive variable, and in both cases, upon simplification of the Lagrangian we show that we can minimize $I(Z; X)$ rather than these two terms directly. This indicates that $I(Z; X)$ alone can be used to improve both fairness and privacy simultaneously, as shown in these results.

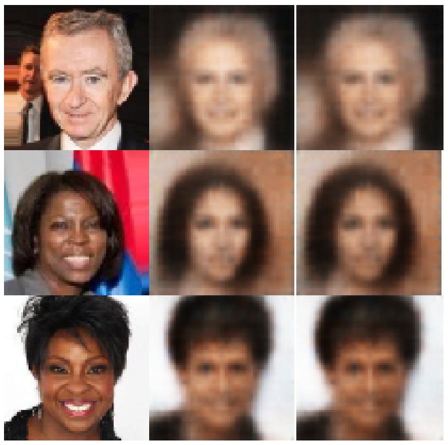
6.3.2 Reconstruction Network Images

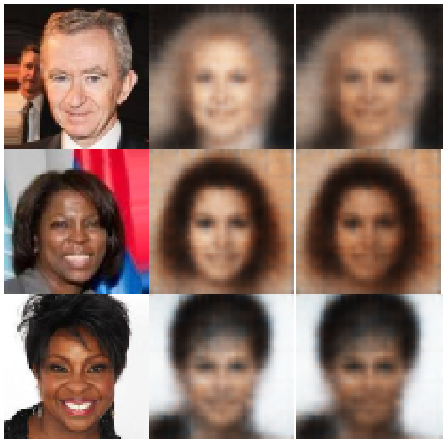
We provide a visual illustration of the previous experiment, showing images generated by the reconstruction network $Q_{X|Z,P}$. We use the same networks as before, systems with $\beta_1 = 0$, $\beta_2 = 0$, $\beta_3 = 0.01$, Y = Age, P = Skin Tone, and S = Skin Tone, and compare networks trained with different values of α. We encode images into Z using the trained encoder network $P_{Z|X}$ and then decode them using the reconstruction network $Q_{X|Z,P}$. The decoder network takes in both Z and P to reconstruct X. If our representation is private as desired then Z does not contain information about P. Therefore, P alone is responsible for determining the skin tone of the reconstructed image. However, if Z is not private then the reconstruction network is able to determine the skin tone for the reconstructed images solely from Z and P does not have an effect. We show reconstructed images from four different networks in Figure 6.3, each trained with a different values of α. The image is reconstructed twice, each reconstruction using same encoding Z but a different value of P.

These images show that as α increases, the quality of reconstructions goes down but P has a greater effect on determining the skin tone of the image, indicating increased privacy. Figure 6.3a has no compression with $\alpha = 0$. These reconstructions look the most similar to the original images but also have the same skin tone as the original images, with P having no effect on the reconstruction. By contrast, the reconstructions in Figure 6.3d with $\alpha = 0.4$ do not resemble the original images as closely but P fully determines the skin tone, with all encoded images reconstructed using $P = 0$ having light skin and images reconstructed using $P = 1$ having dark skin, no matter the original skin tone of the image. This illustrates the trade-off between utility and privacy; more compression results in improved privacy but worsened utility.

We provide a similar visual illustration using newly generated images with the reconstruction network in Figure 6.4. Similar to a Variational Autoencoder, RFPIB is also a generative model capable of creating new content. This is achieved by taking Z to be random noise and inputting it into the reconstruction network. Recall that the Rényi divergence term $D_\alpha(P_{Z|X} \| Q_Z)$ acts as a regularization term, limiting the richness of $P_{Z|X}$ and forcing it to be more similar to Q_Z, a spherical Gaussian. This allows us to take Z to be a sample from a spherical Gaussian distribution, $Z \sim \mathcal{N}(0, \boldsymbol{I}_d)$, and input it into the reconstruction network $Q_{X|Z,P}$ to generate a new image.


(a) $\alpha = 0$


(b) $\alpha = 0.02$


(c) $\alpha = 0.1$

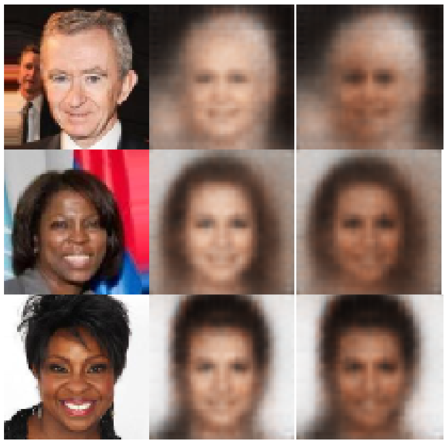
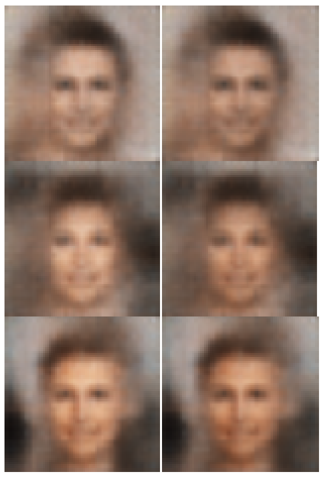
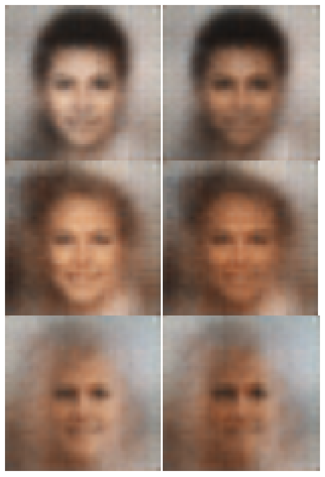

(d) $\alpha = 0.4$

Figure 6.3: Each subfigure shows original images from the CelebA dataset (left column) along with two reconstructions (centre and right columns) created by inputting the image's private representation Z together with P into the reconstruction network $Q_{X|Z,P}$ after our RFPIB system ($\beta_1 = 0, \beta_2 = 0, \beta_3 = 0.01$) was trained on the CelebA dataset using skin tone as the private variable. The centre column reconstruction is with $P = 0$ (light skin) and the right column reconstruction is with $P = 1$ (dark skin). Each subfigure shows reconstructions using a network trained with a different value of α. As α increases, the quality of the reconstruction goes down but the effect that P has on determining the image's skin tone increases.

(a) $\alpha = 0$


(b) $\alpha = 0.02$


(c) $\alpha = 0.1$

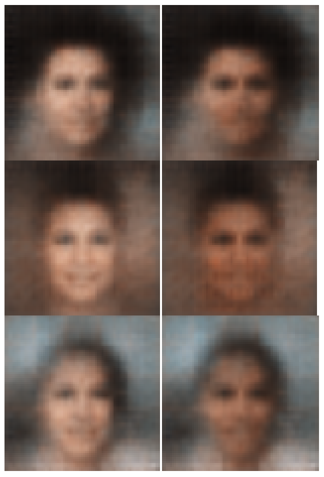
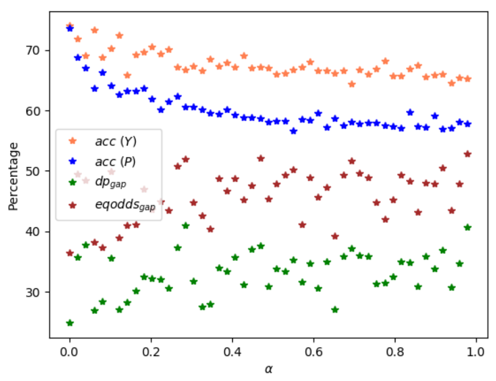

(d) $\alpha = 0.4$

Figure 6.4: Each subfigure shows new images generated by taking Z to be random noise from a spherical Gaussian distribution and inputting it together with P into the reconstruction network $Q_{X|Z,P}$ after our RFPIB system ($\beta_1 = 0, \beta_2 = 0, \beta_3 = 0.01$) was trained on the CelebA dataset using skin tone as the private variable. In the left column, the image is generated using $P = 0$ (light skin), while in the right column, the image is generated using $P = 1$ (dark skin). Each subfigure shows reconstructions using a network trained with a different value of α. No compression with $\alpha = 0$ results in no meaningful output, while increasing α allows new facial images to be generated from random noise. As α increases, P has a greater effect in determining the skin tone of the generated image.

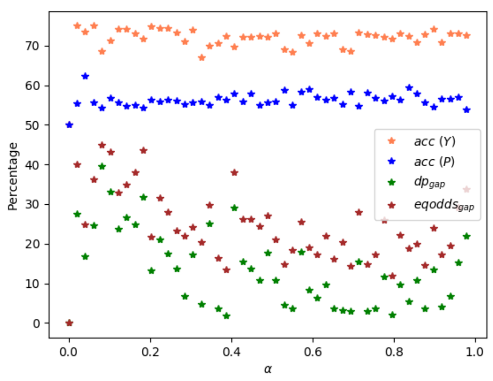
Figure 6.4 shows newly generated images using the same four networks as in Figure 6.3, each trained with a different value of α. Each pair of images is generated using the same random noise for Z but a different value for P. Figure 6.4a shows the importance of using the Rényi divergence $D_\alpha(P_{Z|X} \| Q_Z)$; here α is 0 and there is no divergence, resulting in the reconstruction network failing to generate any meaningful output. However, even a very small value of $\alpha = 0.02$ is enough to result in images of faces being generated, shown in Figure 6.4b. Again, increasing α results in P having a greater effect on the skin tone. In Figure 6.4b where $\alpha = 0.02$, images generated with $P = 1$ are slightly darker than with $P = 0$, whereas in Figure 6.4d where $\alpha = 0.4$ there is a clear difference in skin tone, indicating increased privacy.

6.3.3 Further CelebA Experiments

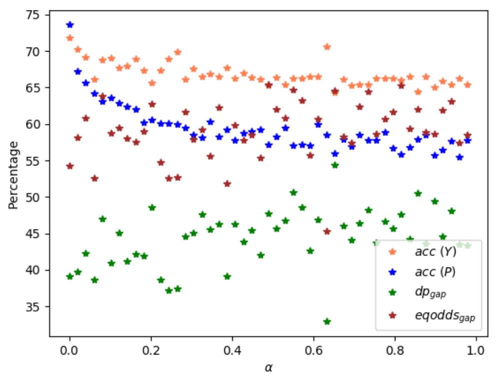
We run further experiments on the CelebA dataset, taking S to be skin tone and P to be gender. We continue to study the effects of changing α and experiment with various combinations of the hyperparameters β_1, β_2, and β_3, shown in Figure 6.5. We begin with systems where $\beta_1 = 0$, $\beta_2 = 10$, $\beta_3 = 0.01$, and α varies between 0 and 1, shown in Figure 6.5a. Compared to the previous experiments in Figure 6.2, we now add the $I(Z;Y|S)$ term to the cost function by changing $\beta_2 = 0$ to $\beta_2 = 10$. Recall that the $I(Z;Y|S)$ term is intended to increase both utility and fairness; maximizing the mutual information between Z and Y ensures accuracy while conditioning on S excludes information about S from being included in Z, increasing fairness. As expected, adding this term results in a significant increase in $acc(Y)$ of over 10%, comparing Figure 6.2 and Figure 6.5a. However, this comes at a cost of worsened fairness metrics. As before, increasing α results in $acc(P)$ decreasing as desired, improving privacy. It also results in a drop in $acc(Y)$, reflecting the trade-off between utility and privacy, but the decrease in $acc(Y)$ is not as significant as the decrease in $acc(P)$, showing that as compression is increased, our method is successful at targeting the information not related to Y to be compressed.



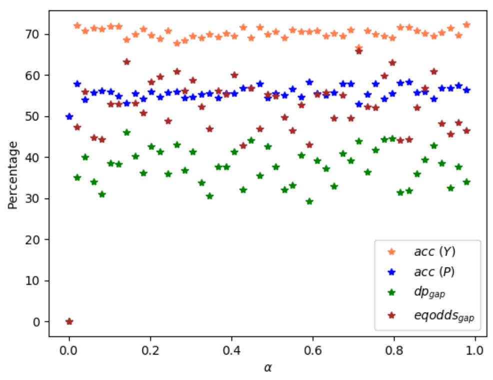
(a) $\beta_1 = 0$, $\beta_2 = 10$, $\beta_3 = 0.01$



(b) $\beta_1 = 0$, $\beta_2 = 10$, $\beta_3 = 0$



(c) $\beta_1 = 10$, $\beta_2 = 0$, $\beta_3 = 0.01$



(d) $\beta_1 = 10$, $\beta_2 = 0$, $\beta_3 = 0$

Figure 6.5: Each subfigure shows results from 50 different systems where α varies from 0 to 1, where Y is age, P is skin tone, and S is gender. After training an RFPIB system with the given parameters, we compute the plotted metrics by running two logistic regression classifiers on Z, one to predict Y and one to predict P. Metrics are computed by running two logistic regression classifiers on Z after training is complete, one to predict Y and one to predict P. For privacy, we want to minimize $acc(P)$ (blue), for fairness we want to minimize dp_{gap} (green) and $eqodds_{\text{gap}}$ (red), and for utility we want to maximize $acc(Y)$ (orange).

In Figure 6.5b we show results from networks where $\beta_1 = 0$, $\beta_2 = 10$, $\beta_3 = 0$, and α varies between 0 and 1, where unlike in Figure 6.5a, we here remove the $I(Z;X|P)$ term from the cost function. Overall, this has positive effects, resulting in a high $acc(Y)$ and a low $acc(P)$. As α is increased, the fairness gaps decrease but $acc(Y)$ and $acc(P)$ stay approximately constant, only decreasing slightly. Despite P not appearing in the cost function, the method is effective at ensuring a low $acc(P)$, again showing that simply compressing Z is enough to ensure notable improvements in privacy. Promisingly, tuning α results in significant improvements to the demographic parity and equalized odds gaps with only very minor decreases in $acc(Y)$.

Figure 6.5c shows results from networks where $\beta_1 = 10$, $\beta_2 = 0$, $\beta_3 = 0.01$, and α varies between 0 and 1. This corresponds to keeping $I(Z;X|P)$ in the cost function as in Figure 6.5a but replacing $I(Z;Y|S)$ with $I(Z;Y)$. This results in similar trends to Figure 6.5a where increasing α results in $acc(Y)$ and $acc(P)$ decreasing. However, the system in Figure 6.5c performs worse on the fairness metrics, which is as expected since S is no longer being directly targeted in the cost function. The systems using $I(Z;Y)$ and $I(Z;Y|S)$ achieve comparable accuracy, which could indicate that for this application knowledge of S is not necessary for the network to predict Y, i.e., knowledge of gender is not necessary to predict age. In this case, using $I(Z;Y|S)$ is clearly beneficial, though we expect that this could vary based on the application.

Figure 6.5d shows results from networks where $\beta_1 = 10$, $\beta_2 = 0$, $\beta_3 = 0$ and α varies between 0 and 1. This corresponds to removing $I(Z;X|P)$ from the cost function compared to Figure 6.5c or replacing $I(Z;Y|S)$ with $I(Z;Y)$ compared to Figure 6.5b. The networks achieve similar results to Figure 6.5b, where both a high $acc(Y)$ and low $acc(P)$ are achieved, but perform worse on the fairness metrics, again showing that the conditioning on S in $I(Z;Y|S)$ provides significant fairness benefits.

6.3.4 Baseline Comparisons

We run more experiments on both the CelebA and EyePACS datasets and compare results from RFPIB with a baseline. For the baseline, we run a ResNet50 network (Section 2.9) pretrained on ImageNet on the original images X. We train the ResNet50 network twice, once to predict $acc(Y)$ and the second time to predict $acc(P)$. Note that the encoder of RFPIB also consists of a ResNet50 network, making the architecture of the baseline similar to RFPIB.

We show results from experiments on CelebA where P is gender and S is skin tone in Table 6.1, experiments on CelebA where P is skin tone and S is gender in Table 6.2, and experiments on EyePACS where both P and S are skin tone in Table 6.3. Note that the training set is approximately balanced across gender but is unbalanced across skin tone, which is why fairness metrics are better in Table 6.2 where S is gender.

Table 6.1: Results for RFPIB ($\beta_1 = 0$, $\beta_2 = 100$, $\beta_3 = 0.001$, $\alpha = 0.4$) trained on CelebA predicting Y = Age where P = Gender and S = Skin Tone, and evaluated on a test set balanced across Age, Skin Tone, and Gender. For metrics with an $\uparrow$ higher is better whereas for $\downarrow$ lower is better.

Method	$acc(P) \downarrow$	$acc(Y) \uparrow$	$acc_{\text{gap}} \downarrow$	$dp_{\text{gap}} \downarrow$	$eqodds_{\text{gap}} \downarrow$
Baseline	92.3	72.2	17.1	33.0	50.1
RFPIB	**57.1**	**75.6**	**10.4**	**16.1**	**26.5**

All experiments show that RFPIB provides clear benefits compared to the baseline. The baseline ResNet50 network achieves significantly higher accuracy predicting P than Y (which is as expected since determining gender or skin tone can be easier than determining age). By contrast, running logic regression classifiers on RFPIB's

Table 6.2: Results for RFPIB ($\beta_1 = 0$, $\beta_2 = 100$, $\beta_3 = 0.001$, $\alpha = 0.4$) trained on CelebA predicting Y = Age where P = Skin Tone and S = Gender, and evaluated on a test set balanced across Age, Skin Tone, and Gender. For metrics with an $\uparrow$ higher is better whereas for $\downarrow$ lower is better.

Method	$acc\ (P) \downarrow$	$acc\ (Y) \uparrow$	$acc_{\text{gap}} \downarrow$	$dp_{\text{gap}} \downarrow$	$eqodds_{\text{gap}} \downarrow$
Baseline	92.5	70.9	0.8	13.9	14.6
RFPIB	**66.7**	**71.8**	**0.3**	**9.4**	**9.8**

Table 6.3: Results for RFPIB ($\beta_1 = 1$, $\beta_2 = 1000$, $\beta_3 = 0.0001$, $\alpha = 1$) trained on EyePACS predicting Y = DR Status where $P = S$ = Skin Tone, and evaluated on a test set balanced across DR Status, Skin Tone, and Gender. For metrics with an $\uparrow$ higher is better whereas for $\downarrow$ lower is better.

Method	$acc(P) \downarrow$	$acc(Y) \uparrow$	$acc_{\text{gap}} \downarrow$	$dp_{\text{gap}} \downarrow$	$eqodds_{\text{gap}} \downarrow$
Baseline	85.9	75.9	4.1	22.4	26.5
RFPIB	**59.8**	**79.9**	**1.8**	**16.7**	**16.8**

encodings Z results in $acc(P)$ lower than $acc(Y)$. Across different datasets, RFPIB drops $acc(P)$ by approximately 25% - 35% while also increasing $acc(Y)$ by several percentage points. We hypothesise that this increase in $acc(Y)$ could be caused by RFPIB's compression acting as a regularization term, preventing overfitting. Finally, these results show that RFPIB also improves upon the baseline for all three fairness metrics, the accuracy gap, the demographic parity gap, and the equalized odds gap.

6.3.5 Discussion

Overall, we show that RFPIB is effective at improving both fairness and privacy while maintaining high utility, significantly outperforming the baseline. By experimenting with various combinations of hyperparameters, we show that privacy and fairness can be improved simultaneously by increasing the amount of compression while trade-offs are typically observed between utility and privacy/fairness. Tuning the Rényi divergence's α parameter makes it possible to find the optimal trade-off that achieves desired results for accuracy, fairness, and privacy. Our experimental results indicate that prioritizing β_2, which controls the $I(Z;Y|S)$ term of the cost function, over β_1, which controls $I(Z;Y)$, is able to achieve significant fairness improvements without compromising on accuracy; however, it is possible that this could vary on different datasets and more experimentation is necessary, which we leave to future work. Our results also indicate that the β_1 and β_2 terms are more effective at increasing $acc(Y)$ than the β_3 term, which controls $I(Z;X|P)$; however, the β_3 term includes neither Y nor S, which makes it suitable for maintaining utility in applications where these variables are unknown. The inclusion of this term provides flexibility for our method to be adapted to a variety of different applications.

Chapter 7

Conclusion

In this thesis, we developed new methods to improve fairness and privacy in machine learning. We proposed RFIB, a novel variational Information Bottleneck fair representation learning method based on Rényi divergence that offers trade-offs in utility, two fairness objectives, and compactness. We showed that it is possible to use Rényi divergence in developing bounds rather than Kullback-Leibler divergence as was traditionally done, and that this provides the benefit of having an additional hyperparameter that can be used to control the amount of compression. We also considered multiple definitions of fairness; compared to prior work which incorporates a single definition, RFIB has the potential benefit of allowing ethicists and policy makers to specify softer and more balanced requirements for fairness that lie between multiple hard requirements. Performing more studies that expand on this idea is an avenue for future research.

Experimenting on three different image datasets, EyePACS, CelebA, and FairFace, we showed that RFIB provides benefits vis-à-vis other methods of record including IB, CFB, and other techniques performing augmentation or adversarial debiasing. For the EyePACS dataset, we trained with ITA as the sensitive variable, used as a proxy for skin tone, and then compared test results using a sensitive variable of ITA and race labels manually annotated by a clinician. The results demonstrated the ability of our method to perform well in this type of protected factor domain adaptation problem, which is particularly important in a setting where the actual protected factor is not revealed for privacy reasons. The type of information blinding approach pursued here may have implications for models that protect against attribute inference attack, another possible direction for future work.

Finally, we extended our method to improve both fairness and privacy, developing RFPIB. We showed that RFPIB can be used to simultaneously target fairness, privacy, and utility. Through experiments on the CelebA dataset, we examined how privacy and fairness relate and showed that the method successfully disentangles the private attribute from the representation. We showed that RFPIB has generative capabilities and that P can be used to control the private attribute of images generated by the reconstruction network. On the CelebA and EyePACS datasets, we demonstrated that RFPIB significantly outperforms a baseline on all desired criteria of accuracy, privacy, and fairness.

Possible directions of future research include studying different techniques to minimize and maximize the mutual information terms, such as using the Mutual Information Neural Estimation algorithm [5] based on the Donsker–Varadhan representation of the Kullback-Leibler divergence [17], or the equivalent algorithm based on a variational representation of the Rényi divergence [9]. We can also examine the use of encoders other than a Gaussian noise encoder, such as an additive noise encoder or propagated noise encoder, and explore ways to extend our work to include private and sensitive attributes that are non-binary, as well as to allow our method to work in cases where the sensitive or private attributes are not explicitly known.

Further future work includes using optimization methods other than the weighted sum method we used, such as the ϵ-constraints method [66]. We can also examine whether adversarial training can be used in conjunction with our approach, comparing our variational approach with adversarial approaches and examining whether adversarial training results in performance improvements or suffers from drawbacks such as mode collapse. Finally, we can compare our approach to privacy with other approaches based on differential privacy [20] and study theoretical guarantees on privacy for our method.

Bibliography

[1] Fady Alajaji and Po-Ning Chen. *An Introduction to Single-User Information Theory*. Springer, Singapore, 2018.

[2] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. In *Proc. ICLR*, pages 1–17, 2017.

[3] Shahab Asoodeh and Flavio P Calmon. Bottleneck problems: An information and estimation-theoretic view. *Entropy*, 22(11):1325, 2020.

[4] Sina Baharlouei, Maher Nouiehed, Ahmad Beirami, and Meisam Razaviyayn. Rényi fair inference. In *Proc. ICLR*, 2020.

[5] Ishmael Belghazi, Sai Rajeswar, Aristide Baratin, R. Devon Hjelm, and Aaron C. Courville. MINE: mutual information neural estimation. *arXiv:1801.04062*, 2018.

[6] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1):151–175, 2010.

[7] Alex Beutel, Jilin Chen, Zhe Zhao, and Ed H Chi. Data decisions and theoretical implications when adversarially learning fair representations. *arXiv:1707.00075*, 2017.

[8] Himesh Bhatia, William Paul, Fady Alajaji, Bahman Gharesifard, and Philippe Burlina. Least kth-order and Rényi generative adversarial networks. *Neural Computation*, 33(9):2473–2510, 2021.

[9] Jeremiah Birrell, Paul Dupuis, Markos A. Katsoulakis, Luc Rey-Bellet, and Jie Wang. Variational representations and neural network estimation of Rényi divergences. *arXiv:2007.03814*, 2021.

[10] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings. In *Proc. NeurIPS*, pages 4349–4357, 2016.

[11] Jacob Burbea. The convexity with respect to Gaussian distributions of divergences of order α. *Utilitas Mathematica*, 26:171–192, 1984.

[12] Philippe Burlina, Neil Joshi, William Paul, Katia D Pacheco, and Neil M Bressler. Addressing artificial intelligence bias in retinal disease diagnostics. *Translational Vision Science and Technology*, 2020.

[13] Simon Caton and Christian Haas. Fairness in machine learning: A survey. *arXiv:2010.04053*, 2020.

[14] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing).* Wiley-Interscience, New York, NY, USA, 2006.

[15] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pages 248–255, 2009.

[16] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[17] Monroe D. Donsker and S.R. Srinivasa Varadhan. Asymptotic evaluation of certain markov process expectations for large time. IV. *Communications on Pure and Applied Mathematics*, 36(2):183–212, 1983.

[18] Flávio du Pin Calmon and Nadia Fawaz. Privacy against statistical inference. In *Proc. 50th Annual Allerton Conference on Communication, Control, and Computing*, pages 1401–1408. IEEE, 2012.

[19] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[20] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[21] Harrison Edwards and Amos Storkey. Censoring representations with an adversary. In *Proc. ICLR*, 2016.

[22] Amedeo R. Esposito, Michael Gastpar, and Ibrahim Issa. Robust generalization via α-mutual information. In *Proc. International Zurich Seminar on Information and Communication*, pages 96–100, 2020.

[23] EyePACS. Diabetic retinopathy detection. Data retrieved from Kaggle, `https://www.kaggle.com/agaldran/eyepacs`, 2015.

[24] Ian Fischer. The conditional entropy bottleneck. *Entropy*, 22(9):999, 2020.

[25] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[26] Bernhard C. Geiger and Gernot Kubin. Information bottleneck: Theory and applications in deep learning. *Entropy*, 22(12), 2020.

[27] AmirEmad Ghassami, Sajad Khodadadian, and Negar Kiyavash. Fairness in supervised learning: An information theoretic approach. In *Proc. IEEE International Symposium on Information Theory (ISIT)*, pages 176–180, 2018.

[28] Manuel Gil, Fady Alajaji, and Tamás Linder. Rényi divergence measures for commonly used univariate continuous distributions. *Information Sciences*, 249:124–131, 2013.

[29] Xavier Gitiaux and Huzefa Rangwala. Learning smooth and fair representations. In *Proc. International Conference on Artificial Intelligence and Statistics*, pages 253–261. PMLR, 2021.

[30] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proc. International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[31] Gabriel Goh. Why momentum really works. *Distill*, 2017.

[32] Ziv Goldfeld and Yury Polyanskiy. The information bottleneck problem and its applications in machine learning. *IEEE Journal on Selected Areas in Information Theory*, 1(1):19–38, 2020.

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

[34] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Proc. NeurIPS*, pages 2672–2680, 2014.

[35] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv:1412.6572*, 2014.

[36] Vincent Grari, Sylvain Lamprier, and Marcin Detyniecki. Fairness-aware neural Rényi minimization for continuous features. In *Proc. Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 2262–2268, 2020.

[37] Adam Gronowski, William Paul, Fady Alajaji, Bahman Gharesifard, and Philippe Burlina. Achieving utility, fairness, and compactness via tunable information bottleneck measures. *arXiv:2206.10043*, 2022.

[38] Adam Gronowski, William Paul, Fady Alajaji, Bahman Gharesifard, and Philippe Burlina. Rényi fair information bottleneck for image classification. In *Proc. 17th Canadian Workshop on Information Theory*, pages 1 – 5, June 2022.

[39] Aditya Grover, Jiaming Song, Ashish Kapoor, Kenneth Tran, Alekh Agarwal, Eric J Horvitz, and Stefano Ermon. Bias correction of learned generative models

using likelihood-free importance weighting. In *Proc. NeurIPS*, pages 11058–11070, 2019.

[40] Moritz Hardt, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In *Proc. NeurIPS*, pages 3315–3323, 2016.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. ICCV*, pages 1026–1034, 2015.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. CVPR*, pages 770–778, 2016.

[43] Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. Beta-VAE: Learning basic visual concepts with a constrained variational framework. In *Proc. ICLR*, 2017.

[44] Catherine F Higham and Desmond J Higham. Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891, 2019.

[45] Geoffery Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning, Lecture 6e. University Lecture, 2012.

[46] Hsiang Hsu, Shahab Asoodeh, Salman Salamatian, and Flavio P Calmon. Generalizing bottleneck problems. In *Proc. IEEE International Symposium on Information Theory (ISIT)*, pages 531–535, 2018.

[47] Sunhee Hwang, Sungho Park, Dohyung Kim, Mirae Do, and Hyeran Byun. Fairfacegan: Fairness-aware facial image-to-image translation. In *Proc. BMVC*, 2020.

[48] Peter Kairouz, Jiachun Liao, Chong Huang, Maunil Vyas, Monica Welfert, and Lalitha Sankar. Generating fair universal representations using adversarial models. *IEEE Transactions on Information Forensics and Security*, 17:1970–1985, 2022.

[49] Kimmo Karkkainen and Jungseock Joo. Fairface: Face attribute dataset for balanced race, gender, and age for bias measurement and mitigation. In *Proc. IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1548–1558, 2021.

[50] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proc. CVPR*, pages 4401–4410, 2019.

[51] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proc. CVPR*, pages 8107–8116, 2020.

[52] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.

[53] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *Proc. ICLR*, 2014.

[54] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019.

[55] Newton M Kinyanjui, Timothy Odonga, Celia Cintas, Noel CF Codella, Rameswar Panda, Prasanna Sattigeri, and Kush R Varshney. Estimating skin tone and effects on classification performance in dermatology datasets. In *Proc. NeurIPS 2019 Workshop on Fair ML for Health*, 2019.

[56] Newton M Kinyanjui, Timothy Odonga, Celia Cintas, Noel CF Codella, Rameswar Panda, Prasanna Sattigeri, and Kush R Varshney. Fairness of classifiers across skin tones in dermatology. In *Proc. International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 320–329, 2020.

[57] Artemy Kolchinsky, Brendan D Tracey, and David H Wolpert. Nonlinear information bottleneck. *Entropy*, 21(12):1181, 2019.

[58] Gowtham R Kurri, Tyler Sypherd, and Lalitha Sankar. Realizing GANs via a tunable loss function. In *Proc. IEEE Information Theory Workshop (ITW)*, pages 1–6, 2021.

[59] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6(6):861–867, 1993.

[60] Yingzhen Li and Richard E. Turner. Rényi divergence variational inference. In *Proc. NeurIPS*, volume 29, pages 1073–1081, 2016.

[61] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proc. ICCV*, December 2015.

[62] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *Proc. ICML*, pages 97–105, 2015.

[63] David Madras, Elliot Creager, Toniann Pitassi, and Richard Zemel. Learning adversarially fair and transferable representations. In *Proc. ICML*, pages 3384–3393, 2018.

[64] Ali Makhdoumi, Salman Salamatian, Nadia Fawaz, and Muriel Médard. From the information bottleneck to the privacy funnel. In *Proc. 2014 IEEE Information Theory Workshop (ITW 2014)*, pages 501–505, 2014.

[65] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv:1804.07612*, 2018.

[66] George Mavrotas. Effective implementation of the ϵ-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, 213(2):455–465, 2009.

[67] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

[68] Michele Merler, Nalini Ratha, Rogerio S Feris, and John R Smith. Diversity in faces. *arXiv:1901.10436*, 2019.

[69] Ilya Mironov. Rényi differential privacy. In *Proc. IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 263–275, 2017.

[70] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[71] Yannis Pantazis, Dipjyoti Paul, Michail Fasoulakis, Yannis Stylianou, and Markos A. Katsoulakis. Cumulant GAN. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[72] William Paul, Armin Hadzic, Neil Joshi, Fady Alajaji, and Phil Burlina. TARA: Training and representation alteration for AI fairness and domain generalization. *Neural Computation*, 34(3):716–753, 2022.

[73] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[74] Stephen Pfohl, Ben Marafino, Adrien Coulet, Fatima Rodriguez, Latha Palaniappan, and Nigam H Shah. Creating fair models of atherosclerotic cardiovascular disease risk. In *Proc. of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 271–278, 2019.

[75] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. On fairness and calibration. In *Proc. NeurIPS*, pages 5680–5689. 2017.

[76] Flavien Prost, Hai Qian, Qiuwen Chen, Ed H Chi, Jilin Chen, and Alex Beutel. Toward a better trade-off between performance and fairness with kernel-based distribution matching. *arXiv:1910.11779*, 2019.

[77] Novi Quadrianto, Viktoriia Sharmanska, and Oliver Thomas. Discovering fair representations in the data domain. In *Proc. CVPR*, pages 8227–8236, 2019.

[78] John Rawls. *Justice as Fairness: A Restatement*. Harvard University Press, 2001.

[79] Alfréd Rényi. On measures of entropy and information. In *Proc. Fourth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 547–562, 1961.

[80] Borja Rodríguez-Gálvez, Ragnar Thobaben, and Mikael Skoglund. A variational approach to privacy and fairness. In *Proc. IEEE Information Theory Workshop (ITW)*, pages 1–6, 2021.

[81] Proteek Chandan Roy and Vishnu Naresh Boddeti. Mitigating information leakage in image representations: A maximum entropy approach. In *Proc. CVPR*, pages 2586–2594, 2019.

[82] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747*, 2016.

[83] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[84] Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. In *Proc. ICML*, 2020.

[85] Aydin Sarraf and Yimin Nie. RGAN: Rényi generative adversarial network. *SN Computer Science*, 2(1):17, 2021.

[86] P. Sattigeri, S. C. Hoffman, V. Chenthamarakshan, and K. R. Varshney. Fairness GAN: Generating datasets with fairness properties using a generative adversarial network. *IBM Journal of Research and Development*, 63(4/5):3:1–3:9, 2019.

[87] Nimit Sohoni, Jared Dunnmon, Geoffrey Angus, Albert Gu, and Christopher Ré. No subclass left behind: Fine-grained robustness in coarse-grained classification problems. *Proc. NeurIPS*, 33:19339–19352, 2020.

[88] Jiaming Song, Pratyusha Kalluri, Aditya Grover, Shengjia Zhao, and Stefano Ermon. Learning controllable fair representations. In *Proc. Twenty-Second International Conference on Artificial Intelligence and Statistics*, pages 2164–2173, 2019.

[89] DJ Strouse and David J Schwab. The deterministic information bottleneck. *Neural Computation*, 29(6):1611–1630, 2017.

[90] Naftali Tishby, Fernando C Pereira, and William Bialek. The information bottleneck method. In *Proc. 37th Annual Allerton Conference on Communication, Control, and Computing*, page 368–377, 1999.

[91] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *Proc. 2015 IEEE Information Theory Workshop (ITW)*, 2015.

[92] Tim van Erwen and Peter Harremos. Rényi divergence and Kullback-Leibler divergence. *IEEE Transactions on Information Theory*, 60(7):3797 – 3820, 2014.

[93] Christina Wadsworth, Francesca Vera, and Chris Piech. Achieving fairness through adversarial learning: an application to recidivism prediction. In *Proc. Conference on Fairness, Accountability, and Transparency in Machine Learning (FATML)*, 2018.

[94] Jian-Jia Weng, Fady Alajaji, and Tamás Linder. An information bottleneck problem with Rényi's entropy. In *Proc. IEEE International Symposium on Information Theory (ISIT)*, pages 2489–2494, 2021.

[95] Marcus Wilkes, Caradee Y Wright, Johan L du Plessis, and Anthony Reeder. Fitzpatrick skin type, individual typology angle, and melanin index in an african population: Steps toward universally applicable skin photosensitivity assessments. *JAMA Dermatology*, 151(8):902–903, 2015.

[96] D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[97] Abdellatif Zaidi, Iñaki Estella-Aguerri, and Shlomo Shamai (Shitz). On the information bottleneck problems: Models, connections, applications and information theoretic views. *Entropy*, 22(2):151, 2020.

[98] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. Learning fair representations. In *Proc. ICML*, pages 325–333, 2013.

[99] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. 2020. `https://d2l.ai`.

[100] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *Proc. 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 335–340, 2018.

[101] Ningshan Zhang, Mehryar Mohri, and Judy Hoffman. Multiple-source adaptation theory and algorithms. *Annals of Mathematics and Artificial Intelligence*, 89(3):237–270, 2021.

[102] Tao Zhang, Tianqing Zhu, Kun Gao, Wanlei Zhou, and S Yu Philip. Balancing learning model privacy, fairness, and accuracy with early stopping criteria. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[103] Han Zhao, Amanda Coston, Tameem Adel, and Geoffrey J Gordon. Conditional learning of fair representations. *arXiv:1910.07162*, 2019.

[104] Han Zhao, Amanda Coston, Tameem Adel, and Geoffrey J Gordon. Conditional learning of fair representations. In *Proc. ICLR*, 2020.