Master's Programme in Computer, Communication and Information Sciences

Generative multi-task learning for the air channel via hierarchical GANs

Juho Kuikka

Aalto University

MASTER'S THESIS

Aalto University
MASTER'S THESIS 2024

Generative multi-task learning for the air channel via hierarchical GANs

Juho Kuikka

Tapiola, 29.7.2024

Supervisor: Professor Esa Ollila
Advisor: Professor Sergiy Vorobyov

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Author
Juho Kuikka

Title
Generative multi-task learning for the air channel via hierarchical GANs

School School of Science

Degree programme Master's Programme in Computer, Communication and Information Sciences

Major Machine Learning, Data Science and Artificial Intelligence **Code** SCI3044

Supervisor Professor Esa Ollila

Advisor Professor Sergiy Vorobyov

Level Master's thesis **Date** 29.7.2024 **Pages** 61 **Language** English

Abstract

In wireless communication, channel model refers to an abstraction that aims to explain how a transmitted signal is altered in the process of wireless communication. Currently, most of the channel models are a compromise between accuracy and computational expenses. In order to achieve higher accuracy with lower computational costs, deep learning based generative modelling has been suggested for the channel modelling problem, with promising results. However, a major drawback within the framework of deep learning is the amount of training data required for success. Since channel measurements are expensive to obtain, methods for enhancing the data efficiency of generative modelling must be investigated. Specifically, as channel models for different locations share inherent similarities, multi-task learning from different, yet related datasets could reduce required data volume for an individual channel model.

This thesis investigates the deep generative modelling via generative adversarial networks (GANs), their Bayesian generalisation, and finally proposes a novel generative modelling scheme for multi-task generation, motivated by Bayesian hierarchical modelling. Our simulations show that our proposed scheme does not only greatly enhance the data efficiency of the channel modelling, but it also decreases instabilities usually present in GAN training. Furthermore, as our proposed modelling scheme is of great generality, it may be utilised in *any* modelling problem where multiple related, but limited datasets are present.

Keywords GANs, Bayesian Deep Learning, stochastic gradient markov chain monte carlo, hierarchical modelling, air channel, multi-task learning

urn https://aaltodoc.aalto.fi

Tekijä
Juho Kuikka

Työn nimi
Generatiivinen monitehtäväoppiminen ilmakanavalle hierarkisilla generatiivisilla kilpailevilla neuroverkoilla

Korkeakoulu Perustieteiden korkeakoulu

Koulutusohjelma Master's programme in Computer, Communication and Information Sciences

Pääaine Machine Learning, Data Science and Artificial Intelligence **Koodi** SCI3044

Valvoja Professori Esa Ollila

Ohjaaja Professori Sergiy Vorobyov

Työn laji Diplomityö **Päiväys** 29.7.2024 **Sivuja** 61 **Kieli** Englanti

Tiivistelmä

Langattomassa tiedonsiirrossa kanavamallilla viitataan abstraktioon, jonka tarkoituksena on selittää, kuinka lähetetty signaali muuttuu tiedonsiirron seurauksena. Tällä hetkellä suurin osa käytetyistä kanavamalleista ovat kompromisseja laadun ja laskennallisten kustannusten välillä. Paremman laadun saavuttamiseksi matalammilla laskennallisilla kustannuksilla, syväoppimiseen perustuvia generatiivisia malleja on ehdotettu kanavamallinnusongelmaan, lupaavin tuloksin. Huomattava varjopuoli syväoppimisen viitekehyksessä on kuitenkin tarve suurelle määrälle dataa. Koska kanavamittaukset ovat kalliita, metodeja datan käytön tehokkuuden parantamiseksi täytyy tutkia. Erityisesti, koska kanavamallit eri sijainneille jakavat luontaisia samankaltaisuuksia, monitehtäväoppiminen (multi-task learning) erilaisista, mutta toisiinsa liittyvistä datajoukoista voisi vähentää vaadittavaa datamäärää yksittäiselle kanavamallille.

Tämä tutkielma tutkii syvää generatiivista mallinnusta generatiivisilla kilpailevilla neuroverkoilla, niiden Bayesilaista yleistystä ja lopuksi ehdottaa uutta mallinnusjärjestelmää monitehtävägeneroinnille (multi-task generation), jonka motivaationa on Bayesilainen hierarkinen mallinnus. Simuloimamme tulokset osoittavat, että ehdotettu mallinnusjärjestelmä ei ainoastaan suuresti paranna datan käytön tehokkuutta kanavamallinnukselle, vaan lisäksi vähentää koulutusprosessin epävakautta, joka on usein läsnä generatiivisten kilpailevien neuroverkkojen koulutuksessa. Lisäksi, koska ehdottamamme mallinnusjärjestelmä on hyvin yleisluontoinen, sitä voidaan käyttää missä tahansa mallinnusongelmassa jossa esiintyy useita toisiinsa liittyviä, mutta kooltaan rajoitettuja datajoukkoja.

Avainsanat GANs, Bayesilainen syväoppiminen, hierarkinen mallinnus, ilmakanava, monitehtäväoppiminen

urn https://aaltodoc.aalto.fi

Preface

This Master's thesis was funded by the Academy of Finland under the academy project AoF-ISAC, led by professor Esa Ollila at Aalto University, Department of Information and Communications Engineering. First, I would like to express my deepest gratitude to professor Esa Ollila for offering me the possibility to do this thesis under the aforementioned academy project. Furthermore, I would like to thank both professors Esa Ollila and Sergiy Vorobyov for providing me excellent guidance throughout the process of writing this thesis, while giving me the full freedom to explore different ideas. Additionally, special thanks to M.Sc. Eeli Susan for the helpful comments and for providing the excellent knowledge in the practicalities of wireless communications.

Tapiola, July 17, 2024

Contents

0. Notations and abbreviations

Notations

$\mathbb{R}$	Field of real numbers
$\mathbb{R}^+$	Field of *positive* real numbers
$\mathbb{C}$	Field of complex numbers
x	Scalar x
$\mathbf{X}$	Matrix X
$\mathbf{I}$	Unit matrix
$\mathbf{x}$	Vector x
$\mathbf{x}^T$	Transpose of x
$\mathbf{X}^{-1}$	Inverse of matrix X
$\mathcal{D}$	Set $\mathcal{D}$
$\lvert\mathcal{D}\rvert$	Cardinality of set $\mathcal{D}$
$\mathcal{S} \times \mathcal{P}$	Cartesian product of sets $\mathcal{S}$ and $\mathcal{P}$
$\log$	Natural logarithm
e	Euler's constant
$\exp a$	e^a
$\mathbf{z} \sim p(\mathbf{z})$	Random variable $\mathbf{z}$ is distributed according to $p(\mathbf{z})$
$p(\mathbf{z} \mid \mathbf{x})$	Conditional probability of $\mathbf{z}$ given $\mathbf{x}$
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and standard deviation σ
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	Multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$
$\mathrm{Exp}(\lambda)$	Exponential distribution with rate λ
$\mathbb{E}[\mathbf{z}]$	Expected value of random variable $\mathbf{z}$
$\mathbb{E}[\mathbf{z} \mid \mathbf{x}]$	Expected value of random variable $\mathbf{z}$ given $\mathbf{x}$
$\mathbb{E}_{p(\mathbf{z})}[\cdot]$	Expectation w.r.t. distribution $p(\mathbf{z})$

$\underset{\mathbf{z}}{\text{argmax}}\ f(\mathbf{z})$	Value of $\mathbf{z}$ that maximises f
$\underset{\mathbf{z}}{\text{argmin}}\ f(\mathbf{z})$	Value of $\mathbf{z}$ that minimises f
$\min f$	Minimum of f
$\max f$	Maximum of f
$\sup f$	Supremum of f
$\mathbf{z} \propto \mathbf{x}$	$\mathbf{z}$ is proportional to $\mathbf{x}$
$\triangleq$	Equal by definition
$\|\cdot\|_n$	nth norm
$\|\cdot\|_L$	Lipschitz norm
$\frac{\partial \mathbf{f}}{\partial x}$	Partial derivative of $\mathbf{f}$ w.r.t. x
$\nabla_x \mathbf{f}$	Gradient of $\mathbf{f}$ with respect to x
$\text{Re}(c)$	Real part of c
$\text{Im}(c)$	Imaginary part of c

Abbreviations

ASGLD	Adam SGLD
BNN	Bayesian (deep) neural network
DNN	Deep neural network
FID	Fréchet Inception Distance
GAN	Generative adversarial network
GP	Gradient Penalty
HMC	Hamiltonian Monte Carlo
IPM	Integral probability metric
JSD	Jensen-Shannon divergence
KL-divergence	Kullback Leibler divergence
LOS	Line-of-sight
LS-GAN	Least Squares GAN
MAP	Maximum a posteriori
MCMC	Markov Chain Monte Carlo
MH	Metropolis-Hastings
MIMO	Multiple-Input and Multiple-Output
MLE	Maximum likelihood estimation
MLP	Multi-layer perceptron
MSGD	Momentum SGD
MSGLD	Momentum SGLD
MSE	Mean squared error
NLOS	Non line-of-sight
ReLU	Rectified linear unit
SGD	Stochastic Gradient Descent
SGLD	Stochastic Gradient Langevin Dynamics
SG-MCMC	Stochastic Gradient MCMC
SISO	Single-Input and Single-Output
SNR	Signal-to-noise ratio
s.t.	such that
TDL	Tapped delay line
VAE	Variational Autoencoder
WGAN	Wasserstein GAN
w.r.t.	with respect to

1. Introduction

Generative modelling of the air transmission channel for wireless communication is relatively novel area of research with no standardized approach. Modelling via deep generative models have been shown to be successful in both time- and frequency domains utilizing for example generative adversarial networks (GANs) or diffusion models. Since deep learning in general is heavily dependent on massive volumes of high-quality data, the data volume becomes the central challenge in channel modelling as the channel measurements are generally expensive to obtain. The problem of limited data is noted on the open literature – which is very scarce at the time of writing – with only few primitive solutions.

Our goal is to develop a rigorous approach for generative modelling from multiple data distributions, inspired by Bayesian hierarchical modelling [1]. Ability to learn from multiple related, but often limited datasets lays a foundation for better data efficiency in the generative channel modelling; it does not only enable structured approach for utilizing many limited datasets, for example channel measurements from different cities, but also the usage of simulated data from similar scenario. Our proposed solution, based on hierarchical Bayesian GANs, is shown to be effective for utilizing data from different data distributions in such way that enhances the modelling of any limited source distribution.

This thesis is organized as follows: In Chapter 2, we introduce the main deep learning concepts required for our modelling purposes. Chapter 3 discusses both Bayesian inference in general level as well as develops a mathematical view of Bayesian neural networks (BNNs) with an illustrative example on hierarchical modelling for multi-task prediction. In Chapter 4 we develop a rigorous view of deep generative modelling via

both classical, as well as Bayesian GANs. Furthermore, Chapter 4 extends the Bayesian GAN formulation to the realm of hierarchical modelling for multi-task generative modelling. Chapter 5 illustrates how hierarchical GAN modelling can be used for generative modelling of the air channel from multiple, related channel datasets. In Chapter 6 we discuss about the results of the thesis, as well as point out possible future research directions.

Mathematical Notations

First, we introduce the mathematical notations used in this thesis. *Vectors* are denoted by lowercase boldface letters or symbols, e.g., $\boldsymbol{\theta}$ and $\mathbf{x}$ are taken to be vectors. *Matrices* are denoted by uppercase boldface letters, such as the identity matrix $\mathbf{I}$. For *sets*, we use uppercase calligraphic letters, e.g., $\mathcal{D}$ denotes training data for any machine learning algorithm, whence $|\mathcal{D}|$ denotes the *cardinality* of this set. For exhaustive listing of mathematical notations, we refer to Chapter 0.

2. Deep Neural Networks

In its modern usage, term *neural network* refers to a differentiable function which can be expressed as a computational graph which nodes are some primitive operations, such as matrix multiplication, and edges represent numerical data [1]. The simplest form of such graph can be built as a linear series of nodes, called *layers* of the network. The *depth* of the neural network comes from the usage of many such layers, making the network *deep*. Mathematically, this kind of simple network may be represented as a composition of functions, for example, for a three-layer network we have:

$$f_{NN}(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x};\boldsymbol{\theta}_1);\boldsymbol{\theta}_2);\boldsymbol{\theta}_3),$$

where $\{\boldsymbol{\theta}_i\}_{i=1}^3$ are the *weights* of the layers. If the function f_i is a linear map, the whole network can be expressed as a simple linear mapping. For this reason, we usually are interested in using *nonlinear* mappings, allowing the composition to represent much more complex functions. Indeed, even this kind of simple network can represent a wide variety of functions, more formally, *capable of approximating any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy* [2], given there are enough of network weights. From this result follows that neural networks are often referred as *universal approximators*.

2.1 Layers and activation functions

The simplest kind of layer in neural networks is called the *linear layer*. The linear layer performs an affine transformation:

$$f_i(\mathbf{x};\boldsymbol{\theta}) = \mathbf{W}^T\mathbf{x} + \mathbf{b},$$

where $\boldsymbol{\theta} = [\mathbf{W}\ \mathbf{b}]$. The linear layer is often called *fully connected* layer,

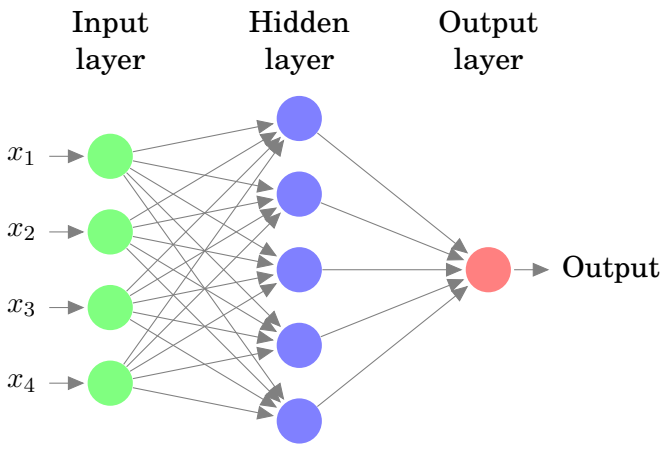

Fig. 2.1. Simple neural network with one hidden layer.

which is illustrated in Figure 2.1. For historical reasons, neural network consisting only of linear layers is often called a *multi-layer perceptron* (MLP). There exists a plethora of different layer classes in modern deep learning: for example *convolutional* layers have had a huge success in computer vision tasks, whereas *recurrent* layers can be useful for working with data with inherently sequential nature. However for the purposes of this thesis, a simple linear layer is useful enough to meet all of our goals. Thus we limit the discussion of different architectural choices and instead focus more on the probabilistic modelling aspects.

In order to learn nonlinear mappings we must use some nonlinear *activation functions* alongside the affine transformation of the linear layer, i.e., let $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ be some nonlinear mapping and set

$$f_i(\mathbf{x}; \boldsymbol{\theta}) = \sigma(\mathbf{W}^T \mathbf{x} + b).$$

The choice of activation function has an effect on the *hypothesis space* of the neural network, but generally the *universal approximation* theorem holds for a broad class of commonly used activation functions, given that there is enough weights in the network [3]. Historically, the belief was that the activation functions should be highly nonlinear, sigmoid and hyperbolic tangent being common choices [3], and the empirical evidence pointed that networks with hyperbolic tangent as an activation function were faster to converge in comparison to those with sigmoidial activation function [4]. An intuitive explanation why a hyperbolic tangent may outperform the sigmoidial activation function follows from the fact that the learning problem is close to learning a linear model when the inputs are close to zero; that is,

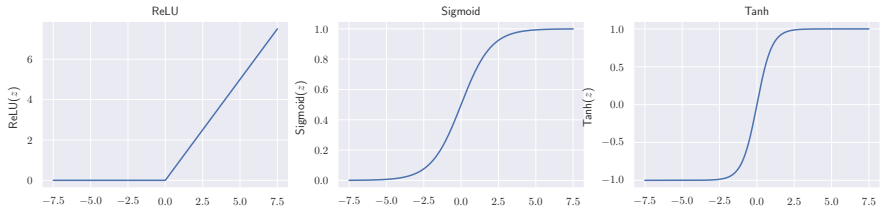

Fig. 2.2. Illustration of different activation functions.

$$\hat{y} = \boldsymbol{\theta}_3^T \tanh(\boldsymbol{\theta}_2^T(\tanh(\boldsymbol{\theta}_1^T \hat{\mathbf{x}})) \approx \boldsymbol{\theta}_3^T \boldsymbol{\theta}_2^T \boldsymbol{\theta}_1^T \hat{\mathbf{x}},$$

when $\|\boldsymbol{\theta}_1^T \hat{\mathbf{x}}\|_2,\ \|\boldsymbol{\theta}_2\|_2,\ \|\boldsymbol{\theta}_3\|_2 \leq \epsilon$, where ϵ lies sufficiently close to neighborhood of zero.

However, in the modern practice it is common to use piecewise linear activation functions due to their more informative gradients and much reduced computational costs. For example, one of the most commonly used activation functions is the *rectified linear unit* (ReLU):

$$\text{ReLU}(z) = \max(0, z).$$

ReLU has favorable computational properties as its gradient is equal to 1 whenever the neuron is active and zero when its inactive. Its most common downside is the fact that it is not differentiable everywhere in its domain, but it is rarely a problem in any practical setting [3]. Another benefit of ReLU is its computational efficiency, compared to sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

and hyperbolic tangent

$$\tanh(z) = 2\sigma(2z) - 1$$

activations, which are nontrivial to compute. The graphs of the aforementioned three activation functions are displayed in Figure 2.2.

2.2 Training a neural network

Even though the universal approximation theorem tells us that in theory the neural networks can approximate a broad class of functions, given enough weights, it does not tell us how can one obtain the set of weights

in order to obtain the desired function. A common paradigm for inferring the weights of the network in order to learn arbitrary functions is called *gradient based learning*, where we let the gradient of the *prediction error* flow backwards through the network. This process is called the *backpropagation* and is based on the *chain rule* of calculus. After the gradients have been computed, each of the parameters may be then updated according to some *optimization* rule.

2.2.1 Chain rule of calculus and backpropagation

Consider two functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$ s.t. $z = f(x)$ and $y = g(z)$. The chain rule of calculus states that

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z}\frac{\partial z}{\partial x}.$$

Since we introduced neural networks as function compositions, it is easy to see how the chain rule can be applied in order to obtain gradients of the parameters with respect to (w.r.t.) some predetermined *loss function*. The process which *automatically* calculates the desired gradients is called the *backpropagation*, and its customary algorithm in *automatic differentiation* typically implemented in any general deep learning framework, such as `PyTorch` or `Tensorflow`.

2.2.2 Learning as an optimization problem

As stated before, the universal approximation theorem do not tell us how to find a set of parameters in order to approximate the desired function. Similarly, the backpropagation itself does not tell how to use the obtained gradient information. Since the *learning problem* may be stated as an optimization one, many optimization algorithms may be used in order to use the gradient information to update the parameters of the model. For illustrative example, let $f_{NN} : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$ be a nonlinear map such that it maps the *covariates* or *features* $\mathbf{x} \in \mathbb{R}^d$ to real valued *outcome* or *response* y using the model parameters $\boldsymbol{\theta} \in \mathbb{R}^p$. Furthermore let $\mathcal{L} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be some *loss* or *objective* function which compares the models prediction to the *true value* of the outcome. If we have a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ of observations, $|\mathcal{D}| = N$, the learning or optimization task is to find such parameters $\boldsymbol{\theta}$ that minimize the loss function:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, f_{NN}(\mathbf{x}_i, \boldsymbol{\theta})). \tag{2.1}$$

In statistical learning theory this approach is called the *empirical risk minimization*, where *risk* is used to denote *loss* or *objective* function. The term *empirical* follows from the fact that we do not know the true data generating distribution analytically, but only possibly noisy instances generated by the true distribution. This approach however can be prone to *overfitting*, i.e., the predictor with enough capacity could memorize all of the training samples, producing a perfect zero risk, while having a poor *generalization* performance. As neural networks are in theory universal approximators, the risk of overfitting can be significant, and must be reduced by some *regularization* techniques.

If we were to use the model negative log-likelihood as the loss function s.t. $\mathcal{L}(y, f_{NN}(\mathbf{x}, \boldsymbol{\theta})) = -\log p_{\text{model}}(y, f_{NN}(\mathbf{x}; \boldsymbol{\theta}))$ the optimization problem (2.1) would equal to maximizing the expectation of the models likelihood in the log-space over the *empirical distribution* $\hat{p}_{\text{data}}$ induced by the training dataset:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}}[\log p_{\text{model}}(\hat{y}, y)], \tag{2.2}$$

where $\hat{y} \triangleq f_{NN}(\mathbf{x}, \boldsymbol{\theta})$. This approach is also called the *maximum likelihood estimation* (MLE).

Since the amount of training data required for training most deep learning models is large, it is often unfeasible to use the whole dataset – historically called the *batch* – in the optimization procedure. A common way of dealing with this problem is to use *minibatches* of data $\mathcal{B} \subset \mathcal{D}$ s.t. $|\mathcal{B}| \ll |\mathcal{D}|$. Even though historically it was usual to call algorithms that used the whole training dataset for the gradient evaluation as batch algorithms, in the modern approach where datasets are large by default we are often using terms batch and minibatch interchangeably. Minibatching introduces some noise to the gradients w.r.t. the loss, which we may assume to be normally distributed due to the *central limit theorem*, given the cardinality of the minibatch is of magnitude of hundreds. However, as the nonlinearities of the neural network will make the loss landscape highly *nonconvex*, the noise introduced by minibatching the dataset has been empirically shown to help the optimizer in escaping from shallow local minimas that

might not yield a good generalization. The nonconvexity of the learning problem also comes with the expense that we will not have any guarantees to converge to global optimum, which however can prevent overfitting in many cases.

2.2.3 Stochastic optimization

Since our goal is to find a set of parameters $\boldsymbol{\theta}^\star$ that minimize some loss function, we may clear the notation a bit and write the objective as function of the models parameters:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},y\sim\hat{p}_{\text{data}}}[\log p_{\text{model}}(\hat{y}, y)],$$

where we again denote $\hat{y} \triangleq f_{NN}(\mathbf{x}, \boldsymbol{\theta})$. Note that we are considering the maximum likelihood estimation here for the ease of exposure.

Since we are relying on gradient information in the optimization procedure, we are interested in the gradient of J w.r.t. the models parameters $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$. Since we are relying on minibatches of data to alleviate the cost of evaluating the gradient w.r.t. the whole dataset, the estimate is inherently noisy. However, the empirical and analytical results have shown that this is hardly an issue. Since the standard error of the mean is given by $\frac{\sigma}{\sqrt{M}}$, where σ is the true standard deviation of the samples, and $M = |\mathcal{B}|$, we observe that the accuracy of the estimate does not scale linearly w.r.t. the number of datapoints used, whereas the gradient evaluation does [3].

Second consideration follows from the fact that there are usually redundancies in the dataset, i.e., many of the samples have an identical contribution to the gradient estimate thus wasting computational resources. Keeping these considerations in mind together with the fact that the noise from minibatching the data can actually help the optimizer to escape from shallow local minimas, the usage of stochastic optimization algorithms is the de-facto practice in modern deep learning. It should also be stated that for our purposes optimization methods based on the *first-order* gradient information are sufficient; if one is to use higher-order methods, the accuracy of the gradient estimate becomes a larger concern. For further references on accuracy considerations for higher-order gradient methods see for example [3].

Stochastic Gradient Descent

One of the simplest stochastic optimization algorithm is the *stochastic gradient descent* (SGD). SGD updates the parameters via the noisy gradient information produced by the expectation over minibatch $\mathcal{B}$ of samples

$$\hat{J}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i \in \mathcal{S}} \log p(\hat{y}_i, y_i)$$

where $\mathcal{S}$ is the set of indexes of the minibatch s.t. $\mathcal{B} = \{\mathcal{D}_i\}_{i \in \mathcal{S}}$. Now the *unbiased* estimate of gradient $\hat{\boldsymbol{g}}$ is given by $\nabla_\theta \hat{J}(\boldsymbol{\theta}_t)$, which is then used to update the model parameters as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta_t \hat{\boldsymbol{g}}_t,$$

where η_t is the *learning rate* at timestep t. The selection of learning rate in context of deep learning is stated to be *more art than science* [3], but sufficient theoretical conditions for the SGD to converge are met if

$$\sum_{t=1}^{\infty} \eta_t = \infty \text{ and } \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

The requirement for the learning rate to decay follows from the noise introduced by minibatching the data. Even though these conditions in theory are sufficient, the choice of η_1 and the decaying schedule can play a major part in the rate of convergence and even in the generalization obtained [3]. A common practice in deep learning is to decay the learning rate linearly.

Momentum SGD

Even though SGD is widely used method due to its simplicity, in highly convex optimization problems, such as learning weights for deep neural networks, its convergence can be slow. In order to speed up the convergence, it is often useful to incorporate *past* gradient information to dampen oscillations and escape shallow modes. The use of *momentum* in optimization algorithms can be traced back to *Polyak's heavy ball method* [5], which introduces a *velocity* $\boldsymbol{v}$ for a particle moving in the optimization landscape. The analogy is simple; when a particle with mass obtains high velocity, and consequently high momentum, it will not stop in the face of small uphill or oscillate much in a rough terrain. The momentum is introduced to the plain SGD by accumulating exponentially decaying moving average of the past gradients in the update. In the momentum SGD (MSGD) we usually

assume unit mass, so the velocity itself may be regarded the momentum of the particle. The update rule in MSGD is given by:

$$\begin{aligned} \boldsymbol{v}_t &= \alpha \boldsymbol{v}_{t-1} - \eta_t \hat{\boldsymbol{g}}_t \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_t, \end{aligned}$$

where $\alpha \in [0, 1)$ is a hyperparameter determining how quickly the contribution of the past gradients is vanishing. The ratio of α to η determines the magnitude of effect the past gradients have on the update direction. Similarly to the learning rate, α may also be decayed over time, however, it is of less importance compared to the shrinkage of the learning rate [3].

Adaptive optimization methods

Learning rate is one of the hardest hyperparameters to select, as it has a significant impact on the models generalization ability [3]. Furthermore, the objective function is often highly sensitive in some directions in the parameter, while being relatively insensitive in others. Thus, a natural approach would be to set independent learning rates for each parameter and adapt these automatically throughout the learning process.

For the aforementioned reasons, optimization methods that adaptively change the learning rate for each hyperparameter individually are widely of great interest in deep learning. Methods such as Adam [6] and RMSProp [7] are widely used, but for a long time they were not well understood. It appears however, that when studied as *preconditioned* SGD, that the adaptive methods indeed outperform SGD in escaping the saddle points of the optimization landscape and can converge faster to second-order stationary points [8]. For us it suffices to know that the both Adam and RMSProp have a great empirical robustness and performance, but furthermore can be analytically studied as preconditioned SGD, where the preconditioner is estimated in an online manner.

2.2.4 Regularization

Since neural networks are universal approximators, they can be prone to overfitting. As mentioned in Subsection 2.2.2, to remedy the issue of overfitting, usage of some *regularization* technique is usually advisable. There exists a plethora of regularization techniques in the modern deep learning practice, however, many of these have no clear statistical motivations. A simple regularization technique arising from classical statistics is

to penalise the model for having large weights. A common penalty term, which is added to the loss function to minimize, is given as:

$$R(\boldsymbol{\theta}; \lambda) = \frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}, \tag{2.3}$$

where $\boldsymbol{\theta}$ are the model parameters, and λ is a hyperparameter controlling the importance of the regularization term. This choice of regularizer is called a *weight decay* in the machine learning literature [9]. Plugging the regularizing term in the optimization problem given in Equation (2.2) gives us:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left[\mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} (\log p_{\text{model}}(\hat{y}, y)) - R(\boldsymbol{\theta}; \lambda) \right]. \tag{2.4}$$

Note that here we are optimizing w.r.t. the model parameters, $\boldsymbol{\theta}$, i.e., the hyperparameter λ is selected *prior* to the model training process. The selection of both the regularization term and its hyperparameters requires expertise as well as experimenting. In machine learning, a common practice is to use an additional *validation set* of the data in order to select the regularization method.

3. Bayesian Machine Learning

In the *frequentist* statistics we are interested in finding a set of parameters that maximizes some quality metric of a model, for example, a likelihood: $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p(\mathcal{D} \mid \boldsymbol{\theta})$. In the context of Bayesian statistics our goal is to find a *posterior distribution* of the parameters given the data, $p(\boldsymbol{\theta} \mid \mathcal{D})$, which can be computed using Bayes' theorem:

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\boldsymbol{\theta})p(\mathcal{D} \mid \boldsymbol{\theta})}{p(\mathcal{D})}.$$

Here $p(\boldsymbol{\theta})$ is called a *prior* distribution of the parameters, in which we can state our *beliefs* about how the parameters should be distributed. The prior term also provides a natural regularisation term for the model at hand. Term $p(\mathcal{D}) = \int p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$ is called the *marginal likelihood* or *evidence*, and is used as normalization constant in order to obtain a proper probability distribution. Since the computation of the evidence is intractable for more complex models, such as neural networks, we must rely on some *approximate* method for approximating the posterior distribution.

Bayesian computation literature has suggested several successful strategies for approximating complex posterior distributions, however, there exists no off-the-shelf methods for approximating highly *multimodal* posteriors [10]. Major ideas for approximating the posterior either rely on *approximate inference*, e.g., fit a normal distribution that minimises some distributional divergence between the true posterior and the approximation, or sampling. Sampling based algorithms rely usually on constructing a *Markov chain* that in the limit converges to the true posterior distribution, from which posterior samples are then collected. Markov chain based sampling algorithms are called Markov chain Monte Carlo (MCMC), and are considered to be the gold standard of Bayesian computation since they do not require any assumptions on the form of the posterior distribution.

3.1 Motivation for Bayesian approach

One of the major motivations for being Bayesian comes from the estimation of uncertainty. For example, consider a regression model $f : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}$ s.t. f maps the observed covariates $\mathbf{x} \in \mathbb{R}^d$ to prediction $\hat{y} \in \mathbb{R}$ using parameters $\boldsymbol{\theta} \in \mathbb{R}^m$. In the case of maximum likelihood estimate, $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmax}} \log p(\mathcal{D} \mid \boldsymbol{\theta})$, we obtain a prediction $\hat{y} = f(\mathbf{x}; \hat{\boldsymbol{\theta}})$ with no measure of uncertainty. When we have the posterior distribution for the parameters $p(\boldsymbol{\theta} \mid \mathcal{D})$, we can obtain posterior predictive distribution for new datapoint $\hat{y}$ as follows:

$$p(\hat{y} \mid \hat{\mathbf{x}}, \mathcal{D}) = \int p(\hat{y} \mid \hat{\mathbf{x}}, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta}, \tag{3.1}$$

where $p(\hat{y} \mid \hat{\mathbf{x}}, \boldsymbol{\theta})$ is the model likelihood for the new observation. The so obtained distribution (3.1) gives us a meaningful way of estimating the *uncertainty* of predictions.

In generative modeling, however, we are not necessarily interested in the uncertainty estimation. Yet working with a posterior distribution instead of a point estimate can be useful if the posterior is multimodal, which usually is the case when working with Bayesian neural networks. Other way to frame this phenomenon is to note that with highly flexible models and complex data there usually exists multiple models that have an equal training performance but yield different generalisations, called *underspecification* [1].

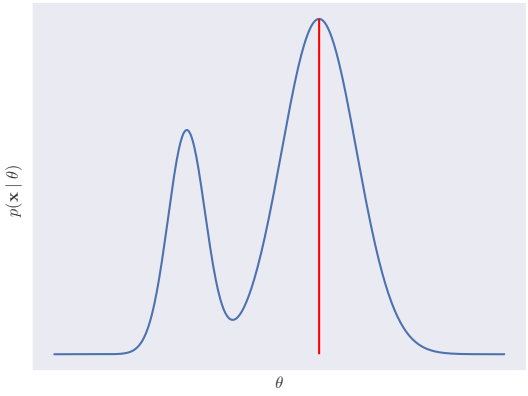

Fig. 3.1. Maximum likelihood estimation misses a second explanation for the data.

Consider a toy example illustrated in Figure 3.1: optimal maximum likelihood estimate will recover a model that explains the data through the

set of parameters $\boldsymbol{\theta}$ that maximizes the likelihood, while missing an alternative explanation for the data. If we have the access to the whole distribution of $\boldsymbol{\theta}$, we can recover every possible model explaining the data. This is of high importance in generative modeling. Indeed, if we want to have a model that can produce samples with good variety, we must be able to produce samples that come from the mode missed by the maximum likelihood estimate. Note the important distinction between multimodal data distribution and multimodal parameter distribution: the maximum likelihood estimation *does not* prohibit the modelling of multimodal data distributions – Bayesian modelling only may make it easier to model those distributions via the introduction of posterior distribution over the model parameters.

Another strength of the Bayesian approach comes from the idea of *hierarchical modelling* [11]. This kind of modelling scheme allows us to model multiple related datasets in a way that captures the individual properties of each dataset while modelling the similarities shared by the datasets. In the following sections we will see that this kind of modelling scheme allows us to have a mathematically sound approach for *multitask learning* and *Bayesian transfer learning*.

3.2 Maximum a posteriori approximation

Simplest way to approximate a posterior distribution is to estimate the posterior as a point-mass $\hat{\boldsymbol{\theta}}$ in the probability space, i.e.,

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{\theta} \mid \mathcal{D}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, [\log p(\boldsymbol{\theta}) + \log p(\mathcal{D} \mid \boldsymbol{\theta})]. \tag{3.2}$$

This estimate is often called the *maximum a posteriori* (MAP) estimate. Note how the MAP estimate consists of log-likelihood function and the prior density. If we were to assign an isotropic Gaussian prior to the model parameters, $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \lambda \mathbf{I})$, the prior density to maximize in Equation (3.2) becomes $\log p(\boldsymbol{\theta}) \propto -\frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} = R(\boldsymbol{\theta}; \lambda)$. Note that with this particular prior choice the MAP estimate equals to the regularized MLE problem given in Equation (2.4), i.e., there exists a clear connection between the MAP and regularized MLE. Even though the MAP estimate does not give us any measure of uncertainty, it can be useful starting point for Bayesian modelling, and furthermore it sheds a light on how the prior selection can

be seen as regularization technique.

3.3 Sampling based posterior computation

As mentioned in the beginning of Chapter 3, MCMC based algorithms are considered to be the "gold standard" of Bayesian computation. MCMC methods rely on constructing a Markov process with stationary distribution equal to the desired posterior distribution, $p(\boldsymbol{\theta} \mid \mathcal{D})$, and running the simulation long enough for the draws from the chain to be close to this stationary distribution [11]. A simple algorithm for constructing such Markov process is called the *Metropolis algorithm* [11], see Algorithm 1. In Metropolis algorithm, the idea is to adapt a random walk with an acceptance/rejection rule such that the walk converges to the desired distribution. The algorithm starts with an initial state $\boldsymbol{\theta}^0$ for which $p(\boldsymbol{\theta}^0 \mid \mathcal{D}) > 0$. Then for $t = 1, 2, \ldots$ new draws are acquired by proposing a new state $\boldsymbol{\theta}^\star$ from a proposal distribution, $J_t(\boldsymbol{\theta}^\star \mid \boldsymbol{\theta}^{t-1})$, which is required to be symmetric, i.e., $J(\boldsymbol{\theta}_a \mid \boldsymbol{\theta}_b) = J(\boldsymbol{\theta}_b \mid \boldsymbol{\theta}_a)$. The proposal is then accepted with a probability $\min(1, r)$, where

$$r = \frac{p(\boldsymbol{\theta}^\star)}{p(\boldsymbol{\theta}^{t-1})}.$$

In layman terms, if the proposal increases posterior density, it is always accepted, but only sometimes accepted in cases when the posterior density would decrease. For a proof why this process converges to the specified distribution, see for example [11]. A generalisation of Metropolis algorithm where the jumping distribution is not required to be symmetric is called the *Metropolis-Hastings (MH) algorithm*, which usually increases the speed of the random walk in a sense that it requires fewer iterations for the process to converge to the stationary distribution.

In order for the *MH* to allow for asymmetric proposal distribution the ratio r is replaced by *ratio of ratios*:

$$r = \frac{p(\boldsymbol{\theta}^\star) J_t(\boldsymbol{\theta} \mid \boldsymbol{\theta}^\star)}{p(\boldsymbol{\theta}) J_t(\boldsymbol{\theta}^\star \mid \boldsymbol{\theta})},$$

which is called the *Hastings correction*. Note that in both Metropolis and MH algorithms, we need to know the target densities only up to a normalization constant, i.e., we can work with unnormalized distributions, which is particularly useful for any posterior distribution.

Algorithm 1 Metropolis algorithm [1]

Initialize $\boldsymbol{\theta}^0$
for $t = 1, 2, \ldots$ **do**
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{t-1}$
  Sample $\boldsymbol{\theta}^\star \sim J_t(\boldsymbol{\theta}^\star \mid \boldsymbol{\theta})$
  $r \leftarrow \frac{p(\boldsymbol{\theta}^\star)}{p(\boldsymbol{\theta})}$
  $A \leftarrow \min(1, r)$
  Sample $u \sim U(0, 1)$
  if $u \leq A$ **then**
    $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^\star$
  else
    $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}$
  end if
end for

There exists numerous ways of modifying the Metropolis or MH algorithms to be more efficient. One can for example reparameterize the model to allow for easier sampling, and even after that there is an infinite number of ways to select the proposal distribution. Even then, the algorithm suffers from its inherent local random-walk nature, zig-zagging in the target distribution, which causes the samples to be serially correlated and the algorithm to take a long time to converge, especially in high-dimensional target distributions usually present in a deep learning setting. Hamiltonian Monte Carlo (HMC) [12] borrows its idea from physics in order to allow for more distant steps in the target distribution by introducing a *moment* variable v_i for every component θ_i in the target space.

3.3.1 Hamiltonian dynamics

As the name suggests, HMC is based on *Hamiltonian dynamics* [12]. Hamiltonian dynamics for a physical system can be understood simply by thinking of a frictionless body sliding on a two dimensional surface with varying height. The state of this system consists of *position* of the body, $\mathbf{x} \in \mathbb{R}^2$ and its *momentum*, $\boldsymbol{v} \in \mathbb{R}^2$, which is the mass of the body times its velocity. Now the *potential energy* of the body is given by $\mathcal{E}(\mathbf{x})$ and its *kinetic energy* by $\mathcal{K}(\boldsymbol{v}) = \|\boldsymbol{v}\|_2^2/2m$, where m denotes the mass of the body. The set of possible values of $(\mathbf{x}, \boldsymbol{v})$ is called the *phase space*, and we define a *Hamiltonian function* for each point of the phase space to be the total energy of the system:

$$\mathcal{H}(\mathbf{x}, \boldsymbol{v}) \triangleq \mathcal{E}(\mathbf{x}) + \mathcal{K}(\boldsymbol{v}).$$

Intuitively, if the body is on a level surface, its momentum will be constant

and its potential energy invariant. If the body is sliding towards upwards slope, its momentum will decrease while its potential energy will increase, leaving the Hamiltonian constant. In statistical scenario, the position of the system is the variable of interest, denoted by the model parameters $\boldsymbol{\theta}$, while the momentum will serve as an auxiliary variable. Furthermore, in statistical setting the potential energy $\mathcal{E}(\boldsymbol{\theta})$ is often defined as the (unnormalized) negative log posterior density of $\boldsymbol{\theta}$, from which we wish to sample, and the kinetic energy $\mathcal{K}(\boldsymbol{v}) = \frac{1}{2}\boldsymbol{v}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{v}$, where $\boldsymbol{\Sigma}$ is some positive definite matrix, known as the *mass matrix*. Often the mass matrix is set to be a scalar multiple of an identity matrix [12].

3.3.2 Integrating Hamilton's equations

Since our goal is to obtain a computer program to produce samples from a specified target distribution, the continuous time system must be discretized in order to update the position and momentum variables. A simple yet accurate way of doing this is via the *Leapfrog integrator*, in which the momentum first receives a "half" update, followed by a full position update and finally a second half update for the momentum:

$$\begin{aligned}
\boldsymbol{v}_{t+1/2} &= \boldsymbol{v}_t - \frac{\eta}{2}\frac{\partial \mathcal{E}(\boldsymbol{\theta}_t)}{\partial \boldsymbol{\theta}} \\
\boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \eta\frac{\partial \mathcal{K}(\boldsymbol{v}_{t+1/2})}{\partial \boldsymbol{v}} \\
\boldsymbol{v}_{t+1} &= \boldsymbol{v}_{t+1/2} - \frac{\eta}{2}\frac{\partial \mathcal{E}(\boldsymbol{\theta}_{t+1})}{\partial \boldsymbol{\theta}}.
\end{aligned}$$

3.3.3 Canonical distributions

In order to sample from a desired target distribution we can relate it to the potential energy function via the concept of a *canonical distribution* originating from statistical mechanics [12]. If we have an energy function for some state of some physical system, the canonical distribution over the states has a density function

$$p(\mathbf{x}) = \frac{1}{Z}\exp(-E(\mathbf{x})/T),$$

where $E(\mathbf{x})$ is the energy for state $\mathbf{x}$, T is the temperature of the system and Z is a normalization constant in order to obtain a proper density function. Since the Hamiltonian is a joint energy function w.r.t. position

and momentum, we obtain a joint distribution:

$$p(\boldsymbol{\theta}, \boldsymbol{v}) = \frac{1}{Z} \exp(-\mathcal{E}(\boldsymbol{\theta})/T) \exp(-\mathcal{K}(\boldsymbol{v})/T).$$

Note that $\boldsymbol{\theta}$ and $\boldsymbol{v}$ are independent with their own canonical distributions. In the MCMC framework the position variables $\boldsymbol{\theta}$ are the variables of interest and the momentum is introduced only to obtain proper Hamiltonian dynamics. The posterior distribution is now expressed as a canonical distribution with $T = 1$ using the potential energy function as follows:

$$\mathcal{E}(\boldsymbol{\theta}) = -\sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x} \mid \boldsymbol{\theta}) - p(\boldsymbol{\theta}),$$

where the first term is the likelihood function and the second one is prior density.

3.3.4 HMC algorithm

We use Metropolis algorithm as a backbone in order for the Hamiltonian dynamics to define an MCMC sampler, i.e., the Hamiltonian dynamics are turned to a proposition for the next state of the Markov chain. In the first step of the HMC algorithm new momenta is sampled independently of the position from their Gaussian distribution: $\boldsymbol{v} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. In the second step, a *Metropolis update* is performed using the Hamiltonian dynamics to propose a new state $(\boldsymbol{\theta}^\star, \boldsymbol{v}^\star)$, from which only the position variable is stored since the momenta will be resampled for each iteration of the algorithm. The Hamiltonian dynamics is simulated for L Leapfrog steps with a stepsize of η as explained above. The parameters L and η need to be *tuned* in order to have a good performance. As in the Metropolis framework, the proposal is then accepted with probability $\min(1, \exp(-\mathcal{H}(\boldsymbol{\theta}^\star, \boldsymbol{v}^\star) + \mathcal{H}(\boldsymbol{\theta}, \boldsymbol{v}))$. In the case of rejection, the new state is set to the previous state just as in the Metropolis algorithm. The outline of the HMC algorithm is given in Algorithm2.

3.4 Stochastic variants of MCMC algorithms

Since the HMC requires a gradient computation w.r.t. the whole dataset, it can be computationally inefficient with large datasets, which are usually present in modern deep learning problems, making the usage of standard HMC questionable. Even a simple MH algorithm requires the MH correc-

Algorithm 2 HMC [1]

Initialize $\boldsymbol{\theta}^0$
for $t = 1, 2, \ldots$ **do**
  Sample momentum $\boldsymbol{v}^{t-1} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$
  $(\boldsymbol{\theta}_0^\star, \boldsymbol{v}_0^\star) \leftarrow (\boldsymbol{\theta}^{t-1}, \boldsymbol{v}^{t-1})$
  $\boldsymbol{v}_{1/2}^\star \leftarrow \boldsymbol{v}_0^\star - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}_0^\star)$ ▷ Half step for the momentum
  for $l = 1 : L - 1$ **do**
    $\boldsymbol{\theta}_l^\star \leftarrow \boldsymbol{\theta}_{l-1}^\star + \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{v}_{l-1/2}^\star$
    $\boldsymbol{v}_{l+1/2}^\star \leftarrow \boldsymbol{v}_{l-1/2}^\star - \eta\nabla\mathcal{E}(\boldsymbol{\theta}_l^\star)$
  end for
  $\boldsymbol{\theta}_L^\star \leftarrow \boldsymbol{\theta}_{L-1}^\star + \eta\boldsymbol{\Sigma}^{-1}\boldsymbol{v}_{L-1/2}^\star$
  $\boldsymbol{v}_L^\star \leftarrow \boldsymbol{v}_{L-1/2}^\star - \frac{\eta}{2}\nabla\mathcal{E}(\boldsymbol{\theta}_L^\star)$ ▷ Half step for the momentum
  $(\boldsymbol{\theta}^\star, \boldsymbol{v}^\star) \leftarrow (\boldsymbol{\theta}_L^\star, \boldsymbol{v}_L^\star)$
  $r \leftarrow \exp(-\mathcal{H}(\boldsymbol{\theta}^\star, \boldsymbol{v}^\star) + \mathcal{H}(\boldsymbol{\theta}^{t-1}, \boldsymbol{v}^{t-1}))$
  $A \leftarrow \min(1, r)$
  Sample $u \sim U(0, 1)$
  if $u \leq A$ **then**
    $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^\star$
  else
    $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1}$
  end if
end for

tion to be computed using the whole dataset, which scales linearly w.r.t. the datasets size, just as the gradient computation in the HMC. In order to alleviate this problem, we must look at *stochastic* variants of the MCMC algorithms, i.e., algorithms that can produce posterior draws only via usage of a subset of the training data.

3.4.1 A general framework for stochastic Markov Chain Monte Carlo algorithms

A *general* framework for constructing *any* stochastic gradient MCMC (SG-MCMC) algorithms was given in [13], where the authors give a *complete* recipe for constructing any continuous Markov process with desired invariant distribution. In order to achieve this, the authors define a stochastic system parameterized by a positive semi-definite diffusion matrix $\mathbf{D}(\mathbf{z})$ and a skew-symmetric curl matrix $\mathbf{Q}(\mathbf{z})$, where $\mathbf{z} = (\boldsymbol{\theta}, \mathbf{r})$ s.t. $\boldsymbol{\theta} \in \mathbb{R}^d$ are the model parameters and $\mathbf{r}$ is a vector of auxiliary variables. For example, in the case of HMC, $\mathbf{z} = (\boldsymbol{\theta}, \boldsymbol{v})$. The dynamics are then written using the target distribution and the two matrices, and by varying the choice of $\mathbf{D}(\boldsymbol{z})$ and $\mathbf{Q}(\mathbf{z})$ the space of MCMC methods maintaining the correct invariant distribution is explored. The general form of SG-MCMC update is then given by:

$$\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1} - \eta_t[\mathbf{D}(\boldsymbol{z}) + \mathbf{Q}(\mathbf{z})]\nabla_{\mathbf{z}}\tilde{H}(\mathbf{z}) + \Gamma(\mathbf{z}) + \sqrt{2\eta_t T}\mathbf{n}, \tag{3.3}$$

where $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{D}(\mathbf{z}))$, $H(\mathbf{z})$ is the energy function of the system, $\nabla_{\mathbf{z}}\tilde{H}(\mathbf{z})$ is an *unbiased estimate* of $\nabla_{\mathbf{z}}H(\mathbf{z})$, η_t is the *learning rate* at step t and $\Gamma_i(\mathbf{z}) \triangleq \sum_{j=1}^{d} \frac{\partial}{\partial z_j}(\mathbf{D}_{ij}(\mathbf{z}) + \mathbf{Q}_{ij}(\mathbf{z}))$. Here T is the temperature of the system similarly as in the case with canonical distribution.

3.4.2 Construction of existing SG-MCMC samplers

One of the first innovations in the SG-MCMC methodology was the *stochastic gradient Langevin Dynamics* (SGLD) [14], which combines *Robbins-Monro* type algorithms with Langevin dynamics such that the trajectory of parameter updates will converge to the full posterior distribution. As this methods builds on the first-order Langevin dynamics, it does not include the *momentum* term of the HMC, which we have seen to be useful in order to make the sampler efficient. In order to combine the efficient sampling of the HMC with stochastic updates, stochastic gradient Hamiltonian Monte Carlo (SG-HMC) was proposed [15]. SG-HMC is based on the *second-order* Langevin dynamics, where a *friction* term is added to the HMC momentum update in order to retain the desired target distribution invariant. These methods, as well as their variants, can be recovered in the framework given by [13].

To give an illustrative example, in the SGLD, parameters are updated as

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \nabla_{\boldsymbol{\theta}} \tilde{\mathcal{E}}(\boldsymbol{\theta}_{t-1}) + \sqrt{2\eta_t}\mathbf{n}, \tag{3.4}$$

where $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\mathcal{E}(\boldsymbol{\theta})$ denotes the *potential energy* of the system, defined identically as in the HMC algorithm, s.t. $\nabla_{\boldsymbol{\theta}}\tilde{\mathcal{E}}(\boldsymbol{\theta})$ is an unbiased estimate of the gradient of the potential energy. By setting $\mathbf{z} = \boldsymbol{\theta}$ s.t. $H(\boldsymbol{\theta}) = \mathcal{E}(\boldsymbol{\theta})$, $\mathbf{D}(\boldsymbol{\theta_t}) = \mathbf{I}$ and $\mathbf{Q}(\boldsymbol{\theta}) = 0$ in (3.3) an identical update rule is recovered. Note that the SGLD update rule does not include the temperature parameter so we may just set $T = 1$. In fact, the temperature plays an important role when sampling from posteriors of deep neural networks [1].

By similar derivation, any SG-MCMC samplers may be either recovered or discovered. The importance of the general framework is that we do not have to give sampler specific proofs; as the framework itself is proven to be *complete*, any sampler arising from it will converge to an invariant

distribution equal to the desired posterior distribution [13].

3.4.3 Adaptive SG-MCMC

A well known fact is that the optimization landscape of a deep neural network exhibits often pathological curvature and saddle points, making the usage of first-order optimization methods inefficient. The same argument can be made for the posterior distributions of a DNN, rendering the simplest SG-MCMC methods unusable. This is likely the case for the GANs as well: even though ProbGAN [16] and BayesGAN [17] stated to use SG-HMC for posterior sampling, they point out that in order to achieve success, the network is first trained with Adam optimizer in order to obtain reasonable MAP estimate as a starting point for the stochastic sampler.

In order for the sampler to escape saddle points and speed up convergence in difficult posterior distributions, [18] proposes the usage of *past* gradients. Namely, the authors of [18] propose two *adaptive* SGLD methods: the momentum SGLD (MSGLD) and the Adam SGLD (ASGLD). A general update rule of an adaptive SGLD algorithm is given by

$$\boldsymbol{\theta}^t = \boldsymbol{\theta}^{t-1} - \eta_t(\nabla_{\boldsymbol{\theta}}\tilde{\mathcal{E}}(\boldsymbol{\theta}) + a\mathbf{a}_t) + \sqrt{2\eta_t T}\mathbf{n}, \tag{3.5}$$

where $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\mathbf{a}_t$ is the adaptive bias term and a is the bias factor.

The MSGLD borrows its idea from the momentum algorithm [19], which guides the direction of new updates by incorporating a fraction of past update directions – the so-called momentum term – to the current gradient. As the momentum term accumulates over iterations, updates to same direction gets increased while updates to the opposite directions gets decreased, reducing the oscillation and accelerating the convergence. In the same spirit MSGLD introduces the momentum term to the SGLD which is calculated as an exponentially decaying average of past stochastic gradients and added as a bias term to the drift of SGLD. The update rule for the MSGLD is obtained by setting $\mathbf{a}_t = \mathbf{m}_t$ in the (3.5). The bias term $\mathbf{m}_t$ is updated by

$$\mathbf{m}_t = \beta_1\mathbf{m}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{\theta}}\tilde{\mathcal{E}}(\boldsymbol{\theta}_{t-1}),$$

where β_1 is a smoothing factor. The outline of MSGLD is given in Algorithm 3.

Algorithm 3 MSGLD[18]

Input: Data $\mathcal{D}$, minibatch size B, smoothing factor $\beta_1 \in (0, 1)$, bias factor a, temperature T and learning rate η
Initialize $\boldsymbol{\theta}^0$, $m_0 \leftarrow 0$
for $t = 1, 2, \ldots$ **do**
  Draw a minibatch of data $\mathcal{B} \subset \mathcal{D},\ |\mathcal{B}| = B$
  Sample $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, 2T\eta\mathbf{I})$
  $\boldsymbol{\theta}^t \leftarrow \boldsymbol{\theta}^{t-1} - \eta_t(\nabla_{\boldsymbol{\theta}}\tilde{\mathcal{E}}(\boldsymbol{\theta}) + am_{t-1}) + \mathbf{n}$
  $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_{\boldsymbol{\theta}}\tilde{\mathcal{E}}(\boldsymbol{\theta}_{t-1})$
end for

As the name suggests, ASGLD derives its idea from the Adam optimization algorithm. The key difference here is that the ASGLD is designed to converge in a distribution rather than a single value. To keep the discussion short, it is worthwhile to note that the authors of [18] experienced the simpler MSGLD to perform on par with the ASGLD, outperforming it in some simulations. Thus we argue that if more complex algorithm does not have a clear performance advantage despite being more computationally expensive, it may be useful to proceed with the simpler one.

3.5 Bayesian Deep Learning

As we have already seen, difference between maximum likelihood and Bayesian approaches can be characterised by the inference process and the explicit introduction of the *prior* distribution. Thus, Bayesian approach to deep learning follows naturally by setting prior distributions to the models parameters inferring the posterior distribution over the parameters via some Bayesian inference method. Here, we consider only the SG-MCMC methods for the posterior computation. In this section, we cover mostly the modelling ideas behind Bayesian neural networks (BNNs), as the posterior computation can be done via any suitable SG-MCMC method presented earlier.

3.5.1 Priors for Bayesian neural networks

The general idea behind the prior distribution is to state our prior *beliefs* about how the model parameters should be distributed. However, for neural networks, this approach does not have an intuitive basis. A common approach of prior selection in the context of BNNs is to use *Gaussian* priors. For a BNN with $L - 1$ layers we set a Gaussian prior for the weights and biases independently for each layer as follows:

$$\mathbf{W}_l \sim \mathcal{N}(\mathbf{0}, \alpha_l^2 \mathbf{I}),\ \mathbf{b}_l \sim \mathcal{N}(\mathbf{0}, \beta_l^2 \mathbf{I}),\ l = 1, \ldots, L. \tag{3.6}$$

The selection of α and β may be done for example via *prior predictive checking*, but some standard choices are *Xavier/Glorot initialization* or *LeCun initialization*, defined respectively as

$$\alpha^2 = \frac{2}{n_{\text{in}} + n_{\text{out}}} \text{ and } \alpha^2 = \frac{1}{n_{\text{in}}},$$

where n_{in} denotes the number of weights coming into neuron at level l, and n_{out} the number of weights flowing out of the neuron [3].

Even though the factored Gaussian is the most common choice for prior in the context of BNNs, it is not the only suitable choice. As we have seen, prior distributions can be seen as Bayesian counterpart of the *regularization* in maximum likelihood paradigm, and therefore, it is natural to consider such prior distributions that promote *sparsity* in the model. A common choice for such prior is the *Laplace distribution* [1]. Another way of dealing with the difficulty of selecting the prior is to *learn* the prior from the data, which can be useful mechanism for both *Bayesian transfer learning* and *multitask learning*.

3.5.2 Priors in function space and architectural priors

In the context of neural networks, prior distributions set on the model parameters are difficult to be understood in the parameter space. To gain better intuition, it is often helpful to study the priors in the *function space*. For example, if we would have a multilayer perceptron (MLP) for a regression problem with a structure $1 - n - 1$ and we were to set Gaussian prior with zero mean and standard deviation β_l independently for each l layer we can study the effects of choosing different sets of $\{\beta_l\}_{l=1}^{L}$, $L = 2$, with some fixed n. Draws from the prior for different selection of $\beta_1 \times \beta_2$ can be seen in Figure 3.2. In the example, the width of the network was set as $n = 20$ and hyperbolic tangent was used as the activation function.

It is clear that the higher variance in the prior of the weights lets the functions have more wiggly shape due to the increased sensitivity to the change of the input value. Note that the prior draws from the network are in a sense *smooth functions*, which follows directly from the activation function used. It should be stated that the choice of prior does not set the

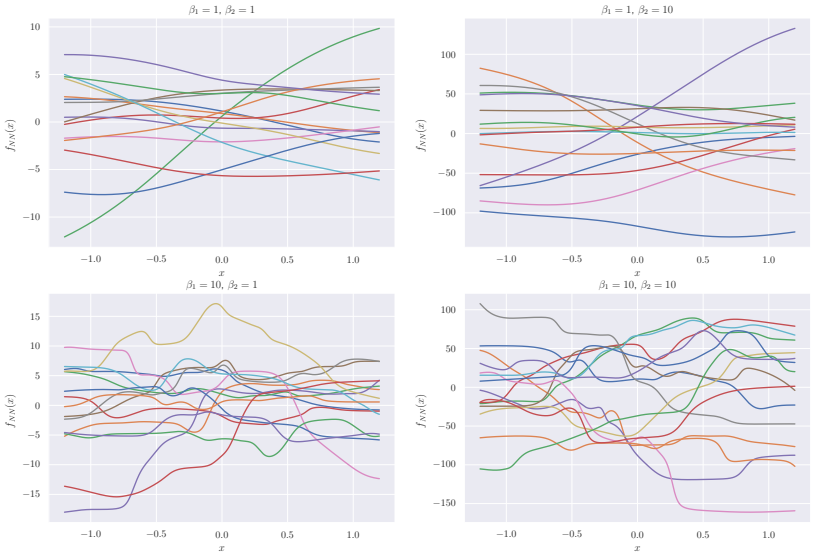

Fig. 3.2. Effects of prior distribution on the functions produced by an MLP.

hypothesis space of the neural network strictly. It merely favors hypotheses closer to those produced by the prior, as the posterior distribution is always a compromise between the data and the prior, converging to the MLE when the number of datapoints goes to infinity. Thus, the choice of prior is of more importance in applications with limited amounts of data.

In addition to the choice of prior distribution laid on the weights of the model parameters, the choice of activation functions and structure of the neural network has similar effect on the hypotheses the model favours. These architectural considerations are sometimes called *architectural priors* [1]. It has been established in the field of *computer vision* that even with random weights a neural network can be a powerful feature extractor, given a proper architecture of the network.

3.5.3 Hierarchical priors

If we wish not to state any strong prior beliefs about the distribution of the models weights, we can set a functional form of the prior distribution, set *hyperpriors* for the prior distribution parameters and infer the posterior distribution for the hyperparameters from the data. Consider for example a prior $\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\alpha}, \beta^2\mathbf{I})$. In order to learn the hyperparameters $\boldsymbol{\alpha} \in \mathbb{R}^p$ and $\beta \in \mathbb{R}$ from the data, we may set *weakly informative* hyperpriors for those parameters and infer their posterior distribution. We may for example set

$$\alpha_p \sim \mathcal{N}(0, 10^2),\ \beta \sim \mathbf{Exp}(0.5),$$

and infer the posterior distribution of the hyperparameters. Even though at glance counterintuitive approach, as we should state prior beliefs *before* seeing the data, it becomes evidently useful when we can *learn* the prior distribution from different, but similar datasets. This approach is sometimes called a *Bayesian transfer learning* [1]. Another perspective for hierarchical modelling arises from the approach of sharing information between multiple models with common hyperparameters: via hierarchical structure the hyperparameters may be inferred from multiple related, but usually limited datasets, allowing to learn a stronger prior which then guides the learning process of the individual models. This may be seen as a Bayesian approach for *multitask learning*.

3.5.4 Multitask learning via Bayesian hierarchical modelling

In multitask learning the goal is to learn a model from multiple datasets so that the model learns the similarities between different data distributions while allowing the model to excel in the individual tasks. This is usually realized by sharing some of the model parameters while leaving some other parameters to be task-specific [1]. Our approach to realize such model is through hierarchical Bayesian modelling. For an illustrative example, consider a simple regression problem with $M = 3$ related datasets $\{\mathcal{D}_i\}_{i=1}^{M}$, each having relatively small number of datapoints, $|\mathcal{D}_i| = N_i = 32,\ i \in \{1, \ldots, M\}$. The data (x_i, y_i) are generated as $y = f(x)$ where $f(x) = x^3$, and x_i are covariates between -1 and 1. To obtain three related datasets, the generated observations are consequently rotated by 15 and 30 degrees, so that the first dataset contains the original observations, second contains original observations rotated by 15 degrees and third one contains the original observations rotated by 30 degrees. Finally, observation noise $\epsilon \sim \mathcal{N}(0, 0.1^2)$ is injected to each of the three datasets. The generated data are illustrated in Figure 3.3.

Our goal is to learn a function that performs regression task well on all three datasets. Since the data is limited in volume, training a neural network individually for each of the datasets will be prone to overfitting. Pooling the data could help to increase the training data volume, but yield

Fig. 3.3. The datasets share a similar shape but are coming from different distributions.

a model that is not suited for any of the regression tasks. Instead, we are applying the idea of Bayesian hierarchical model to learn the similarities shared between the datasets while having individual parameters for each of the tasks.

The model of choice for this multitask learning problem will be hierarchical Bayesian neural network, which we denote as $f_{NN}(x, \boldsymbol{\theta}_i)$, where i denotes for which task we are mapping the covariates x. We assume that the underlying target variable $y_i = f_{NN}(x, \boldsymbol{\theta}_i) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$, i.e., $y_i \sim \mathcal{N}(f_{NN}(x, \theta_i), \sigma^2)$. For the sake of simplicity, we assume that we know the variance of the underlying noise process, $\sigma = 0.1$. Since we are using a simple neural network as the model, for notational convenience we divide the parameters for each layer separately, so that $\boldsymbol{\theta}_i^{(l)}$ denotes the parameters of the models' l-th layer. For each of the model layer weights, we place a Gaussian prior: $\boldsymbol{\theta}_i^{(l)} \sim \mathcal{N}(\boldsymbol{\mu}^{(l)}, s^{2(l)}\boldsymbol{I})$, and for the priors hyperparameters, we place weakly informative priors: $\boldsymbol{\mu}^{(l)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, $s^{(l)} \sim \text{Exp}(1)$.

Note that it becomes hard for the sampler to sample from $\mathcal{N}(\mu, \sigma^2)$ when the standard deviation σ becomes small. This problem is known as the *Neals funnel*. For this reason, we use the *non-centered parameterization* for the model weights $\boldsymbol{\theta} = \boldsymbol{\mu} + \hat{\boldsymbol{\theta}}s$ s.t. $\hat{\boldsymbol{\theta}} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $s \sim \text{Exp}(1)$. The

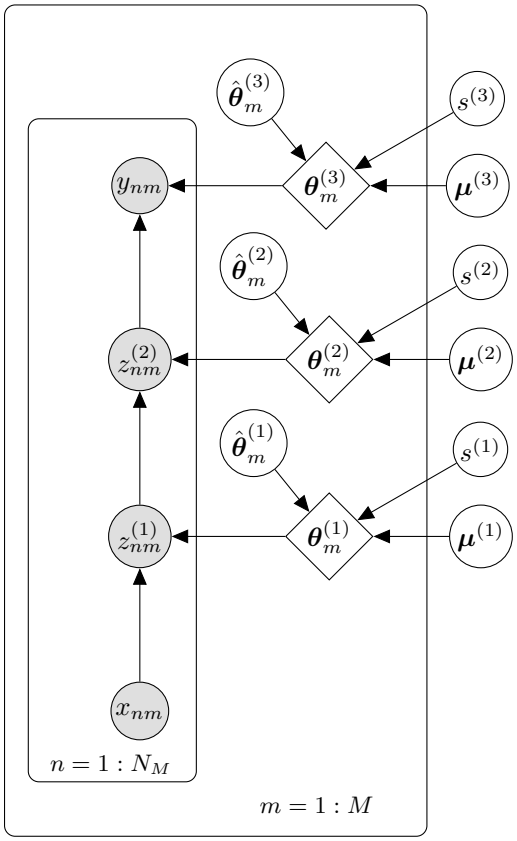

Fig. 3.4. Graphical illustration of the structure of hierarchical Bayesian neural network for M different tasks.

reparameterization scheme is illustrated in Figure 3.4, where we have used triangular boxes to underline the fact that with this parameterization scheme the model weights are indeed *deterministic* product of their parents. Since we have now defined the model likelihood function and the prior densities, the posterior is obtained via Bayes' formula:

$$p(\hat{\boldsymbol{\theta}}_{1:M}^{1:L}, \boldsymbol{\mu}^{(1:L)}, s^{(1:L)}) \mid \mathcal{D}_{1:M}) \propto p(\mathcal{D}_{1:M} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \hat{\boldsymbol{\theta}}, \boldsymbol{\mu}, \boldsymbol{s}) p(\hat{\boldsymbol{\theta}}) p(\boldsymbol{s}) p(\boldsymbol{\mu}). \quad (3.7)$$

Even with this very limited amount of data we can easily perform the posterior computation with the HMC. However, it has been stated that the HMC may not be the best option with hierarchical models [12]. Thus, to make the most of this illustration, we use the MSGLD for the posterior computation. For the MSGLD, we set hyperparameters $\beta_1 = 0.99$, $T = 1$, $a = 10$ and $\eta = 10^{-4}$. For each minibatch, we randomly select half of the training data, $|\mathcal{B}| = 16$. We sample from the posterior distribution for 10,000 iterations and discard the first half of the draws. Since the neural networks are *unidentifiable*, using any statistics for the raw posterior draws of the parameters is not sensible [20]. Instead, we use the

posteriors draws for *predictions* and take the 50th percentile to measure the location of the prediction and illustrate the uncertainty with 5th and 95th percentiles of the predictions, i.e., 90% of the predictions lay in the range of illustrated uncertainty as shown in Figure 3.5.

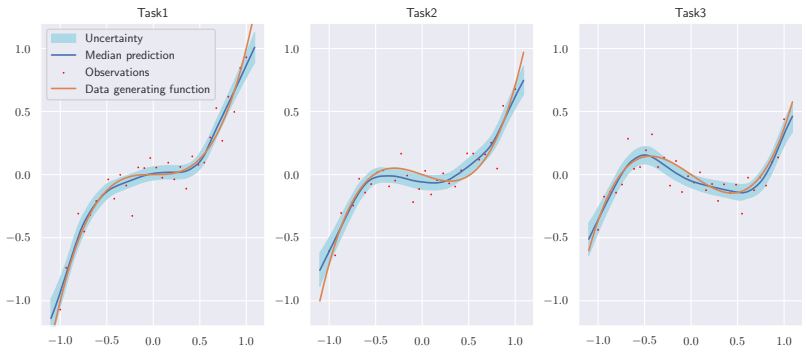

Fig. 3.5. Hierarchical structure helps the model to learn the similar shape of the regression problem without overfitting even when only small number of datapoints are available.

From Figure 3.5 we can see that the hierarchical BNN gives reasonable generalization w.r.t. the very limited number of datapoints. The model does not overfit the noisy data and generally learns the shape of the regression tasks well. Note that Figure 3.5 illustrates only the uncertainty w.r.t. the model parameters, and it does not reflect on the observation noise, which we assumed to be normally distributed with standard deviation of 0.1. For any practical scenario the observation noise should be learned and reflected in the posterior predictions. In the hierarchical modelling paradigm the learnable noise could be coming from common prior for all datasets, but if we are unsure if the assumption is correct, we could easily learn the observation noise for each task as an independent parameter. This would make sense if we would have measurements generated by similar measure instruments from different manufacturers, or with different physical conditions.

For this example, we used three limited datasets with equal number of datapoints. However, the hierarchical modelling can be used with great imbalance of datapoints per task without any modifications to the modelling or inference processes. In practical setting imbalances between datasets are present often, making the hierarchical modelling a great tool for learning strong prior knowledge about some data generating process, which can then easily be utilised in similar task with only small number of

available data.

4. Deep Generative Modeling

The objective in generative modelling is to model the data distribution, $p(\mathbf{x})$ for $\mathbf{x} \in \mathcal{X}$. Furthermore, for our use case, the model must be able to produce new, high fidelity samples $\mathbf{x} \sim p(\mathbf{x})$ relatively fast. Autoregressive models, normalizing flows, diffusion models and energy based models do take relatively large amount of time to produce new samples, and are therefore not considered. Both variational autoencoders (VAEs) and generative adversarial networks (GANs) can produce samples rapidly, however VAEs often produce inferior quality samples in terms of fidelity. Furthermore, GANs have been used succesfully in the channel modeling literature and thus we limit the scope of this thesis to the usage of GANs.

4.1 Divergence metrics for training a generative model

As our goal is to generate samples from data distribution $\mathcal{P}$, we need to define some measures for divergences between two distributions. For our case, we need a measure between the true data distribution $\mathcal{P}$ and the generated data distribution $\mathcal{Q}$, both defined in the same space. Two commonly used ways of comparing two distributions are their ratio, $\frac{\mathcal{P}}{\mathcal{Q}}$, and their difference $\mathcal{P} - \mathcal{Q}$ [1]. Ideally, we would have a *divergence* metric $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$ that is computationally efficient and can be evaluated only through the samples from the respective distributions, for reasons that will become clear when we define our generative modelling procedure. Note that we are not limiting our interest only to *distances*; i.e., we do not require D to be symmetric.

4.1.1 f-divergence

f-divergence provides us a way of quantifying the similarity between two distributions in terms of their ratio:

$$D_f(p||q) = \int q(x) f\left(\frac{p(x)}{q(x)}\right) dx,$$

where $f : \mathbb{R}^+ \to \mathbb{R}$ is a *convex* function satisfying $f(1) = 0$ [1]. It follows from Jensen's inequality that $D_f(p||q) \geq 0$. Obviously, it also holds that $D_f(p||p) = 0$ and thus D_f is a valid divergence. The choice of f plays an important role for the properties of the divergence. In fact, if we set $f(r) \triangleq r \log r$, we obtain *Kullback Leibler* (KL) divergence [21], which plays a major part in *variational* Bayesian methods. For our purposes the KL divergence has a downside: it requires *analytical* distributions in order to be evaluated. In addition, if our objective is to *learn* a generative procedure that minimizes the KL divergence between the true data distribution and the generated one, then cases where q has zero density in the support of p will lead to KL divergence equal to zero, giving us no gradient information for the learning procedure. Fortunately, we will see both of these downsides vanish when using neural approximation of the KL divergence. With different definitions of f we can recover for example alpha divergences, Hellinger distance and χ^2 distance.

4.1.2 Integral probability metrics

As stated before, another way of measuring similarity between distributions is via their difference, $P - Q$, which can be computed by integral probability metrics (IPMs), defined as follows [1]:

$$D_{\mathcal{F}}(P, Q) \triangleq \sup_{f \in \mathcal{F}} |\mathbb{E}_{p(\mathbf{x})}[f(\mathbf{x})] - \mathbb{E}_{q(\mathbf{y})}[f(\mathbf{y})]|. \tag{4.1}$$

The set $\mathcal{F}$ is some class of *smooth* functions, and the function f maximizing the difference between the expectations is called a *witness* function. We can, for example, define $\mathcal{F}$ to be a set of functions that have bounded Lipschitz constant, $\mathcal{F} = \{\| f \|_L \leq 1\}$, where $\| \cdot \|_L$ is the Lipschitz norm, defined as follows:

$$\| f \|_L = \sup_{\mathbf{x} \neq \mathbf{x}'} \frac{|f(\mathbf{x}) - f(\mathbf{x}')|}{\|\mathbf{x} - \mathbf{x}'\|_2}.$$

This formulation yields an IPM (4.1) to be equivalent to *Wasserstein-1* distance:

$$W_1(P, Q) \triangleq \sup_{\|f\|_L \leq 1} |\mathbb{E}_{p(x)}[f(x)] - \mathbb{E}_{q(y)}[f(y)]|. \tag{4.2}$$

In the following sections we will see that this metric has favorable proper-

ties for GAN training.

4.1.3 Density ratio estimation via binary classification

Consider a binary classification problem with a classifier $f : \mathbb{R}^d \to \{0, 1\}$ s.t. $\mathbf{x} \in \mathbb{R}^d \sim P \Rightarrow f(\mathbf{x}) = 1$ and $\mathbf{x} \in \mathbb{R}^d \sim Q \Rightarrow f(\mathbf{x}) = 0$. Now $P(\mathbf{x}) = p(\mathbf{x} \mid f(\mathbf{x}) = 1)$ and $Q(\mathbf{x}) = p(\mathbf{x} \mid f(\mathbf{x}) = 0)$. Further, we define the class priors as $\pi_1 = p(f(\mathbf{x}) = 1)$ and $\pi_0 = p(f(\mathbf{x}) = 0)$. Via the Bayes' rule, we obtain the density ratio $r(\mathbf{x}) = \dfrac{P(\mathbf{x})}{Q(\mathbf{x})}$ as:

$$r(\mathbf{x}) = \frac{p(\mathbf{x} \mid f(\mathbf{x}) = 1)}{p(\mathbf{x} \mid f(\mathbf{x}) = 0)} = \frac{p(f(\mathbf{x}) = 1 \mid \mathbf{x})}{p(f(\mathbf{x}) = 0 \mid \mathbf{x})} \frac{\pi_0}{\pi_1}.$$

If we let the class prior $\pi_1 = 0.5 \Rightarrow \pi_0 = 1 - \pi_1$, which is a reasonable assumption when there does not exist a class imbalance, we can estimate the ratio $r(\mathbf{x})$ with a binary classifier $f(\mathbf{x})$ and compute $r = f/(1-f)$, which is called the density ratio estimation trick [1]. If we optimize the classifier f by minimizing the empirical risk, depending on the loss function used, we can recover *any* f-divergence. Furthermore, there exists a connection between the binary classification and IPMs.

4.2 Metrics for generative model evaluation

The literature on measuring performance of generative modelling revolves around the most common use cases for generative modelling, i.e., image and text generation. For images, a commonly used metric is the Fréchet Inception Distance (FID) [22], computation of which is based on high-dimensional features extracted from the generated images using some pretrained image classification neural network. However, for the channel modelling there does not exist any commonly used classifier to extract high-dimensional features from generated channel instances. Thus, to compare the generated channel instances, we resort to manual labour in inspecting the quality of the generated channel instances, and first- and second order statistics on checking the distributional coverage of the generative model.

4.3 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are *implicit generative models*, i.e., they lack an explicit likelihood function [1]. In contrast to *prescribed*

generative models, which provide an explicit parametric likelihood function $p(\mathbf{x} \mid \boldsymbol{\theta})$ for observed random variable $\mathbf{x}$, implicit generative models define stochastic procedure to directly generate data. In mathematical terms, implicit generative models define a deterministic parametric mapping from latent space to the data space $G_{\boldsymbol{\theta}} : \mathbb{R}^m \to \mathbb{R}^d$. In the case of GANs, the function $G_{\boldsymbol{\theta}}$ is a nonlinear mapping with $d > m$ specified by deep neural network. Furthermore, the latent space of the generator is usually taken to be defined by isotropic standard normal distribution. To retain generality, we denote the samples of the latent space with $\mathbf{z} \sim q(\mathbf{z})$. Since the GANs lack the likelihood term, the learning problem is called *likelihood-free inference*, and its solution relies on a comparing real and simulated data. For our mathematical treatment of GANs we will mostly rely on the presentation given in [1].

4.3.1 Learning by comparison

In order to generate realistic data, we must learn a generative model $q_{\boldsymbol{\theta}}$ that minimizes the KL-divergence between the unknown true data distribution, $p^\star$, which effectively corresponds to maximizing the likelihood under seen true data. Since GANs are implicit models, we do not have the likelihood term, and thus have to learn by comparing the true and generated data. Therefore, we are looking for an objective $\mathcal{D}(p^\star, q_{\boldsymbol{\theta}})$ that can be computationally effectively evaluated using only samples of data, and provides guarantees about learning the true data distribution. The problem then becomes:

$$\operatorname*{argmin}_{q_{\boldsymbol{\theta}}} \mathcal{D}(p^\star, q_{\boldsymbol{\theta}}) = p^\star.$$

Since many of the distributional distances and divergences are either computationally intractable or cannot be evaluated by using samples, we will *learn* a model D that will act as *critic* or *discriminator* s.t. $\mathcal{D}(p^\star, q_{\boldsymbol{\theta}}) = \operatorname*{argmax}_D \mathcal{F}(D, q_{\boldsymbol{\theta}}, p^\star)$, where $\mathcal{F}$ is a functional that depends on the $p^\star$ and $q_{\boldsymbol{\theta}}$ only through samples. Similarly to the generator $G_{\boldsymbol{\theta}}$, the discriminator is a nonlinear mapping $D_{\boldsymbol{\phi}} : \mathbb{R}^d \to [0, 1]$, or, in some cases $D_{\boldsymbol{\phi}} : \mathbb{R}^d \to \mathbb{R}$, realized by a deep neural network.

Intuitively, we can view GANs as two neural networks where the generator tries to fool the discriminator thinking that the generated data is coming

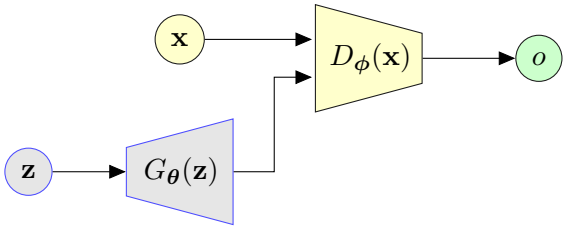

Fig. 4.1. Structure of the GAN. The generator $G_{\boldsymbol{\theta}}$ aims map samples from its latent space to data similar to the true data generating distribution, whereas the discriminator $D_{\boldsymbol{\phi}}$ tries to distinguish between the samples produced by the generator and samples coming from the true data generating distribution.

from the true data distribution, and the discriminator tries to learn to classify data presented to it as real or fake. The general structure of GAN is illustrated in Figure 4.1. Next, we will establish more analytical view on how the data are actually compared using the divergence metrics presented in Section 4.1.

4.3.2 Learning via density ratio estimation

As we stated earlier, we can convert the density ratio estimation into a binary classification problem. In the case of classifier realized by a neural network D, using the density estimation trick presented in Subsection 4.1.3, the density ratio estimation becomes:

$$\frac{p^{\star}(\mathbf{x})}{q_{\boldsymbol{\theta}}(\mathbf{x})} = \frac{D_{\boldsymbol{\phi}}(\mathbf{x})}{1 - D_{\boldsymbol{\phi}}(\mathbf{x})}.$$

We learn the discriminator $D_{\boldsymbol{\phi}}$ parameters $\boldsymbol{\phi}$ by minimizing some loss function. If we use for example Bernoulli log-loss, the objective function becomes:

$$V_{\boldsymbol{\phi}}(q_{\boldsymbol{\theta}}, p^{\star}) = \frac{1}{2}\mathbb{E}_{p^{\star}(\mathbf{x})} \log D_{\boldsymbol{\phi}}(\mathbf{x}) + \frac{1}{2}\mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} \log(1 - D_{\boldsymbol{\phi}}(\mathbf{x})). \tag{4.3}$$

From the definition of density ratio estimation using binary classification, we obtain the optimal classifier as:

$$\frac{p^{\star}(\mathbf{x})}{q_{\boldsymbol{\theta}}(\mathbf{x})} = \frac{D^{\star}(\mathbf{x})}{1 - D^{\star}(\mathbf{x})} \Leftrightarrow D^{\star}(\mathbf{x}) = \frac{p^{\star}(\mathbf{x})}{p^{\star}(\mathbf{x}) + q_{\boldsymbol{\theta}}(\mathbf{x})}.$$

If we plug the optimal classifier into (4.3), the objective boils down to

minimization of *Jensen-Shannon divergence* (JSD):

$$\begin{aligned} V^\star(q_\theta, p^\star) &= \frac{1}{2}\mathbb{E}_{p^\star(\mathbf{x})} \log \frac{p^\star(\mathbf{x})}{p^\star(\mathbf{x}) + q_{\boldsymbol{\theta}}(\mathbf{x})} + \frac{1}{2}\mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} \log(1 - \frac{p^\star(\mathbf{x})}{p^\star(\mathbf{x}) + q_{\boldsymbol{\theta}}(\mathbf{x})}) \\ &= \frac{1}{2} D_{\mathbb{KL}}\left(p^\star \middle|\middle| \frac{p^\star + q_{\boldsymbol{\theta}}}{2}\right) + \frac{1}{2} D_{\mathbb{KL}}\left(q_{\boldsymbol{\theta}} \middle|\middle| \frac{p^\star + q_{\boldsymbol{\theta}}}{2}\right) - \log 2 \\ &= \mathrm{JSD}(p^\star, q_{\boldsymbol{\theta}}) - \log 2 \end{aligned} \tag{4.4}$$

Recall that our goal is to learn a generator $G_{\boldsymbol{\theta}}$ that minimizes the density ratio between true and generated data distributions when we do not have access to the optimal classifier, but we are learning a neural approximation $D_{\boldsymbol{\phi}}$ of it. All of this combined results in a min-max optimization problem:

$$\min_{\theta} \max_{\phi} \frac{1}{2}\mathbb{E}_{p^\star(\mathbf{x})}[\log D_\phi(\mathbf{x})] + \frac{1}{2}\mathbb{E}_{q(\mathbf{z})}[\log(1 - D_\phi(G_\theta(\mathbf{z})))]. \tag{4.5}$$

The resulting objective (4.5) is equivalent to the definition given by Goodfellow et al. in [23]. Note that the resulting objective is obtained when we use Bernoulli loss for the discriminative model that tries to approximate the density ratio between the true and generated data distribution. If we use for example Brier loss we are minimizing Pearson χ^2 divergence, which leads to Least Squares GAN (LS-GAN) [1]. Via different loss functions for the discriminator, we may obtain a wide variety of different GAN objectives, under the assumption of optimal classifier, which is rarely accurate in practical scenarios since it would require that the discriminator capacity is infinite.

4.3.3 Learning via integral probability metrics

Instead of comparing the ratio of distributions, we may also compare their difference. The general class of IPMs, introduced in subsection 4.1.2 are given as:

$$I_{\mathcal{F}}(p^\star(\mathbf{x}), q_{\boldsymbol{\theta}}(\mathbf{x})) = \sup_{f \in \mathcal{F}} |\mathbb{E}_{p^\star(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} f(\mathbf{x})|.$$

If we let $\mathcal{F}$ to be set of 1-Lipschitz functions, the obtained IPM corresponds to the *Wasserstein distance:*

$$W_1(p^\star(\mathbf{x}), q_{\boldsymbol{\theta}}(\mathbf{x})) = \sup_{f : \|f\|_{\mathrm{Lip} \leq 1}} \mathbb{E}_{p^\star(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} f(\mathbf{x}).$$

Since the supremum over the set of 1-Lipschitz functions is usually intractable, we again have to approximate the divergence with a neural network, $D_{\boldsymbol{\phi}}$:

$$
\begin{aligned}
W_1(p^\star(\mathbf{x}), q_\theta(\mathbf{x})) &= \sup_{f:\|f\|_{\text{Lip}\leq 1}} \mathbb{E}_{p^\star(\mathbf{x})} f(\mathbf{x}) - \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} f(\mathbf{x}) \\
&\geq \max_{\boldsymbol{\phi}:\|D_{\boldsymbol{\phi}}\|_{\text{Lip}\leq 1}} \mathbb{E}_{p^\star(\mathbf{x})} D_{\boldsymbol{\phi}}(\mathbf{x}) - \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} D_{\boldsymbol{\phi}}(\mathbf{x}).
\end{aligned} \tag{4.6}
$$

Note that equation (4.6) assumes that the weights ϕ of the discriminator are such that the Lipschitz constraint is satisfied, i.e., we have to make sure to regularize $D_{\boldsymbol{\phi}}$ to be 1-Lipschitz. Now to train the generative model, we must learn a generator $G_{\boldsymbol{\theta}}$ that minimizes the learned divergence:

$$
\begin{aligned}
\min_{\boldsymbol{\theta}} W_1(p^\star(\mathbf{x}), q_{\boldsymbol{\theta}}(\mathbf{x})) &\geq \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}:\|D_{\boldsymbol{\phi}}\|_{\text{Lip}\leq 1}} \mathbb{E}_{p^\star(\mathbf{x})} D_{\boldsymbol{\phi}}(\mathbf{x}) - \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} D_{\boldsymbol{\phi}}(\mathbf{x}) \\
&= \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}:\|D_{\boldsymbol{\phi}}\|_{\text{Lip}\leq 1}} \mathbb{E}_{p^\star(\mathbf{x})} D_{\boldsymbol{\phi}}(\mathbf{x}) - \mathbb{E}_{q(\mathbf{z})} D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z})).
\end{aligned} \tag{4.7}
$$

This formulation recovers the widely used *Wasserstein GAN* (WGAN) [24]. The authors of [24] did not touch upon any specialized techniques for enforcing the 1-Lipschitz constraint; instead they only forced the discriminator networks parameters to lie in a compact space by *clipping* the weights – which they stated to be a *terrible* practice, but good enough to provide reasonable performance.

In [25] it was further stated that clipping the weights in order to enforce the 1-Lipschitz constraint can indeed lead to undesirable performance: low quality samples produced by the generator or even complete failure to converge. They provided a new way of enforcing the Lipschitz constraint by introducing a gradient penalty (GP) term, which they stated helped to avoid the problems in the GAN training. The GP term is added to the original WGAN objective and weighted by hyperparameter λ :

$$
\min_{\theta} \max_{\phi:\|D_\phi\|_{\text{Lip}\leq 1}} \mathbb{E}_{q(\mathbf{z})} D_\phi(G_\theta(\mathbf{z})) - \mathbb{E}_{p^\star(\mathbf{x})} D_\phi(\mathbf{x}) + \lambda \mathbb{E}_{\hat{p}(\hat{\mathbf{x}})}[(\|\nabla_{\hat{\mathbf{x}}} D_\phi(\hat{\mathbf{x}})\|_2 - 1)^2], \tag{4.8}
$$

where the samples $\hat{\mathbf{x}}$ are generated by a distribution that is a random linear combination of the true data distribution and the distribution implicitly prescribed by the generative model:

$$
\begin{aligned}
\hat{p} &= \alpha p^\star + (1-\alpha) q_\theta \\
\alpha &\sim U(0,1).
\end{aligned}
$$

In a practical scenario, $\hat{\mathbf{x}}$ is a randomly weighted linear combination of $\mathbf{x}$

and $G_{\boldsymbol{\theta}}(\mathbf{z})$.

4.3.4 Theoretical problems with f-divergences

As we have already seen, f-divergences are problematic for training generative models when the distribution induced by the generative model has zero density under the support of the true data distribution, i.e., KL divergence between the distributions equals to infinity or $\log 2$ for the JSD, providing no useful gradient for training the generative model. However, we are not using analytical form of any of the f-divergences but only a neural approximation. This will remedy the problem since the approximation will usually be smoother than the analytical form of f-divergence.

4.4 Training GANs

We have now shown how GANs arise from likelihood-free inference with neural discriminator trained to estimate divergence between real and generated distributions. We have formulated the minimization problem, but not touched upon how to *train* the networks in order to find a solution for the optimization problem at hand. First of all, we have formulated the GANs with zero-sum objective, however, that needs not to be the case. More generally, we may write the objectives for the discriminator and generator, respectively, as:

$$\max_{\boldsymbol{\phi}} L_D(\boldsymbol{\phi}, \boldsymbol{\theta}); \quad \max_{\boldsymbol{\theta}} L_G(\boldsymbol{\phi}, \boldsymbol{\theta}). \tag{4.9}$$

The majority of typical GAN objectives can now be written as

$$\begin{aligned} L_D(\boldsymbol{\phi}, \boldsymbol{\theta}) = &\quad \mathbb{E}_{p^\star(\mathbf{x})} g(D_\phi(\mathbf{x})) + \mathbb{E}_{q(\mathbf{z})} h(D_\phi(G_\theta(\mathbf{z}))) \\ L_G(\phi, \theta) = &\quad \mathbb{E}_{q(\mathbf{z})} l(D_\phi(G_\theta(\mathbf{z}))), \end{aligned} \tag{4.10}$$

with g, h, $l : \mathbb{R} \to \mathbb{R}$. With certain choices of g, h and l we can recover a wide variety of different GAN formulations [1].

4.4.1 Gradient based learning

In order to learn both discriminator and generator, which are deep neural networks, the usual convention is to calculate the gradients of the loss functions and *backpropagate* the loss through the networks. For the discriminator we have:

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} L_D(\boldsymbol{\phi}, \boldsymbol{\theta}) &= \nabla_{\boldsymbol{\phi}}[\mathbb{E}_{p^\star(\mathbf{x})} g(D_{\boldsymbol{\phi}}(\mathbf{x})) + \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} h(D_{\boldsymbol{\phi}}(\mathbf{x}))] \\ &= \mathbb{E}_{p^\star(\mathbf{x})} \nabla_{\boldsymbol{\phi}} g(D_{\boldsymbol{\phi}}(\mathbf{x})) + \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} \nabla_{\boldsymbol{\phi}} h(D_{\boldsymbol{\phi}}(\mathbf{x}))) \\ &\approx \frac{1}{M} \sum_{m=1}^{M} [\nabla_{\boldsymbol{\phi}} g(D_{\boldsymbol{\phi}}(\mathbf{x}_m)) + \nabla_{\boldsymbol{\phi}} h(D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z}_m))],\end{aligned} \tag{4.11}$$

where $\nabla_{\boldsymbol{\phi}} g(D_{\boldsymbol{\phi}}(\mathbf{x}))$ and $\nabla_{\boldsymbol{\phi}} h(D_{\boldsymbol{\phi}}(\mathbf{x})))$ can be computed via backpropagation and the expectation itself is approximated via *Monte Carlo integration*. As for the generator, the gradient of the loss is obtained as:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} L_G(\boldsymbol{\phi}, \boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{x})} l(D_{\boldsymbol{\phi}}(\mathbf{x})) \\ &= \mathbb{E}_{q(\mathbf{z})} \nabla_{\boldsymbol{\theta}} l(D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z}))) \\ &\approx \frac{1}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}} l(D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z}_m))).\end{aligned} \tag{4.12}$$

Here we again used the *reparameterization trick* introduced in subsection 4.1.3. Specifically, since $q_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z})$ is deterministically induced by our generative model, we can draw samples from it by first sampling from $q(\mathbf{z})$ and then transforming them via our generative model. This allows us to change the order of integration, since $q_{\boldsymbol{\theta}}(\mathbf{x})$ depends on the differentiation parameter $\boldsymbol{\theta}$, and $q(\mathbf{z})$ does not.

We have defined a general framework for updating both discriminator and generator, however, we have not discussed on *how* the training should precisely be done. Recall that the theoretical viewpoint of learning the true generating distribution relies on *optimal* discriminator. However, fully solving the optimization problem $\min_{\boldsymbol{\phi}} L_D(\boldsymbol{\phi}, \boldsymbol{\theta})$ is not computationally feasible. The general way of approximating this in practice is to perform first K updates for the discriminator and then update the generator once.

4.5 Instabilities in GAN training

GANs are notoriously difficult to train due to their adversarial nature; most notably they suffer from both *mode collapse* and *mode hopping* [1]. Mode collapse refers to a behavior when the generator stops producing data from one of the modes of the data distribution, or even worse, starts only producing variants of handful of training examples. Mode hopping exhibits

similar kind of behaviour; at some point of training the generator produces data from one of the modes of the data distribution and later moves to another mode. Note that these behaviours have been a problem since the invention of GANs and there exists now a plethora of work focusing on overcoming these problems, which include different loss functions, such as the usage of Wasserstein-1 distance, network level regularization, i.e., using dropout layers, different optimization algorithms and other modifications, which we shall discuss more in detail later.

4.5.1 Better gradient information

Before diving deeper in the refinements for GANs proposed in the literature, we first take a look to a simple improvement proposed in the original GAN paper. Usage of Bernoulli log-loss leads to the JSD, as discussed, and can be interpreted so that the generator tries to *minimize* the probability that the discriminator labels the sample that it produces as fake. Albeit theoretically sound objective, it may not provide good gradient information for the generator to learn. Instead, Goodfellow et al. [23] propose to use *nonsaturating loss*, in which the generator aims to *maximize* the probability for the discriminator to label the sample it produces as real:

$$\min_{\theta} \mathbb{E}_{q_\theta(\mathbf{x})} - \log(D_\phi(\mathbf{x})). \tag{4.13}$$

Figure 4.2 illustrates the difference between the original and nonsaturating objectives: the original objective offers no gradient information when the generator is performing poorly, whereas the nonsaturating loss will help the generator to actually learn in such cases.

Wasserstein GAN

Even though the nonsaturating loss offers favorable properties for training GANs, in practice it still suffers from unstable training [26]. The already discussed Wasserstein GAN (WGAN) is shown to be generally much more stable to train, which is mainly due to the better gradient properties of the loss function. However, even though the original authors of WGAN [24] stated that none of their experiments encountered the mode collapse problem, further investigation done in [25] showed that WGAN implemented via the discriminator weights clipping can perform poorly when the depth of the networks are increased. This is due to the fact that the weight clipping in practice will cause the weights to tend towards either maximum of their clipping range, wasting the representative power of the network.

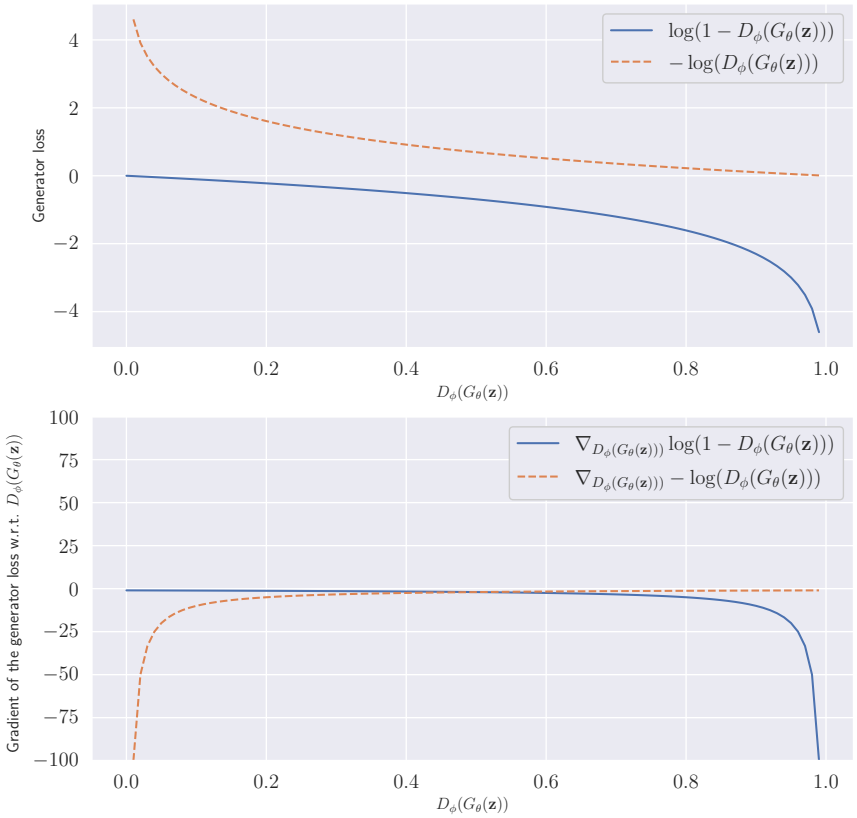

Fig. 4.2. Nonsaturating loss offers meaningful gradient information when the generator is performing poorly.

When implemented with the gradient penalty term, the weights tend to be distributed more evenly, even when the networks depth is increased.

4.5.2 Optimization

Since have laid the general ideas of neural network training in Subsection 2.2, we will now discuss the special considerations of optimization for GANs. Since GANs can be unstable to train, the choice of optimization algorithm and its hyperparameters plays even greater part on achieving stable training. Generally, momentum-based optimization with high momentum values is the typical choice on supervised learning, but in order to achieve stability on GAN training lower momentum values are preferred [1]. The authors of WGAN found out that for their WGAN implementation momentum-based optimizers, such as Adam [6] do not work very well, and used the RMSProp [7] instead. However, when implemented with the GP term, WGANs have shown to be stable to train with Adam [25]. Even though more complex optimization methods may have edge in GAN

training, their scalability for bigger datasets and network sizes can be questionable [1].

4.5.3 Conditioning GANs

Conditioning generative models has attractive properties from the practical viewpoint. Consider for example GAN trained on images of different animal species; now the generator will yield different animal species with no way of specifying the species. If we however add conditioning information about the species in the training part for both generator and discriminator, we can generate pictures of the desired animal on command. In mathematical terms, we are learning *conditional* distribution $p^\star(\mathbf{x} \mid y)$ instead of $p^\star(\mathbf{x})$. This requires the access to the conditioning information, i.e., class label of the data sample. In addition, conditioning forces the networks to use more of their representative power since the weights are shared for different generation tasks. For our use case, we might like to add information about the channel samples, e.g., signal-to-noise ratio (SNR) of the channel or type of the channel. Further, conditioning GANs may be useful for the regularization purposes, since the parameters of the network's have to be *shared* between different generation tasks.

4.6 Bayesian GANs

Since we have shown some general strengths of Bayesian modelling and inference, and also introduced GANs from the maximum likelihood perspective, developing Bayesian GANs follows naturally. At the moment, the literature on Bayesian GANs is scarce. Yet, it has been investigated to a degree with promising results. Authors of BayesGAN [17] showed that their formulation of Bayesian GAN is stable to train without any interventions to the GAN training and can produce *diverse* results with good fidelity. They proved their approach to be efficient to produce good training data to complement limited real-world data in semi-supervised learning scenario, which is a huge interest for us since real-world channel measurements are expensive and often only limited data can be obtained. ProbGAN proposed in [16] is shown to be theoretically sufficient in learning highly multimodal distributions and succeed in scenarios where the BayesGAN fails.

We should note that even though Bayesian treatment of GANs mitigate the

training instabilities without much of intervention techniques discussed already, the Bayesian framework does not prohibit the usage of those; i.e., we can train Bayesian GAN with the Wasserstein critic if desired. In this section we will have a deeper look at the already existing literature of Bayesian GANs, discuss some of the challenges arising from the Bayesian inference process as well as extend the current Bayesian formulations for GANs to allow for generative modelling of multiple related datasets via Bayesian hierarchical modelling.

4.6.1 Bayesian formulation for GANs

The goal of a Bayesian GAN is to model the data generating distribution $p_{\text{data}}(\mathbf{x})$ via modelling the distribution over generators $q_{\text{gen}}(\boldsymbol{\theta})$ and discriminators $q_{\text{disc}}(\boldsymbol{\phi})$. We denote the data density induced by the generator $G_{\boldsymbol{\theta}}(\mathbf{z})$ as $p_{\text{G}}(\mathbf{x};\boldsymbol{\theta})$. It follows that the total data distribution induced by the model is a mixture of distributions induced by the distribution over generators: $p_{\text{model}}(\mathbf{x}, q_{\text{gen}}) = \mathbb{E}_{\boldsymbol{\theta}\sim q_{\text{gen}}}[p_{\text{G}}(\mathbf{x};\boldsymbol{\theta})]$. Thus, our goal is to find a distribution $q^{\star}_{\text{gen}}$ s.t. $p_{\text{model}}(\mathbf{x}, q^{\star}_{\text{gen}}) = p_{\text{data}}(\mathbf{x})$.

In order to perform Bayesian inference for the model, we first must state our *prior beliefs* about the distributions over the discriminator and generator. In one of the first contributions for the Bayesian GANs weakly informative Gaussian priors were assigned on both the discriminator and the generator [17]. A follow-up work assigned an *evolving* prior on the generator, while using improper uniform prior on the discriminator, in order to achieve theoretical guarantees for the generator to converge to arbitrary data generating distribution [16]. Since our aim is to learn a generative model using multiple target distributions, we are restricted to use some fixed parametric prior distributions in order to allow the hierarchical structure of the model.

To formulate the posterior distribution, we must set the *likelihood function* over the models parameters. Here we follow the formulation presented in [16] and use the exponentiated GAN objective function as the likelihood function, i.e., the discriminators and generators likelihoods are obtained via taking an exponential w.r.t. (4.10). Note that since we are now working with distributions over both discriminators and generators, we must revise the GAN objectives given in (4.10) slightly. Essentially for the discrimina-

tors objective we set

$$L_{\mathrm{D}}(\boldsymbol{\phi}, p_{\mathrm{model}}), \tag{4.14}$$

emphasizing the fact that the loss is evaluated over the distribution of generators. For the generators, we set

$$L_{\mathrm{G}}(D, \boldsymbol{\theta}), \tag{4.15}$$

where $D(\cdot) = \mathbb{E}_{\boldsymbol{\phi} \sim q_{\mathrm{disc}}}[D(\cdot; \boldsymbol{\phi})]$, i.e., the discriminating score function is an average over the distribution of discriminators. In practice, the distributions over both the discriminators and the generators are achieved via running multiple chains of both models, which is a typical way of dealing with multimodal posteriors given in [10].

As for the posterior computation, SG-HMC algorithm was used in [17], but the authors noted that in order to have reasonable success with the convergence, Adam optimizer should be used for the first few thousand of iterations before switching to the SG-HMC. This strengthens the motivation given for the usage of MSGLD as well as the ASGLD for Bayesian deep learning; it might be desirable to rely only on the sampling based inference in order to justify the claims of *covering broad, multimodal distribution* via Bayesian methods. However, most of the claims will be fulfilled with the usage of multiple generators covering different modes of the posterior distribution, as it is nontrivial to use the posterior draws of one chain in generative modelling.

4.6.2 Problems with the posterior inference

As we already mentioned, it is nontrivial to use posterior draws from individual chains for the data generation with deep generative models. This is due the fact that in posterior computation the general goal of any sampler is to *converge* in some local mode of the posterior, thus requiring us to use multiple chains in order to approximate the multimodal posterior. One should note that these considerations do not arise similarly with predictive modelling, as we can use the samples even from individual chains for multiple predictions, which can then be taken as a measure of uncertainty. However, since the uncertainty in generative modelling does not have similarly well defined purpose as it does in predictive modelling, it is questionable to refer the posterior computation as *sampling* since we are essentially running multiple chains of inference which are then used

to represent individual point-masses in the parameter space.

To conclude, for the most part, Bayesian GANs seems to be mathematically well defined *ensemble* of generators. However, even in such case the empirical evidence is that the Bayesian formulation of GANs leads to easier training process, and the ensemble it produces can cover well even highly multimodal data distributions. Furthermore, the Bayesian formulation allows for the *hierarchical modelling*, which we show to be useful for learning a generative model from multiple target data distributions.

4.6.3 Bayesian GANs with hierarchical priors

As our goal is to learn a generative model from multiple $M \geq 2$ distributions, a feasible approach is to extend the current Bayesian GAN formulations to the realm of hierarchical modelling. To our best knowledge there exists no literature on hierarchical GANs at the time of writing, and thus, our contribution is to establish a simple generalization of Bayesian GAN with learnable prior distribution. To keep the formulation simple we will only rely on one generator and discriminator, i.e., the model represents only one mode of the posterior distribution. This helps to save computational time and it is a well motivated choice when the source data distribution is relatively simple. Specifically, as we will see, even with unimodal approximation of the posterior our model is successful in capturing the shared effects between the datasets while producing individual samples with good quality and diversity. Furthermore, the generalization to multiple generators and discriminators is relatively straightforward, even though there are some foundational choices on how to use the multiple models in the inference process. Ultimately, the number of generators and discriminators is a training hyperparameter that must be tuned accordingly to the application.

The choice of hierarchical modelling limits us from using the kind of evolving prior used in ProbGAN [16] as it prohibits the use of hyperpriors. Furthermore, usage of improper uniform priors is as well ruled out. Thus, an easy and well motivated choice is to place a Gaussian priors on the model parameters and some weakly informative hyperpriors on its hyperparameters, in the same manner as we saw earlier in the example with hierarchical BNN for multi-task regression modelling, illustrated in Subsection 3.5.4. For the likelihood of the hierarchical GAN, we follow the same convention as the ProbGAN [16] and use the exponential of some

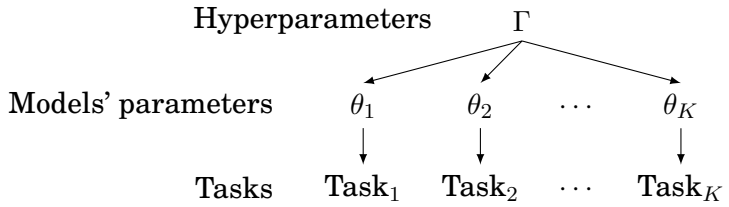

Fig. 4.3. Illustration of the hierarchical structure behind the generator. The structure of the discriminator is identical.

GAN loss function as the likelihood for the generator and discriminator.

4.6.4 Hierarchical GANs

As explained, we place Gaussian priors for the generators and discriminators layer parameters $\boldsymbol{\theta}_l$ and $\boldsymbol{\phi}_k$, respectively:

$$\begin{aligned}\boldsymbol{\theta}^{(l)} &\sim \mathcal{N}(\boldsymbol{\mu}_{\mathrm{G}}^{(l)}, \sigma_{\mathrm{G}}^{2(l)}\boldsymbol{I}) \\ \boldsymbol{\phi}^{(k)} &\sim \mathcal{N}(\boldsymbol{\mu}_{\mathrm{D}}^{(k)}, \sigma_{\mathrm{D}}^{2(k)}\boldsymbol{I}).\end{aligned}$$

For the prior hyperparameters we may set any kind of weakly informative hyperpriors. However, as the priors have a regularizing effect on the networks, it is advisable to play around with different values and asses which parameter configuration yields the best results. Our empirical observation points that somewhat broad hyperprior distributions can yield to successful training and good quality samples produced by the generator.

For the models' likelihood we use (4.14) and (4.15). Note that since there are multiple tasks, each of the tasks needs an individual likelihood. Fortunately, the extension to multiple models is trivial if we assume the tasks to be independent *a priori*. Now we may write likelihood for each of the ith generators as $\ell_i(\boldsymbol{\theta}_i \mid \Gamma_{\mathrm{G}})$ and similarly for discriminators as $\ell_i(\boldsymbol{\phi}_i \mid \Gamma_{\mathrm{D}})$, where Γ_i, $i \in \{\mathrm{D}, \mathrm{G}\}$ denotes the model hyperparameters. We denote the parameters of the hyperpriors with α, β, and let $\boldsymbol{\Theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$ and $\boldsymbol{\Phi} = (\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_K)$. The posteriors for K generators and discriminators in the hierarchical setting may now be written as:

$$\begin{aligned}p(\boldsymbol{\Theta}, \Gamma_{\mathrm{G}} | \boldsymbol{\Phi}, \alpha_{\mathrm{G}}, \beta_{\mathrm{G}}) &\propto p(\alpha_{\mathrm{G}}, \beta_{\mathrm{G}}) p(\Gamma_{\mathrm{G}} | \alpha_{\mathrm{G}}, \beta_{\mathrm{G}}) \textstyle\prod_{i=1}^{K} \ell_i(\boldsymbol{\theta}_i | \Gamma_{\mathrm{G}}) \\ p(\boldsymbol{\Phi}, \Gamma_{\mathrm{D}} | \boldsymbol{\Theta}, \alpha_{\mathrm{D}}, \beta_{\mathrm{D}}) &\propto p(\alpha_{\mathrm{D}}, \beta_{\mathrm{D}}) p(\Gamma_{\mathrm{D}} | \alpha_{\mathrm{D}}, \beta_{\mathrm{D}}) \textstyle\prod_{i=1}^{K} \ell_i(\boldsymbol{\phi}_i | \Gamma_{\mathrm{D}}),\end{aligned} \tag{4.16}$$

respectively. Note that we used the notion of K generators and discrimina-

tors to describe a generative model for K tasks. For each of the task, it is possible to use multiple generators (and discriminators) in order to cover the multimodality of the posterior distribution, as discussed in the general case of Bayesian GANs. We note that for our use case with relatively simple data generating distribution per task, even a simple generator can yield good results.

The posterior computation may now be done via any of the discussed posterior inference methods. We note that even a simple MAP estimate can yield great results, and may be a well motivated choice given the unclear motivation behind the full posterior computation discussed earlier. In our experiments, we observed that even with the special case of one dataset the hierarchical structure leads to a learning objective which stabilizes the GAN training well even when the nonsaturating loss is used and the generator and discriminator are trained with equal number of epochs, i.e., there was no need to first train the discriminator for e epochs before training the generator.

Even though the training process of the GAN may be more stable in the Bayesian framework, it should be emphasized that the architectural choices for both the generator and discriminator have a huge impact on the training of the GANs and the performance of the generator. To keep the discussion as general as possible we will not touch the architectural choices here, as they are application specific. The hierarchical formulation can be used with any kind of neural networks acting as generator and discriminator. We will defer the discussion of architectural choices and other practicalities for our application of channel modelling.

5. Generative modelling for the air channel

Air as a communication channel has many challenges for reliable, high-speed communication [27]. As transmitted signal propagates through the air, its power decreases due to absorption of the air itself – this effect is particularly strong when utilising the millimeter frequency band, required by many modern wireless systems. Additionally, when the signal propagates through obstacles such as trees or walls, its power attenuates similarly, an effect often called *shadowing*. Furthermore reflections and scattering from any obstacles causes both time- and phase shifts for the signal, called the *multipath propagation*. Due to these effects, the received signal arrives in many time- and phase shifted components of varying power.

Since the availability of channel measurements is often a bottleneck in developing a wireless communication system, there exists numerous analytical models trying to approximate the properties of the air channel. One of the simplest of such models is the *tapped delay line* (TDL) channel model. The TDL channel model may be modelled via a finite impulse response (FIR) filter. In time domain, the filter may be written by a difference equation:

$$y[n] = \sum_{i=0}^{N} c_i x[n-i],$$

where $c_i \in \mathbb{C}$ are complex-valued weights. The N denotes the number of *taps* in the TDL model. Thus, the characteristics of the TDL channel model are parameterized through the number of taps and the complex-valued weights. In order to have realistic model of the air channel in various scenarios, 3GPP project [28] has standardized five TDL channel models, three of which represents the non line-of-sight (NLOS) scenarios, and the rest line-of-sight (LOS) scenarios.

For the objectives of this thesis, we find the TDL channel models sufficient. However, the most accurate channel models are based on *raytracing* [27]. In raytracing based channel models the environment of the wireless communication system is modelled precisely in aims for the raytracing algorithms faithfully reproduce the signals' propagation through various environments. Countrary to the analytical channel models, which are relatively computationally inexpensive, the accuracy of raytracing based modelling comes with high computational cost. Thus, our aim is to utilise *generative modelling* to *learn* the channel conditions accurately from channel measurements so that such model can produce new channel measurements in a rapid fashion.

5.1 Recent work

As the generative channel modelling is relatively novel area of research, there exists only a handful of literature on the subject. However, GAN based modelling has been shown to be effective for the problem at hand. Particularly, MIMO-GAN [29] proposed a GAN based modelling scheme where a generative channel model was learnt for the channel impulse response. Even though the modelling scheme was shown to be effective for MIMO, the training data was generated in a manner where impulses were transmitted sequentially from individual transmitter antennas, which is infeasible in any practical scenario. Furthermore, in ChannelGAN similar modelling scheme was proposed for modelling the air channel in the time domain [30]. Diffusion models for this same problem have been proposed as well [31, 32], but due to the computational cost of generating channel instances, we argue that GANs may be a better alternative. Since GANs have negligible generation time, they can serve a better purpose in applications where the generated channel instances are used to further train components of the transmission chain, such as deep-learning based receivers.

We note that there exists no standardized way of approaching the modelling problem. Channel instances may be for example in time- or frequency domain, or in both, if desired. Our aim is to approach the problem in such a way that the format of the channel instance has little effect to the modelling part. However, we limit the scope of the modelling to the simple single-input single-output (SISO) case.

Channel measurements are expensive to obtain. Therefore, our aim is to achieve as much data efficiency as possible in the generative modelling. The problem of limited data is noted in the literature, yet there exists few proposals on how to optimize the model performance with limited data. In [32], the proposed solution was to train initial model with larger dataset and fine-tune the model on a smaller dataset with decreased learning rate. Even though this kind of approach is often used in many applications, it lacks rigorous reasoning, and it poses the challenge on setting the limited learning rate for the adaptation. Fine-tuning in this manner equals to finding a starting point via the first dataset and then optimizing with limited learning rate on another dataset, thus not utilizing any of the dataset to their fullest extent. Furthermore, this approach leaves us with only one channel model, not enabling the modelling of multiple scenarios with information shared between them. Thus, our contribution is to propose a novel way of generative modelling from multiple related datasets via hierarchical Bayesian GAN.

5.2 Hierarchical generative modelling of the air channel

As discussed, we limit the scope of modelling to the SISO case in time-domain. Since the channel impulse responses are complex-valued, we have to rearrange them in some way to make learning with real-valued neural networks possible. A simple solution is to "flatten" the complex valued array to one real-valued array, so that the first part of the array contains the real part of the impulse response and the second part of the array contains the complex part of the impulse response. The function $\texttt{flatten} : \mathbb{C}^n \rightarrow \mathbb{R}^{2n}$ is illustrated in Figure 5.1.

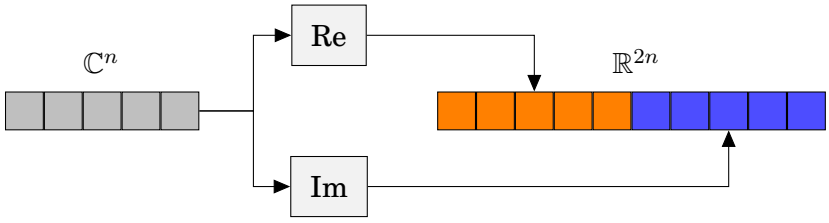

Fig. 5.1. Complex-valued impulse response is transformed to a real-valued vector.

5.2.1 Channel datasets

Since our aim is to enhance data-efficiency of channel modelling via hierarchical modelling, our model must utilize multiple datasets. For a realistic scenario we utilize MATLABs 5G toolbox to generate instances from two TDL-A channels with different delay spreads, 300ns and 200ns, respectively. The first dataset consist of 12,000 channel impulse responses, whereas the second one only contains 3,000 channel impulse responses. Both of the channels are sampled for 4.17ns with a sampling rate of 30.72GHz, identically to what was done in MIMO-GAN [29], resulting in 128 samples per impulse response. Four randomly sampled channel instances from the two channels are illustrated in Figure 5.2.

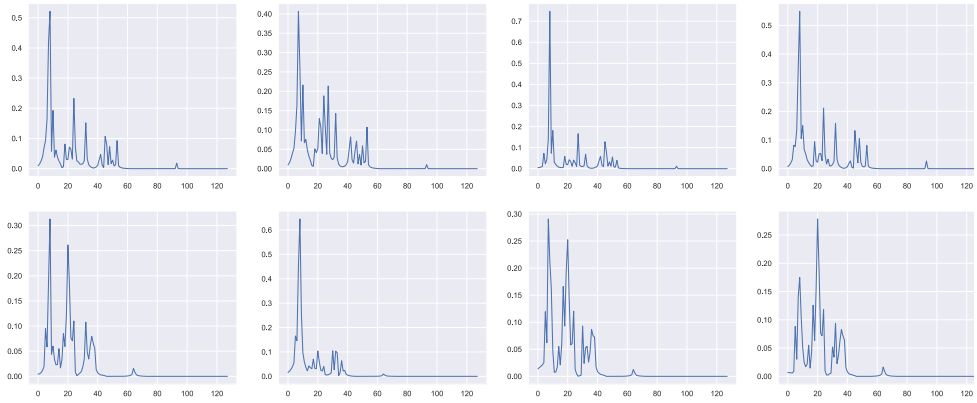

Fig. 5.2. Magnitudes of impulse responses in the time domain from the two different channels. The first row represents instances from the first channel, whereas the second row from the second one.

We note here that even 12,000 instances is a modest amount, but the amount of data needed for modelling is highly variable on the complexity of the channel, and for our SISO case with stationary receivers, it is likely to be enough. It could be argued that the two generated channels are very close to each other, however, it does not make much sense to apply hierarchical modelling for data generated by completely different mechanisms. For example, the conditions here could be generated by two different cities with different density of obstacles and distances between transmitters and receivers.

Our goal is to achieve an accurate generative model for the second channel dataset, which is very limited in data volume. In practical scenario this

kind of data augmentation could be done first by simulating the channel conditions as close as possible and hierarchically learn model using real channel measurements. Another possibility would be to have a modest number of datasets all of limited size. However, as it is possible to simulate highly realistic channel scenarios via for example ray-tracing based solutions, there is no reason for not to do so for more accurate generative modelling.

5.2.2 Hierarchical Bayesian GAN for multi-task generative modelling

We have laid the mathematical foundations of hierarchical GANs, leaving us with only the practical choices of the generator and discriminator networks themselves. For both, we use simple hierarchical MLPs with non-centered parameterization, similarly as illustrated in Figure 3.4. The generator network consists of two hidden layers with hyperbolic tangent as an activation function, whereas for the discriminator network, we use only one hidden layer with LeakyReLU activation and sigmoidial output. The input noise for the generator was sampled from 128-dimensional isotropic standard normal distribution, which was then mapped to 256-dimensional real space so that the first 128 values represent the real part of the impulse response and the rest the imaginary part. The layers of generator have a shape of $128 - 128 - 256 - 256$. The discriminators input is the 256-dimensional impulse response, which is then mapped to real number between 0 and 1, representing the probability for the fed impulse response being fake or real, respectively. Layers of the discriminator are of shape $256 - 128 - 1$. Architectures of both generator and discriminator networks are illustrated in Figure 5.3.

For the hyperparameters of both networks we assigned nearly noninformative hyperpriors: $\boldsymbol{\mu}^{(l)} \sim \mathcal{N}(\mathbf{0}, 100^2\boldsymbol{I})$, $\sigma^{(l)} \sim \text{Exp}(0.5)$, to allow the model to learn the hyperparameters from the data with little to no regularization. Furthermore, for this kind of generating task, there exists no meaningful way to assign any kind of informative priors as the models are considered to be black boxes. The batch sizes were set to 128 for both networks, i.e., for the discriminator, we fed 128 samples and with the generator we generated 128 samples. The training itself did not require any kind of interventions typical for GAN training: the discriminator and generator were trained for equal epochs and the loss function used was the nonsaturating loss. In the training, we did not experience mode collapses typical for GAN training,

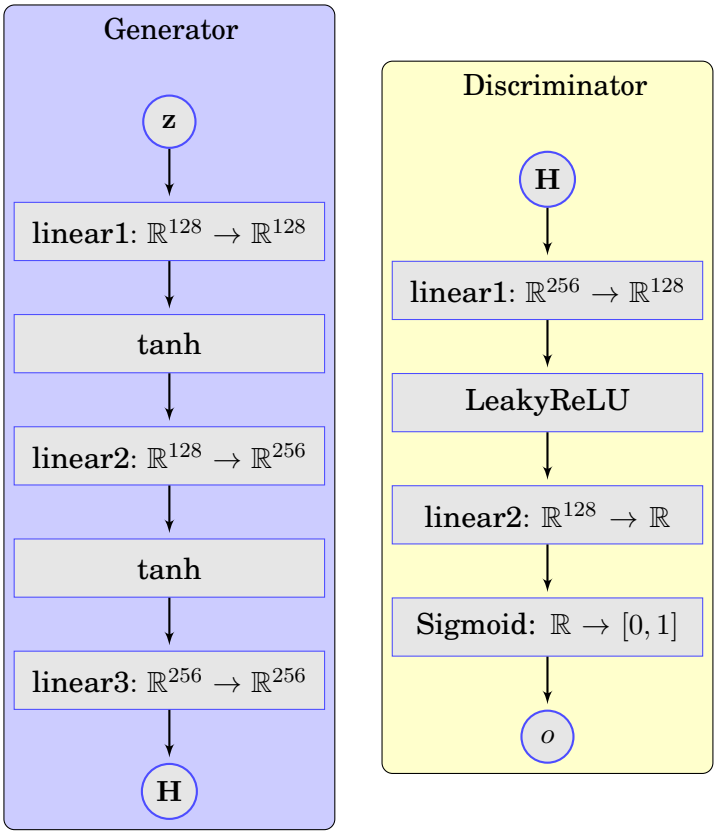

Fig. 5.3. Neural architectures of the generator and discriminator networks.

although finding the right architectural choices helped the process.

The training was done in the same spirit as with BayesGAN [17] and ProbGAN [16]. First, Adam optimizer was used to find a reasonable MAP estimate of the posterior. However, after the MAP estimate was found, we did not continue to the sampling process with any SG-MCMC method because of the difficulty of utilizing the posterior samples, discussed in subsection 4.6.2, and due to the fact that even this MAP estimate produced excellent results in terms of sample diversity and fidelity. We however do not rule out the usage of some SG-MCMC method in order to obtain even better parameter configuration. However, as long as there are no sensible way of utilising the posterior samples, every method for the inference will only rely on a point-mass in probability space then utilised for the generation, motivating the usage of a simple MAP estimate from the practical perspective. In order to reduce the noise from minibatching, the learning rate was reduced linearly.

Figure 5.4 shows magnitudes of the impulse responses generated from the second channel. We can seen that the generator produces diverse, high-fidelity samples even if the size of the dataset was very limited. However,

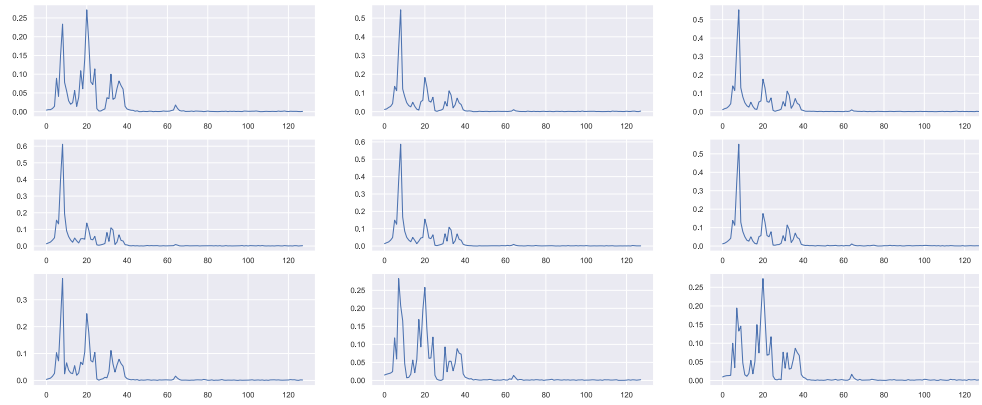

Fig. 5.4. Magnitudes of the impulse responses generated from the second channel via hierarchical GAN trained with the two datasets.

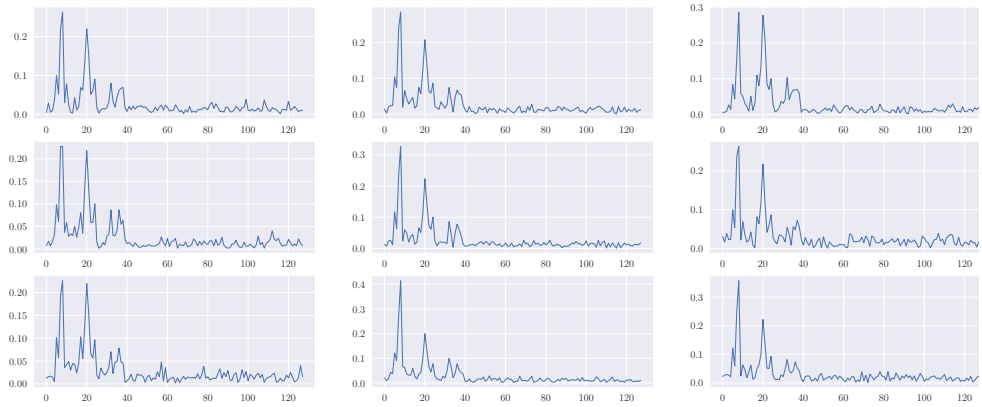

Fig. 5.5. Samples produced by the hierarchical generator trained only with the second dataset yields much worse individual sample quality.

this kind of inspection allows only for intuitive view of the success of the model. To gain more confidence that the utilisation of first, larger dataset yields any boost for performance, we trained an identical hierarchical GAN, using only the second dataset, which yielded much worse sample fidelity, shown in Figure 5.5.

Note that we have only compared the magnitudes of the impulse responses, but the generator and discriminator directly produced and evaluated complex valued impulse responses, without direct information of the magnitude response in time-domain. To gain even deeper understanding, we compare the mean impulse responses between generated data and validation set of the second channel. For validation set, we had 17,000 instances from the second channel. From Figure 5.6, we can see that the average channel impulse response is nearly identical to the one of the true value, which is

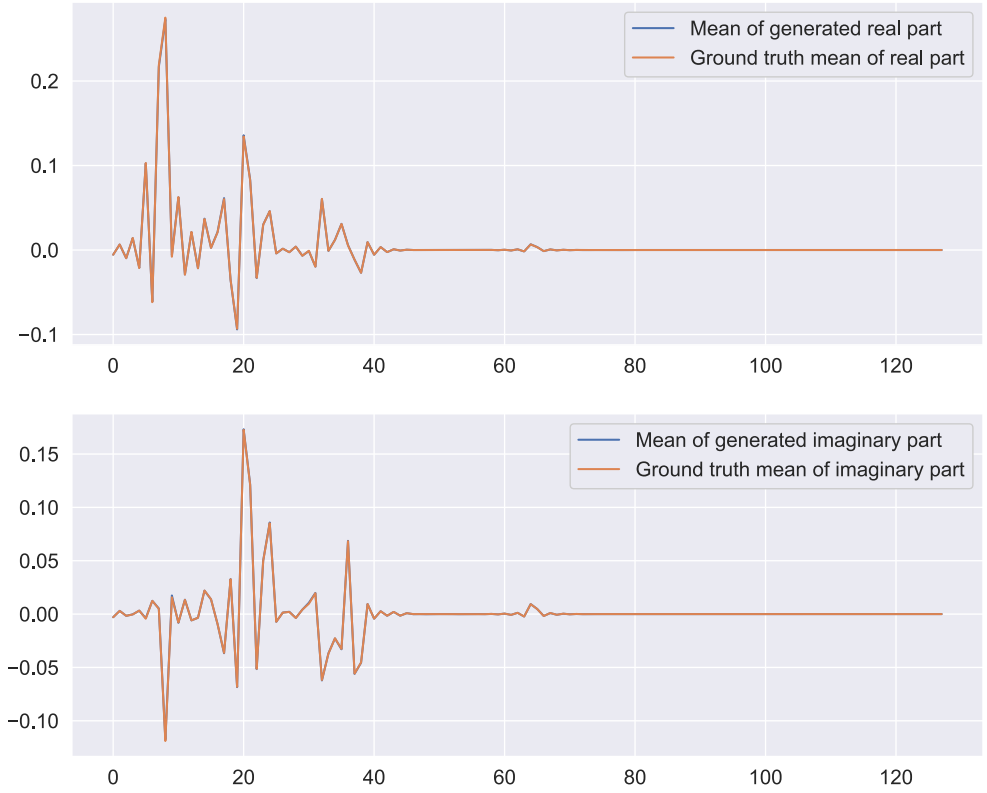

Fig. 5.6. Comparison of the mean impulse response between the validation set of the second channel and instances generated by the hierarchical generator.

impressive since GANs do not have any explicit mechanism to measure the average of the data distribution they are trying to learn.

5.3 Analytical evaluation of the produced channel

In order to measure the performance of the channel generation analytically, we must resort to some statistics. We may for example assume that the channel magnitude response in time domain is normally distributed with a diagonal covariance: $|H| \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2\mathbf{I})$, where $H \in \mathbb{C}^{128}$ and $|H| = [|h_1|, |h_2|, \ldots, |h_{128}|]^T \in \mathbb{R}^{128}$. Now we may estimate the parameters of the aforementioned distribution via samples generated by our hierarchical GAN and the validation dataset, and compare the estimates, assuming the estimates from the validation data to be the ground truth. Note that we use the notion of the magnitude response on purpose here, since our MLP based GAN does not have access to the magnitude explicitly, thus making the results more interpretable, as the magnitude couples the real-

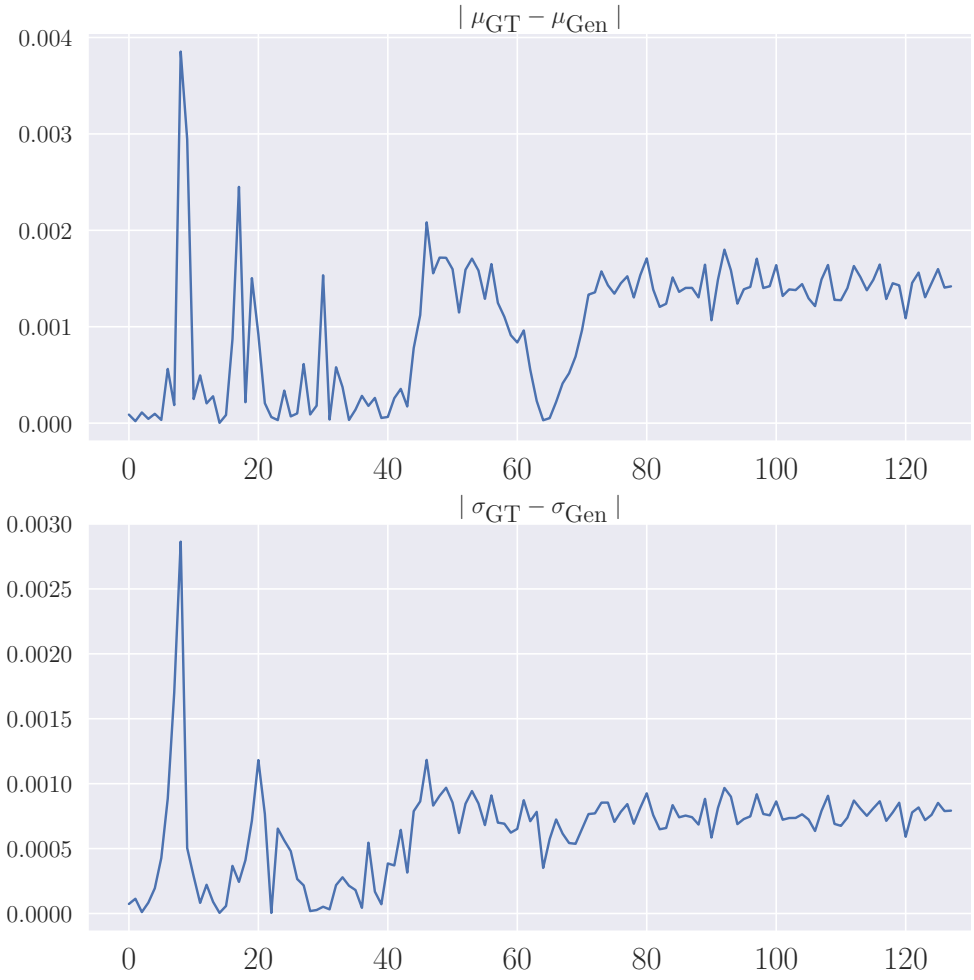

Fig. 5.7. Differences between the true channel and generated one in terms of first- and second order statistics show that the generative model learns the channel producing mechanism accurately.

and imaginary parts of the generated impulse response.

We denote the ground truth mean by μ_{GT} and standard deviation by σ_{GT}. Conversely, the mean statistic from the generated channel is denoted by μ_{Gen} and standard deviation by σ_{Gen}. The ground truth statistics are estimated from the validation set with 17000 samples, wheres from the generative model we sample 100000 samples. Since the both statistics are 128-dimensional points in the real space, intuition can be gained by plotting the absolute value of the differences between the statistics, $|\mu_{\text{GT}} - \mu_{\text{Gen}}|$ and $|\sigma_{\text{GT}} - \sigma_{\text{Gen}}|$. From Figure 5.7 we can see that the differences between the true and generated channel are quite small.

In addition to visualizing the differences, we may also calculate some error metrics between the statistics of true and generated channel. Consider for example mean squared error (MSE), defined as:

$$\frac{1}{d}\sum_{i=1}^{d}(y_i - \hat{y}_i)^2, \tag{5.1}$$

where $\hat{y}$ denotes the estimated value, and y the ground truth. For the mean statistics, the MSE is $1.47 \cdot 10^{-4}$, and for the standard deviation it is $5.23 \cdot 10^{-5}$.

It is clear that our generative procedure produces high-quality channel instances with good coverage of the true data generating distribution. Even though the dataset used for the illustration is relatively simple, clearly GANs do not have any problem modelling much more diverse distributions. Furthermore, we have shown that the modelling of a channel with only limited data is feasible using hierarchical generative modelling, given that we have multiple related datasets, which is usually the case in designing wireless communication systems for similar, but ultimately different locations. These results are also reported in a conference paper to be submitted to IEEE ICASSP 2025 [33].

6. Conclusions

In this thesis, we have investigated generative modelling of the air transmission channels via GANs. Specifically, we have utilised the Bayesian hierarchical modelling in order to extend the earlier Bayesian GAN formulation to allow for multi-task generation. Our simulations show that via hierarchical modelling, the data efficiency of GANs can be enhanced by using multiple related, but limited datasets. Even though we have utilised MLPs, the hierarchical formulation can be extended straightforwardly to different neural architectures. Since the amount of data in many practical applications is limited due the sheer expenses of collecting data, utilisation of multiple related datasets in generative modelling can remedy plethora of such applications. Futher, we observed that our hierarchical formulation for GANs leads to a training process which seems to be relatively stable, even without any traditional interventions usually required for stabilizing the training process.

6.1 Future research directions

Although our experiments with hierarchical GANs were evident and convincing, there exists a plenty of room for improvement. With more diverse source data distributions, usage of multiple generators should be studied. Further, the training process of the GANs should be based on computation of the full posterior distribution, instead of simple point estimate. Furthermore, to better model the air channel, different neural architectures should be investigated. Specifically, for modelling the air channel in both time- and frequency domains, convolutional neural networks should be considered. It should also be emphasized that our current experiments relied on access to all of the channel datasets in the training process. Since this increases the requirements of storage, which may be an issue in a

practical scenario, *transfer learning* for Bayesian GANs should be further investigated.

References

[1] K. P. Murphy, *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023. [Online]. Available: http://probml.github.io/book2

[2] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[3] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016, http://www.deeplearningbook.org.

[4] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford university press, 1995.

[5] B. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0041555364901375

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[7] T. Tieleman, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, p. 26, 2012.

[8] M. Staib, S. J. Reddi, S. Kale, S. Kumar, and S. Sra, "Escaping saddle points with adaptive gradient methods," *CoRR*, vol. abs/1901.09149, 2019. [Online]. Available: http://arxiv.org/abs/1901.09149

[9] C. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. Springer, 2006. [Online]. Available: https://books.google.fi/books?id=qWPwnQEACAAJ

[10] Y. Yao, A. Vehtari, and A. Gelman, "Stacking for Non-mixing Bayesian Computations: The Curse and Blessing of Multimodal Posteriors," *arXiv preprint arXiv:2006.12335*, 2021. [Online]. Available: https://arxiv.org/abs/2006.12335

[11] A. Gelman, J. Carlin, H. Stern, D. Dunson, A. Vehtari, and D. Rubin, *Bayesian Data Analysis, Third Edition*, ser. Chapman & Hall/CRC Texts

in Statistical Science. Taylor & Francis, 2013. [Online]. Available: https://books.google.fi/books?id=ZXL6AQAAQBAJ

[12] R. M. Neal *et al.*, "MCMC using Hamiltonian dynamics," *Handbook of Markov Chain Monte Carlo*, vol. 2, no. 11, p. 2, 2011.

[13] Y.-A. Ma, T. Chen, and E. Fox, "A Complete Recipe for Stochastic Gradient MCMC," *Advances in neural information processing systems*, vol. 28, 2015.

[14] M. Welling and Y. W. Teh, "Bayesian Learning via Stochastic Gradient Langevin Dynamics," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11. Madison, WI, USA: Omnipress, 2011, p. 681–688.

[15] T. Chen, E. Fox, and C. Guestrin, "Stochastic Gradient Hamiltonian Monte Carlo," *31st International Conference on Machine Learning, ICML 2014*, vol. 5, 02 2014.

[16] H. He, H. Wang, G.-H. Lee, and Y. Tian, "Bayesian modelling and monte carlo inference for GAN," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=H1l7bnR5Ym

[17] Y. Saatci and A. G. Wilson, "Bayesian GAN," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/312351bff07989769097660a56395065-Paper.pdf

[18] S. Kim, Q. Song, and F. Liang, "Stochastic Gradient Langevin Dynamics Algorithms with Adaptive Drifts," *arXiv preprint arXiv:2009.09535*, 2020. [Online]. Available: https://arxiv.org/abs/2009.09535

[19] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608098001166

[20] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

[21] S. Kullback and R. A. Leibler, "On information and sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[22] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, G. Klambauer, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Nash Equilibrium," *CoRR*, vol. abs/1706.08500, 2017. [Online]. Available: http://arxiv.org/abs/1706.08500

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Nets," in *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Curran Associates, Inc., 2014. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[24] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *arXiv preprint arXive:1701.07875*, 2017. [Online]. Available: https://arxiv.org/abs/1701.07875

[25] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved Training of Wasserstein GANs," *CoRR*, vol. abs/1704.00028, 2017. [Online]. Available: http://arxiv.org/abs/1704.00028

[26] M. Arjovsky and L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," *arXiv preprint arXive:1701.04862*, 2017. [Online]. Available: https://arxiv.org/abs/1701.04862

[27] A. Goldsmith, *Wireless Communications*, ser. Cambridge Core. Cambridge University Press, 2005. [Online]. Available: https://books.google.fi/books?id=n-3ZZ9i0s-cC

[28] 3GPP, "Study on channel model for frequencies from 0.5 to 100 GHz." 3rd Generation Partnership Project, Technical Specification Group Radio Access Network, TR 38.901, Tech. Rep., 2017.

[29] T. Orekondy, A. Behboodi, and J. B. Soriaga, "MIMO-GAN: Generative MIMO Channel Modeling," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 5322–5328.

[30] H. Xiao, W. Tian, W. Liu, and J. Shen, "ChannelGAN: Deep Learning-Based Channel Modeling and Generating," *IEEE Wireless Communications Letters*, vol. 11, no. 3, pp. 650–654, 2022.

[31] M. Arvinte and J. Tamir, "Score-based generative models for wireless channel modeling and estimation," in *ICLR Workshop on Deep Generative Models for Highly Structured Data*, 2022.

[32] U. Sengupta, C. Jao, A. Bernacchia, S. Vakili, and D.-s. Shiu, "Generative Diffusion Models for Radio Wireless Channel Modelling and Sampling," *arXiv preprint arXiv:2308.05583*, 2023.

[33] J. Kuikka, E. Ollila, and S. A. Vorobyov, "Multi-task generative modelling for the air channel via hierarchical GANs," *ICASSP 2025-IEEE International Conference on Acoustics, Speech and Signal Processing*, 2025.