UNIVERSITÀ DEGLI STUDI DI PADOVA

Department of Mathematics "Tullio Levi-Civita"
Doctoral Program in Mathematics
Curriculum: Computational Mathematics
(Ciclo XXXVI)

Ph.D. Thesis

Wasserstein Data Generation Models

Coordinator: Giovanni Colombo
Supervisor: Mario Putti
Co-supervisor: Gabriele Santin

Candidate: Amna Mohsin

July, 2024

Contents

Abstract

Model-based or PDE-based uncertainty quantification tries to reconstruct the probability distribution of the quantity of interest, represented for example by the solution of a parametric PDE, given the probability distribution of the uncertainty on the model data. Typically this approach requires the use of Monte-Carlo methods and the consequential construction of very large ensemble, each member of which corresponds to the solution of a PDE with data sampled from their uncertainty distribution. This constitutes an enormous task often unfeasible in terms of computational time. It is then very important to be able to reduce the computational requirements of the construction of the Monte-Carlo ensemble.

The main focus of this thesis deals with the efficient construction of these Monte-Carlo ensembles for general parametric scalar PDEs by leveraging on the strength of existing AI tools to ascertain if they can be used effectively for the purpose of Monte-Carlo data generation. In particular, we consider the weaker requirement of achieving accurate reproduction not of every single realization of the ensemble but only of its empirical moments.

Specifically, we work with Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs)m because of the possibility to enforce on the trained latent space a given probability distribution. A well trained VAE will then allow the construction of an accurate ensemble capable of reproducing the empirical moments of the sought distribution by sampling from the pre-imposed probability distribution of the latent space and using the decoder to generate the full dimensional data.

The success of these generative models depends fundamentally by their accurate training, which require the evaluation of the distance between two parametric PDE solution. Standard implementations employ the Jensen-Shannon or Kullback-Leibler divergences for this task. However, the limited continuity and accuracy of these distances are believed to be the main cause of inaccuracy in the training phase. To alleviate this problem, the Wasserstein distance, a metric used to measure the distance between probability distributions, has been often employed. However, standard approaches, because of limitations due essentially to the computational difficulties in the calculation of the Wasserstein distance, admit a tradeoff in favor efficiency sacrificing accuracy. In the case of data generation this sacrifice cannot be done and accurate evaluation of the Wasserstein distance seems mandatory.

The work in this thesis builds on this idea by employing a modern and recently developed Wasserstein distance calculation capable of accuractely calculate the Wasserstein-1 distance between two distributions. Thus, in the first part of the thesis builds on this work with the aim of adapting this approach towards its implementation in VAEs or GANs. This work is still in progress.

In the second part, we address the problem of implementing this distance calculator within the framework of Uncertainty Quantification (UQ) and parametric Partial Differential Equations (PDEs). The primary goal is to quantify the effect on the solution of the uncertainties in the input parameters. We employ Monte Carlo (MC) method to evaluate the quantities of interest, where we utilize data generation techniques for the construction of MC ensemble. We present the current results of our implementation of the standard VAE for this problem, along with some initial results of the Wasserstein Autoencoder (WAE) model. Lastly, we outline our future directions to achieve the ultimate goal of this research.

Introduction

Uncertainty quantification (UQ) is a critical aspect of modern computational science and engineering, particularly in the context of solving complex mathematical models described by partial differential equations (PDEs). In many real-world applications, these models are subject to various forms of uncertainty, which can arise from several sources, including measurement errors, approximations in the model, and inherent variability in physical processes. As a result, understanding how these uncertainties propagate through the system and affect the solution of the model is essential for making informed predictions and decisions.

One approach to uncertainty quantification is model-based or PDE-based uncertainty quantification, which seeks to reconstruct the probability distribution of a quantity of interest—often the solution to a parametric PDE—given the probability distribution of the uncertainty in the model's input data. This type of analysis is typically carried out using Monte Carlo (MC) methods, which require generating a large ensemble of model realizations, each corresponding to the solution of a PDE with model data sampled from the uncertainty distribution. While Monte Carlo methods are conceptually simple and versatile, they can be computationally expensive due to the large number of model evaluations required, especially when the dimensionality of the problem is high or the complexity of the model is significant.

Reducing the computational burden associated with the construction of these Monte Carlo ensembles is a critical challenge. Traditional techniques for sampling and simulating these large ensembles often suffer from inefficiencies, leading to prohibitively high computational costs. Therefore, the need for more efficient techniques that can still produce accurate results has driven research in this area, particularly in the application of machine learning (ML) and artificial intelligence (AI) tools to streamline the data generation process.

This thesis explores the potential of leveraging existing AI tools, specifically generative models, to efficiently construct Monte Carlo ensembles for uncertainty quantification in the context of parametric scalar PDEs. Rather than aiming to exactly reproduce every realization of the ensemble, we focus on a more achievable goal: accurately reproducing the empirical moments of the ensemble. The empirical moments, which are statistical quantities like the mean, variance, and higher-order moments, provide a sufficient characterization of the underlying probability distribution in many cases, and can be used to derive the desired uncertainty estimates for the solution of the PDE.

Generative Models for Data Generation

Among the various AI techniques available, Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) have shown great promise for generating data that approximates a desired distribution. These models have been widely applied in image generation, anomaly detection, and other domains, and they hold potential for improving Monte Carlo data generation in uncertainty quantification tasks.

In the context of this work, we focus on VAEs and GANs due to their ability to learn and sample from a latent space that can be constrained to match a given probability distribution. A well-trained VAE, for example, learns a probabilistic mapping from a latent space to the original data space. Once trained, we can sample from the latent space, using the decoder to generate full-dimensional realizations of the data. By appropriately training the VAE to match the distribution of the PDE solutions, we can generate an ensemble of solutions that accurately reproduces the empirical moments

of the uncertainty distribution, without the need to perform extensive Monte Carlo simulations.

However, the success of these generative models depends critically on their training process, which requires measuring the distance between two distributions. In standard implementations, metrics such as the Jensen-Shannon divergence or Kullback-Leibler divergence are often used to quantify this distance. While these metrics have proven effective in some applications, they suffer from limitations in terms of continuity and accuracy when applied to the comparison of solutions to PDEs. These limitations are believed to be a major source of inaccuracy during the training phase of generative models.

The Wasserstein Distance: A More Accurate Metric

To address these limitations, this thesis explores the use of the Wasserstein distance, a metric that has gained significant attention in recent years due to its ability to more accurately measure the distance between probability distributions. The Wasserstein distance, particularly the Wasserstein-1 distance, is known for its strong theoretical properties and ability to handle distributions with limited overlap, making it well-suited for comparing solutions to parametric PDEs.

One of the challenges in applying the Wasserstein distance to generative model training is the computational difficulty involved in its calculation. Traditional methods for computing the Wasserstein distance are often inefficient and may require significant approximations, which can reduce the accuracy of the training process. In this thesis, we focus on recent advancements in Wasserstein distance calculation that enable more efficient and accurate computation of the Wasserstein-1 distance, which is crucial for training generative models in the context of uncertainty quantification.

Objectives and Structure of the Thesis

The primary goal of this thesis is to investigate and develop methods for efficiently constructing Monte Carlo ensembles for uncertainty quantification using generative models, with a further focus on the use of the Wasserstein distance in model training. The thesis is organized as follows:

- **Chapter 1** introduces some fundamental results on Optimal Transport and on the Wasserstein distance. We present in particular the Monge and Kantorovich formulations of Optimal Transport, and the existence of an Optimal Transport map, focusing in particular on its L^1 formulation. We then discuss the Monge Kantorovich equations and their dynamical counterparts, that will be the basis of the efficient computational techniques presented in the next chapter.

- **Chapter 2** presents the novel compuatational method for the approximation of the Wasserstein-1 distance. This is based on the spectral solution, using Fourier methods, of the dynamic Monge-Kantorovich formulation introduced in the first chapter. We discuss in particular details of its derivation and a Matlab implementation.

- **Chapter 3** provides an overview of the theory and methods underlying uncertainty quantification in the context of parametric PDEs. We discuss the standard Monte Carlo methods for uncertainty quantification, as well as the challenges associated with their computational cost. This chapter sets the stage for the introduction of machine learning-based methods as a potential solution.

- **Chapter 4** delves into the specifics of generative models, with a particular emphasis on VAEs and GANs. We discuss their theoretical foundations, the process of training these models, and how they can be integrated in a larger Machine Learning framework.

- **Chapter 5** focuses on the integration of generative models with uncertainty quantification for parametric PDEs. We present the current results of using VAEs to generate Monte Carlo ensembles and analyze the preliminary performance of Wasserstein Autoencoders (WAEs) in this context.

In summary, this thesis aims to develop novel methods for reducing the computational cost of uncertainty quantification using Monte Carlo simulations, by leveraging the power of generative models and the Wasserstein distance. The success of this work could pave the way for more efficient and accurate uncertainty quantification in a wide range of scientific and engineering applications.

Several aspects discussed in this thesis are still work in progress and will require further work. In particular, the approximation method of Chapter 2 is still not converging to the exact solution. Moreover, the computational method for the approximation of the approximation of Monte Carlo simulations has so far been applied only to a rather simple test problem. Moreover, the extensive training required already for this simple test case, have highlighted that a more detailed controlled of the training of the generative models will be required for scaling this methods to produce reliable approximations of more challenging test cases.

Chapter 1

Optimal Transport and Wasserstein distance

In this chapter, we delve into the essential background of the Optimal Transport Problem (OTP), a fundamental mathematical problem centered around determining the most efficient means of redistributing mass or resources from one initial configuration to a desired final configuration. We present the original formulation of the problem by Gaspard Monge, and the subsequently developed Kantorovich formulation, which introduced greater flexibility. Our research primarily focuses on a specific variant of OTP, which is L_1-OTP, where transportation cost is the Euclidean distance.

Building upon this introduction, we draw upon prior research conducted in [15], shedding light on the most intriguing findings from that work that have direct relevance and significance to our own research objectives.

1.1 Optimal Transport

In this section, we provide an overview of the Optimal Transport Problem (OTP), which aims to find the least-cost strategy for redistributing mass from an initial configuration to another. We cover fundamental definitions and well-known results in OTP theory, focusing on aspects relevant to our contribution. We begin with Gaspard Monge's original formulation in 1781 and then describe the relaxed Kantorovich formulation. Specifically, we focus on $L^1 - OTP$, where the transport cost is the Euclidean distance. Additionally, we define the Monge-Kantorovich partial differential equations, which play a central role in $L^1 - OTP$ theory and in our thesis. Overall, this section provides a comprehensive overview of the Optimal Transport Problem, its key formulations, and the significance of the Monge-Kantorovich equations.

1.1.1 Monge Formulation

The Monge formulation of the Optimal Transport Problem (OTP) [44] is one of the original approaches to solving this problem. It was first introduced by Gaspard Monge in 1781 and focuses on finding an optimal mapping between two probability distributions.

In the Monge formulation, the goal is to determine a transport map that minimizes the total cost or distance of moving mass from the source distribution to the target distribution. The transport map specifies how each point in the source distribution is mapped to a point in the target distribution.

Mathematically, given two non-negative Radon measures f^+ and f^- with equal volume (we will denote with $\mathcal{M}_+(X)$ the set of the non-negative measures defined on a measure space X), the ambient spaces for the measures f^+, f^- are two complete and separable spaces X and Y , but in most of the cases we will assume $X = Y = \Omega$ where Ω is an open, bounded, convex, and connected domain in $\mathbb{R}^d$ and with smooth boundary. A measurable map $T : X \to Y$ is called a ***transport map*** from f^+ to f^- if:

$$f^- = T_{\#} f^+$$

where $T_{\#}$ denotes the push-forward measure.

The Monge formulation seeks to find a transport map T that minimizes the total transportation cost, it is as follows:

Minimize:

$$\mathbb{M}(T) = \int_X c(x, T(x))\, df^+(x) \tag{1.1}$$

among all transport maps T from f^+ to f^-. Here $c(x, y)$ is the cost of moving a unit of mass from point x to point y. In the original Monge problem the cost function was the distance $|x - y|$.

Despite its historical significance, the Monge formulation can be ill-posed and the optimal map may not exist. For example when f^+ is a Dirac measure and f^- is not, the set of transport maps is empty since the image measure $T_{\#}f^+$ is atomic [45]. This has led to the development of alternative formulations, such as the Kantorovich formulation, which relaxes some of the constraints and allows for multiple optimal transport plans.

1.1.2 Kantorovich Formulation

After 150 years, Leonid Kantorovich revisited Monge's problem from a different view point [45]. He consider transport plans rather than transport maps. Given that π_X and π_Y are the projection maps on X and Y, respectively, we define the set of transport plans as follow: A map $\gamma \in \mathcal{M}_+(X \times Y)$ is called a ***transport plan*** from f^+ to f^- if:

$$(\pi_X)_{\#}\gamma = f^+, \quad (\pi_Y)_{\#}\gamma = f^-$$

Denoting by $\mathcal{A}dm(f^+, f^-)$ the set of all transport plans from f^+ to f^-, the Kantorovich formulation is the following:

Minimize:

$$\mathbb{K}(\gamma) = \int_{X\times Y} c(x, y)\, d\gamma(x, y) \tag{1.2}$$

among all transport plans $\gamma \in \mathcal{A}dm(f^+, f^-) \subset \mathcal{M}_+(X \times Y)$ from f^+ to f^-.

Kantorovich Formulation is now known as the *Monge-Kantorovich* problem. However, the Kantorovich formulation of the optimal transport problem is considered weaker than Monge's formulation, as for any map T belonging to the set of transport maps, we can define a plan $\gamma_T = (Id, T)_{\#}f^+$ that belongs to $\mathcal{A}dm(f^+, f^-)$. Furthermore, if T^* solves Monge's problem, then γ_{T^*} solves Kantorovich's problem.

One of the key advantages of the Kantorovich relaxed formulation is that, under mild assumptions on the cost function c, it becomes relatively straightforward to establish the existence of the optimal transport plan. As stated in the following theorem

Theorem 1 *For any $c : X \times Y \to \mathbb{R}$ lower semi-continuous and bounded, then Kantorovich problem admits a solution $\gamma^* \in \mathcal{A}dm(f^+, f^-)$.*

The proof utilizes the direct method of the calculus of variations, it can be found in details in [36].

1.2 Kantorovich Dual Problem

The Kantorovich dual problem [45, 36] arises from the primal optimal transport problem, it states: Given two non-negative finite measures f^+ and f^- on X and Y satisfying $f^+(X) = f^-(Y)$, and given a lower semi-continuous and bounded cost function $c : X \times Y \to \mathbb{R}$. We can define the set $\mathcal{L}_c$ to be the set of all measurable functions $(u, v) \in L^1(\, df^+) \times L^1(\, df^-)$ satisfying

$$u(x) + v(y) \leq c(x, y) \tag{1.3}$$

Find $(u^*, v^*) \in \mathcal{L}_c$ that solves the maximization problem

$$\sup_{(u,v)\in\mathcal{L}_c} \mathbb{I}[u,v] := \int_X u(x)\, df^+(x) + \int_Y v(y)\, df^-(y) \tag{1.4}$$

And we have the following duality theorem:

Theorem 2 *(Kanorovich Duality) Given two non-negative finite measures f^+ and f^- on X and Y satisfying $f^+(X) = f^-(Y)$, and a cost function $c : X \times Y \to \mathbb{R}$ lower semi-continuous and bounded, the following equality holds*

$$\min_{\gamma\in\mathcal{A}dm(f^+,f^-)} \mathbb{K}(\gamma) = \max_{(u,v)\in\mathcal{L}_c} \mathbb{I}[u,v]$$

The complete proof of the statement can be found in [45].

We give the following definition that is necessary for the existence result below

Definition 1 *Given a function $\chi : X \to \overline{\mathbb{R}}$ we define its $c-$transform $\chi^c : Y \to \overline{\mathbb{R}}$ by*

$$\chi^c(y) = \inf_{x\in X} c(x,y) - \chi(x)$$

We also define the $\overline{c}-$transform of $\zeta^{\overline{c}} : Y \to \overline{\mathbb{R}}$ by

$$\zeta^{\overline{c}}(x) = \inf_{y\in Y} c(x,y) - \zeta(y)$$

We say a function v defined on Y is $\overline{c}-$concave if there exists χ such that $v = \chi^c$, similarly, a function u on X is said to be $c-$concave if there is ζ such that $u = \zeta^{\overline{c}}$. And we denote by $c - conc(X)$ and $\overline{c} - conc(Y)$ the sets of $c-$ and $\overline{c}-$concave functions, respectively [36].

Realizing that, within the Dual problem, it is straightforward to observe that for any given pair (u, v), it is always possible to substitute it with (u, u^c), followed by (u^{cc}, u^c), while maintaining the constraints and increasing the integrals. In fact, this substitution process can be extended indefinitely; however, we can easily establish that $u^{ccc} = u^c$ for any u [See section 1.6 in [36]].

Remark 1 *As we are interested in the case where $c(x,y) = |x-y|$, it is worth to mention this property about the c-transform of 1-Lipschitz functions, if f is 1-Lipschitz, then $f^c = -f$ ([44], sec. 5).*

A consequence of these considerations is the following existence result

Proposition 1 *Assume that X, Y are compact and the function c is continuous then the dual problem admits a solution pair $(u^*, v^*) \in \mathcal{L}_c$ with $v^* = (u^*)^c \in \overline{c} - conc(Y)$. This means that the dual problem can be rewritten as:*

$$\sup_{u\in c-conc(X)} \int_X u(x)\, df^+(x) + \int_Y (u^c)\, df^-(y)$$

The function u^* is called Kantorovich Potential.

You can find a detailed proof in [36].

1.2.1 Existence of Optimal Transport map

Below, we present a general result, as stated in [36], which guarantees both the uniqueness of an optimal transport plan, solving the Kantorovich Primal Problem, and the existence of an optimal transport map for the Monge Problem.

Theorem 3 *Consider a compact domain $\Omega \subset \mathbb{R}^d$, two balanced measures $f^+, f^- \in \mathcal{M}_+(\Omega)$, such that $\partial\Omega$ is f^+-negligible, and f^+ is absolutely continuous with respect to the Lebesgue measure. Assume that the transport cost is of the form $c(x,y) = h(|x-y|)$ with h a strictly convex function, then there exists a unique transport plan $\gamma^* \in \mathcal{A}dm(f^+, f^-)$ of the form $\gamma^* = (Id, T^*)_{\#} f^+$, with T^* transport map. Moreover, there exists a Kantorovich potential u and T^* satisfies the following relation:*

$$T^*(x) = x - (\nabla h)^{-1}(\nabla(u(x)))$$

Remark 2 *All the costs of the form $c(x,y) = |x-y|^p$ with $p > 1$ can be dealt with via Theorem 3, and it is a particular problem that has been extensively studied in the field. However, this theorem cannot be applied to the case when the cost function is $c(x,y) = |x-y|$, which corresponds to the original Monge Problem, and also of our main interest in this thesis. And the reason for this limitation lies in the essential role played by the strict convexity of function h in the proof of the aforementioned theorem.*

1.2.2 L^1 Optimal Transport ($c(x,y) = |x-y|$)

The mathematical theory of L^1 optimal transport (L^1-OTP) is rich and extensive, and in this thesis it is of our main concern. We will restrict to the case $X = Y = \Omega$, where $\Omega \subset \mathbb{R}^d$ is an open, bounded, connected, and convex domain with smooth boundary.

Theorem 4 *(Kantorovich-Rubinstein Theorem) Consider $\Omega \subset \mathbb{R}^d$ an open, bounded, connected, and convex domain with smooth boundary. Take two nonnegative measures f^+ and f^- on Ω such that $df^+(\Omega) = df^-(\Omega)$. Let $\mathbb{K}_1$ denote Kantorovich problem with cost $c(x,y) = |x-y|$, i.e.*

$$\mathbb{K}_1(\gamma) = \inf_{\gamma \in \mathcal{A}dm(f^+, f^-)} \int_{X \times X} |x-y| \, d\gamma(x,y)$$

Let $Lip_1(\Omega)$ denote the space of all 1-Lipschitz functions on Ω, then Kantorovich problem can be written as find $u \in Lip_1(\Omega)$ that solves

$$\sup_{u \in Lip_1(\Omega)} \int_\Omega u \, df \tag{1.5}$$

where $f = f^+ - f^-$.

The proof can be found in [45].

Now, we introduce a minimization problem, it is referred to as *Beckmann Problem*, which, as demonstrated in the proposition immediately following, is found to be equivalent to the problem presented in equation (1.5). And this fact plays an essential role in our future work

Beckmann Problem: [36]

Consider $\Omega \subset \mathbb{R}^d$ an open, bounded, connected, and convex domain with smooth boundary. Take two nonnegative measures f^+ and f^- on Ω such that $df^+(\Omega) = df^-(\Omega)$. Find $v^* \in [\mathcal{M}(\Omega)]^d$ solving

$$\inf_{v \in [\mathcal{M}(\Omega)]^d} \left\{ \int_\Omega |v| : \operatorname{div}(v) = f \right\} \tag{1.6}$$

where $f = f^+ - f^-$, and the divergence constraint on v is the sense of distributions.

Proposition 2 *Consider $\Omega \subset \mathbb{R}^d$ an open, bounded, connected, and convex domain with smooth boundary. Take two nonnegative measures f^+ and f^- on Ω such that $df^+(\Omega) = df^-(\Omega)$. Then Beckmann Problem and the problem in 1.5 are equivalent, i.e.*

$$\sup_{u \in Lip_1(\Omega)} \int_\Omega u \, df = \inf_{v \in [\mathcal{M}(\Omega)]^d} \left\{ \int_\Omega |v| : div(v) = f \right\}$$

with $f = f^+ - f^-$.

1.3 Wasserstein Distance

The Wasserstein distance, also called Earth Mover's Distance (EM) or Kantorovich-Rubinstein distance, is a metric used to measure the difference between two probability distributions. Unlike traditional metrics, such as Euclidean distance or Kullback-Leibler divergence, it considers both distribution values and spatial relationships between elements. This is valuable for comparing distributions representing mass, probability, etc. In the context of optimal transport theory, it finds the most efficient way to transform one distribution into another while minimizing transportation cost. We mainly consider costs of the form $c(x, y) = |x - y|^p$ in $\Omega \subset \mathbb{R}^d$.

Here we give the basic definitions of Wasserstein distances and spaces in $\Omega \subset \mathbb{R}^d$ [36]. When Ω is unbounded we need to restrict to the following set of probabilities, we denote by $\mathcal{P}(\Omega)$ the set of probability measures on Ω, which we refer to as the *Wasserstein space* of order p

$$\mathcal{P}_p(\Omega) := \left\{ \mu \in \mathcal{P}(\Omega) : \int_\Omega |x|^p \, d\mu < +\infty \right\}$$

For $f^+, f^- \in \mathcal{P}_p(\Omega)$, let us define

$$W_p(f^+, f^-) := \min \left\{ \int_{\Omega\times\Omega} |x - y|^p \, d\gamma : \gamma \in \mathcal{A}dm(f^+, f^-) \right\}^{\frac{1}{p}}, \tag{1.7}$$

i.e. the p-th root of the minimal transport cost, for the cost $c(x, y) = |x - y|^p$.

Proposition 3 *The quantity W_p defined above is a distance over $\mathcal{P}_p(\Omega)$.*

Through out this thesis we are interested W_1 distance, the case where $c(x, y) = |x - y|$

$$W_1(f^+, f^-) := \min \int_{\Omega\times\Omega} |x - y| \, d\gamma \tag{1.8}$$

1.3.1 Properties of Wasserstein distance

As in [44], we will present here the most important properties and features of Wasserstein distance. From the duality results (Prop. 1) and the theory of L^1-Optimal Transport, we get the following useful duality formula for W_1 distance

$$W_1(f^+, f^-) = \sup_{||u||_{Lip}\leq 1} \int_X u(x) \, df^+(x) - \int_Y u \, df^-(y) \tag{1.9}$$

Remark 3 *Using Hölder inequality we can show that*

$$p \leq q \implies W_p \leq W_q.$$

In Particular [44],

- *W_1 distance is the weakest of them all.*
- *In general, W_1 distance is more flexible and easier to bound,*
- *W_2 reflects more geometric features.*
- *Results in W_2 are stronger and more difficult to establish than results for W_1 distance.*

Convergence in Wasserstein sense, we will use the notation $\mu_k \to \mu$ to mean that μ_k converges weakly to μ (*i.e.* $\int \phi \, d\mu_k \to \int \phi \, d\mu$, for any bounded continuous map ϕ).

Remark 4 *By Polish spaces we mean, complete, separable metric spaces.*

Definition 2 ***(Weak Convergence in $\mathcal{P}_p$)***
Let $(\mathcal{X}, d)$ be a Polish space, and $p \in [1, \infty)$. Let (μ_k) be a sequence of probability measures and μ in $\mathcal{P}_p(X)$. Then $\mu_k \to \mu$ (weakly) if any of the following equivalent properties is satisfied for any $x_0 \in \mathcal{X}$

- $\mu_k \to \mu$ *and* $\int d(x, x_0)^p \, d\mu_k(x) \to \int d(x, x_0)^p \, d\mu(x)$;
- $\mu_k \to \mu$ *and* $\limsup_{k\to\infty} \int d(x, x_0)^p \, d\mu_k(x) \leq \int d(x, x_0)^p \, d\mu(x)$;
- $\mu_k \to \mu$ *and* $\lim_{R\to\infty} \limsup_{k\to\infty} \int_{d(x,x_0)\geq R} d(x, x_0)^p \, d\mu_k(x) = 0$;
- *For all continuous functions ϕ with $|\phi(x)| \leq C(1 + d(x, x_0)^p)$ $C \in \mathbb{R}$ we have*

$$\int \phi(x) \, d\mu_k(x) \to \int \phi(x) \, d\mu(x).$$

The following results are taken from [44] where you find detailed proofs.

Theorem 5 ***(W_p metrizes $\mathcal{P}_p$)***
Let $(\mathcal{X}, d)$ be a Polish space, and $p \in [1, \infty)$; then W_p metrizes the weak convergence in $\mathcal{P}_p(\mathcal{X})$, i.e. if a sequence (μ_k) and μ in $\mathcal{P}_p$, then the statements

$$\mu_k \to \mu$$

and

$$W_p(\mu_k, \mu) \to 0$$

are equivalent.

An immediate corollary follows the theorem above

Corollary 1 ***(Continuity of W_p)***
Let $(\mathcal{X}, d)$ be a Polish space, and $p \in [1, \infty)$; then W_p is continuous on $\mathcal{P}_p(\mathcal{X})$. More preciesly, if μ_k(resp. ν_k) $\to \mu$ (resp. ν), then

$$W_p(\mu_k, \nu_k) \to W_p(\mu, \nu)$$

There are several reasons of the preference of using Wasserstein distance over other ways to metrize weak convergence, for which we refer to [44]

1.4 Monge Kantorovich equations

Before we can dive into Monge Kantorovich equations, we need to define a quantity called *Optimal Transport Density*, we give the definition as stated in [15]

Definition 3 *(Optimal Transport Density) Consider $\Omega \subset \mathbb{R}^d$ an open, bounded, connected, and convex domain with smooth boundary. Take two nonnegative measures f^+ and f^- on Ω such that $df^+(\Omega) = df^-(\Omega)$. Given $\gamma^* \in \mathcal{A}dm(f^+, f^-)$ a minimizer for Kantorovich Problem in (1.2) with cost $c(x, y) = |x - y|$, the Optimal Transport Density (OT density) $\mu^* \in \mathcal{M}_+(\Omega)$ associated to f^+, f^- is defined as*

$$\langle \mu^*, \phi \rangle := \int_{\Omega\times\Omega} \int_0^1 |\omega'_{x,y}(t)| \phi(\omega_{x,y}(t)) \, dt \, d\gamma(x, y) \qquad \forall \phi \in \mathcal{C}(\Omega) \tag{1.10}$$

where

$$\omega_{x,y}(t) = (1 - t)x + ty$$

In [1, 20, 35] you can find all the results that assure the uniqueness and continuity of the density μ.

In the following proposition we can see the importance of the OT density in the theory of L^1-OTP

Proposition 4 *Consider $\Omega \subset \mathbb{R}^d$ an open, bounded, connected, and convex domain with smooth boundary. Take two nonnegative measures f^+ and f^- on Ω such that $df^+(\Omega) = df^-(\Omega)$. If f^+ and f^- admit L^p-densities with $1 \leq p \leq +\infty$, a solution v^* of the Beckmann Problem belongs to $[L^p(\Omega)]^d$ and it can be written as*

$$v^* = -\mu^* \nabla u^* \tag{1.11}$$

where μ^* is OT density $\mu^*(f^+, f^-)$ and u^* is solution of the Dual Kantorovich Problem.

We now come to the most interesting finding of L^1-OTP, which asserts that the optimal transport density can be characterized by the a system of equations, known as the Monge-Kantorovich equations (MK equations). Authors in [14, 13] illustrate that by examining the p-Laplacian equation below, it is possible to construct a solution for the classical of the Monge-Kantorovich problem

$$-\text{div}(|Du_p|^{p-2} Du_p) = f^+ - f^- \tag{1.12}$$

as p approaches infinity, the objective is to demonstrate the convergence of u_p towards u, where u is a solution that satisfies

$$|Du| \leq 1, \quad -\text{div}(aDu) = f^+ - f^- \tag{1.13}$$

with a non-negative density a, the goal is to construct a flow by solving an ordinary differential equation that incorporates a, Du, f^+, and f^-. So here comes the following proposition of Monge-Kantorovich Equations

Proposition 5 *(Monge-Kantorovich Equations) Consider $\Omega \subset \mathbb{R}^d$ an open, bounded, connected, and convex domain with smooth boundary. Take two nonnegative measures f^+ and f^- on Ω such that $df^+(\Omega) = df^-(\Omega)$. Assume that f^+ and f^- admit L^p-densities. The OT density $\mu^*(f^+, f^-)$ and Kantorovich potential u^* solve the following equations*

$$-div(\mu^* \nabla u^*) = f \quad \text{in } \Omega \tag{1.14}$$

$$|\nabla u^*| \leq 1 \quad \text{in } \Omega \tag{1.15}$$

$$|\nabla u^*| = 1 \quad \text{a.e. in } \mu^* > 0 \tag{1.16}$$

with $f = f^+ - f^-$

Following the approach done in [14], here we present the solution of the Monge Problem

Proposition 6 *Consider $\Omega \subset \mathbb{R}^d$ an open, bounded, connected, and convex domain with smooth boundary. Take two nonnegative measures f^+ and f^- on Ω such that $df^+(\Omega) = df^-(\Omega)$. Suppose that the probability densities of both f^+ and f^- are Lipschitz continuous with respect to the Lebesgue measure. For an arbitrary point x from the support of f^+, consider the solution $z(t, x)$ to the following ordinary differential equation*

$$z'(t) = Z(t, z(t)), \qquad Z(t, z) = \frac{-\mu^*(z) \nabla u^*(z)}{(1-t) f^+(z) + t f^-(z)}$$

with initial condition $z(0) = x$. Then the map T^* defined a s

$$T^*(x) := z(1, x)$$

where $T^* : supp(f^+) \to supp(f^-)$, it is the solution of the Monge Problem with cost equal to the Euclidean distance.

1.5 Dynamic Monge-Kantorovich

In this section, we present the work of the authors from [15] introduce a novel dynamical formulation of L^1-OTP. It is an extension to a continuous model that describes the dynamics of a slime mold called Physarum Polycephalum (PP). The original model represents PP's behavior on a finite-dimensional planar graph, imitating its ability to find the shortest path between two food sources in a maze using a pipe-flow analogy as in [41]. Furthermore, the same authors establish that the model describes the PP's behavior is equivalent to an optimal transport problem within the framework of the graph.

Here is the new model as described in [15, 16], by omitting the graph structure and defining the problem on an open bounded domain $\Omega \subset \mathbb{R}^d$. The focus is specifically on f^+ and f^- continuous with respect to the Lebesgue measure, denote their densities as f^+ and f^- respectively. The problem they introduced is as following: find the pair $(\mu, u) : [0, +\infty) \times \Omega \times \Omega \mapsto \mathbb{R}^+ \times \mathbb{R}$ that solves

$$-\text{div}\left(\mu(t,x)\nabla u(t,x)\right) = f(x) = f^+(x) - f^-(x) \tag{1.17}$$

$$\partial_t \mu(t,x) = \mu(t,x)|\nabla u(t,x)| - \mu(t,x) \tag{1.18}$$

$$\mu(0,x) = \mu_0(x) > 0 \tag{1.19}$$

with zero-Neumann boundary conditions. It is conjectured that the new system of equations reaches a time-asymptotic equilibrium, and this equilibrium point corresponds to the solution of Monge-Kantorovich partial differential equations. While a complete proof of the conjecture is not provided, the article offers theoretical and numerical evidence supporting their thesis. Numerical simulations using finite elements and backward Euler time stepping confirm that the approximate solution of the transportation problem converges at large times to an equilibrium configuration, closely matching the numerical solution of the Monge-Kantorovich equations. Moreover, the authors find out that, using the proposed dynamic model to solve the Monge-Kantorovich equations offers numerical simplicity compared to directly solving the classical Monge-Kantorovich equations. This indicates that introducing time into the equation alleviates numerical challenges, leading to a more robust, flexible, and efficient approach for solving OT problems.

The existence and uniqueness results in [15] is due to the local nature of Lipschitz continuity, and it becomes difficult to extend the results to larger time intervals. To overcome this limitation and still understand the long-term behavior of the model, the authors defined a Lyapunov functions which has a strictly negative derivative along the solution trajectory, indicating stability [18]. It is defined for general $\mu \in L^1(\Omega)$ and given by

$$\mathcal{L}(\mu) := \mathcal{E}_f(\mu) + \mathcal{M}(\mu) \tag{1.20}$$

$$\mathcal{E}_f(\mu) := \sup_{\phi \in \mathcal{C}^1(\overline{\Omega})} \int_\Omega \left(f\phi - \mu \frac{|\nabla \phi|^2}{2} \right) dx \qquad \mathcal{M}(\mu) := \frac{1}{2}\int_\Omega \mu \, dx \tag{1.21}$$

The energy functional $\mathcal{E}_f(\mu)$ is written in variational form, and, under the assumption $\mu \in \mathcal{C}^\delta(\Omega)$ and $f \in L^\infty(\Omega)$, is equivalent to

$$\mathcal{E}_f(\mu) = \frac{1}{2}\int_\Omega \mu |\nabla u(\mu)|^2 \, dx$$

where u is the solution of (1.17). Hence we rewrite $\mathcal{L}$ as

$$\mathcal{L}(\mu) = \frac{1}{2}\int_\Omega \mu |\nabla u(\mu)|^2 \, dx + \frac{1}{2}\int_\Omega \mu \, dx$$

In [17], authors prove the following equality

$$\mathcal{E}_f(\mu) = \sup_{\phi \in Lip(\Omega)} \int_\Omega \left(f\phi - \mu \frac{|\nabla \phi|^2}{2} \right) dx \tag{1.22}$$

$$= \inf_{w \in [L^2_\mu(\Omega)]^d} \left\{ \int_\Omega \frac{|w|^2}{2} \mu \, dx : -\mathrm{div}(\mu w) = f \right\} \tag{1.23}$$

The following Proposition stated in [15], shows the equivalence between the minimization of Lyapunov-candidate functional $\mathcal{L}$ and the Beckmann Problem which is equivalent to solve the Monge-Kantorovich equations by Proposition 2

Proposition 7 *Given Ω an open, bounded, convex, and connected domain in $\mathbb{R}^d$ with smooth boundary. Consider $f \in L^1(\Omega)$ with zero mean, then Beckmann Problem (in (1.6)) and the minimization of $\mathcal{L}$ are equivalent, which means*

$$\min_{v \in [L^1(\Omega)]^d} \left\{ \int_\Omega |v| \, dx : div(v) = f \right\} = \min_{\mu \in L^1_+(\Omega)} \mathcal{L}(\mu)$$

where $L^1_+(\Omega)$ is the space of the non-negative functions in $L^1(\Omega)$.

For the complete proof refer to Proposition 39 in [15].

Now we can state the following Proposition, which is our main interest in this thesis

Proposition 8 *Given $f = f^+ - f^- \in L^1(\Omega)$ with zero mean, then the OT density μ^* is a minimizer for $\mathcal{L}$ with value equal to the W_1 distance between f^+ and f^-, i.e. $\mathcal{L}(\mu^*) = W_1(f^+, f^-)$*

Numerical results in [15, 18] show that the proposed scheme is highly efficient and accurate in calculating Wasserstein-1 distances.

Chapter 2

An Accurate Calculation of Wasserstein-1 distance

Our work focus is on creating a highly efficient and accurate Dynamic Monge-Kantorovich (DMK) solver. As a first step, we studied the *energy functional* introduced in [19], known as the "L^1 transport energy", importantly, they prove that its unique minimizer is the optimal transport density μ^*. Our goal here is to design, test and study an approximation algorithm for μ^*, based on the so-called *discretize-minimize approach*. The idea is to discretize the transport energy using a truncated Fourier (or cosine) series, and to minimize the resulting discrete functionals.

2.1 Transport Energy

We define the L^1 transport energy functional as follow: let Ω be a bounded Lipschitz domain of $\mathbb{R}^d$ and let $f \in L^\infty(\Omega)$ be such that $\text{conv}(\text{supp}\, f) \subset\subset \Omega$ and $\int_\Omega f dx = 0$. We denote by $\mathcal{E} : \mathcal{M}(\Omega) \to]0, +\infty]$ the transport energy functional

$$\mathcal{E}(\mu) := \mathcal{D}(\mu) + \int_\Omega d\mu, \tag{2.1}$$

where

$$\mathcal{D}(\mu) := \sup_{\substack{u \in \mathscr{C}^1(\overline{\Omega}) \\ \int u dx = 0}} D(\mu, u) := \sup_{\substack{u \in \mathscr{C}^1(\overline{\Omega}) \\ \int u dx = 0}} \left(2 \int_\Omega f u \, dx - \int_\Omega |\nabla u|^2 d\mu \right). \tag{2.2}$$

Remark 5 *our interest of studying the minimization of $\mathcal{E}$ is because:*

- *it admits a unique minimizer μ^*, that is exactly the L^1* optimal transport density *[19],*
- *its negative minimum is exactly twice the minimum value of the Lyapunov candidate functional (1.20) and thus the Wasserstein-1 distance betwees f^+ and f^-, i.e.:*

$$-\mathcal{E}(\mu^*)/2 = \mathcal{L}(\mu^*) = W_1(f^+, f^-)$$

In the following the description will be confined in a one-dimensional setting ($\mathbb{R}^d$ with $d = 1$), i.e., $\Omega = (0, 1)$, keeping in mind that its extension to the two-dimensional interval $(0, 1) \times (0, 1)$ will be straight-forward.

2.1.1 Fourier Discretization

We start our description by denoting with $\phi_k := e^{2\pi i k x}$, $x \in \Omega$, the Fourier character of frequency $k \in \mathbb{Z}$, and by

$$S_N := \text{span}(\phi_{-N}, \ldots, \phi_0, \phi_1, \ldots, \phi_N),$$

the space of Fourier sums of order N, i.e., the space of expansions

$$\hat{f}(x) := \sum_{k=-N}^{N} \hat{f}_k \phi_k(x) \;\; x \in \Omega.$$

Here and in the following, we always denote as $\hat{f}_k$ the k-th Fourier coefficient of f, i.e., the k-th coefficient of $\hat{f}$. Since we will need to work with positive functions and with functions of zero mean, we additionally introduce the cone S_N^+ and the space S_N^0 as

$$\begin{aligned} S_N^+ &:= \{g \in S_N : \; g(x) \in \mathbb{R}, \; g(x) \geq 0, \; \forall x \in [0,1]\}, \\ S_N^0 &:= \operatorname{span}(\phi_{-N}, \ldots, \phi_{-1}, \phi_1, \ldots, \phi_N). \end{aligned}$$

Using these spaces, we can discretize the functional (2.2). Namely, for $N, M \in \mathbb{N}$ we require that $u \in S_N^0$ (i.e., $u \in S_N$ with $\int u dx = 0$), and $\mu \in S_M^+$ (i.e., $\mu \in S_M$ and $\mu(x) \geq 0$ for all $x \in \Omega$). This results in the following discrete functionals:

$$\mathcal{E}_{M,N}(\mu) := \begin{cases} \mathcal{D}_{M,N}(\mu) + \hat{\mu}_0, & \mu \in S_M^+ \\ +\infty, & \text{otherwise} \end{cases} \tag{2.3}$$

and

$$\mathcal{D}_{M,N}(\mu) := \sup_{u \in S_N^0} D(\mu, u) := \max_{u \in S_N^0} \int_\Omega (2fu - \mu |\nabla u|^2) dx. \tag{2.4}$$

From the observations we should necessarily assume $M \leq 2N$ in order to get a strictly convex functional $\tilde{E}_{M,N}$.

Recall that $\nabla u = \sum_{j=-N}^{N} 2\pi i j \hat{u}_j \phi_j$ $(j \neq 0)$, then

$$\begin{aligned} \int_0^1 \mu |\nabla u|^2 \, dx &= \sum_{k=-M}^{M} \hat{\mu}_k \sum_{\substack{j,l=-N \\ j,l \neq 0}}^{N} 4\pi^2 \hat{u}_j \overline{\hat{u}_l} \int_0^1 \phi_{k+j-l} \, dx \\ &= 4\pi^2 \sum_{\substack{j,l=-N \\ j,l \neq 0}}^{N} \hat{u}_j j l \overline{\hat{u}_l} \sum_{k=-M}^{M} \hat{\mu}_k \delta_{k+j-l}(0) \\ &= 4\pi^2 \hat{u}^H A(\hat{\mu}) \hat{u} \end{aligned}$$

where

$$A(\hat{\mu}) = D(\hat{\mu}) T(\hat{\mu}) D \tag{2.5}$$

$$D = diag(-N, \ldots, -1, 1, \ldots, N) \tag{2.6}$$

$$T(\hat{\mu}) = \text{Toep}\left(\hat{\mu}_{-M}, \ldots, \hat{\mu}_M\right), \tag{2.7}$$

and $\text{Toep}(v)$ is the Toeplitz matrix generated by the vector v.

We can re-write

$$\mathcal{D}_{\hat{\mu}}(\hat{u}) = 2\langle \hat{f}, \hat{u} \rangle - 4\pi^2 \hat{u}^H A(\hat{\mu}) \hat{u}$$

After deriving the right hand side of the above equation with respect to $\hat{u}$, we would like to conclude that, possibly under further assumption on $\hat{\mu}$, we have

$$\sup_{\hat{u} \in S_N^0} \mathcal{D}(\hat{u}) = \max_{\hat{u} \in S_N^0} \mathcal{D}(\hat{u}) = \mathcal{D}(\hat{\mu}, \hat{u}_{\hat{\mu}}),$$

where

$$\hat{u}_{\hat{\mu}} := \frac{1}{4\pi^2} (A(\hat{\mu}))^{-1} \hat{f}$$

Hence, we can write

$$\sup_{\hat{u} \in S_N^0} \mathcal{D}(\hat{u}) = \mathcal{D}(\hat{\mu}, \hat{u}_{\hat{\mu}}) = \frac{1}{4\pi^2} \hat{f}^H (A(\hat{\mu}))^{-1} \hat{f}$$

Definition 4 (Fourier discretized transport energy) *Let $\Omega =]0,1[^d$. Let $f \in L^\infty(\Omega, \mathbb{R})$ be such that* $\operatorname{conv}\operatorname{supp} f \subset\subset \Omega$ *and $\int_\Omega f dx = 0$. Let $\delta_N > 0$ with $\delta_N \downarrow 0$ as $N \to +\infty$. Then we define the Fourier (M,N)-discretized transport energy by setting*

$$\tilde{E}_{M,N}(\underline{\hat{\mu}}) := (D^{-1}\hat{f})^H T(\hat{\mu}_\delta)^{-1}(D^{-1}\hat{f}) + \hat{\mu}_0. \tag{2.8}$$

with $\hat{\mu}_\delta := (\dots, \hat{\mu}_{-1}, \hat{\mu}_0 + \delta, \hat{\mu}_1, \dots)$

2.2 Implementation

We can proceed now to describe the minimization algorithm. It is based on the variational Euler discretization of the following gradient flow:

$$\begin{cases} \frac{d}{dt}\underline{\hat{\mu}} = -\nabla \tilde{E}_{M,N}(\underline{\hat{\mu}}) \quad \forall t > 0 \\ \underline{\hat{\mu}}(0) = \underline{\hat{\mu}^0} \end{cases} \tag{2.9}$$

In this case, the variational Euler method coincides with the Implicit-Euler discretization of the above ODE, i.e.:

$$\begin{cases} \underline{\hat{\mu}}^{k+1} = \underline{\hat{\mu}}^k - \tau \nabla \tilde{E}_{M,N}(\underline{\hat{\mu}}^{k+1}) \quad k = 0, 1, \dots \\ \underline{\hat{\mu}}(0) = \underline{\hat{\mu}^0} \end{cases} \tag{2.10}$$

where τ is the time-step size. The implicit Euler scheme thus defined requires at each time-step the solution of a nonlinear system of algebraic equations. This is done by means of Newton method. To this aim we need to calculate both the Gradient and the Hessian of the Fourier-discrete energy functional $\tilde{E}_{M,N}(\mu)$. The calculations for these quantities as well as the implementation of the positivity constraint for $\underline{\hat{\mu}}$ are described in the next section.

Note that, because of the analyticity of the discrete energy, the gradient flow converges exponentially in time to the equilibrium solution for a fixed Euler time-step size τ. This implies that in principle we could increase τ as well as time progresses. This however cannot be done as it would prevent convergence of the Newton scheme. Thus we proceed by defining an algorithm where the τ sequence increases geometrically. It can be shown that, under mild hypothesis, this is sufficient to guarantee superlinear convergence of the overall algorithm, as also evidenced in the reported numerical results.

2.2.1 Calculation of the Gradient and the Hessian of the energy functional

In the following we will use different complex and real vectors w of different dimensions, usually indexed with positive and negative indices. We will use the convention that $w_j := 0$ if j is outside of the range of indices where w is defined.

Let $M, N \in \mathbb{N}$, $M \leq 2N-1$, and $\hat{\mu} := [\hat{\mu}_{-M}, \dots, \mu_0, \dots, \hat{\mu}_M]^T \in \mathbb{C}^{2M+1}$. The energy $E : \mathbb{C}^{2M+1} \to \mathbb{R}$ can be written as:

$$\begin{aligned} E(\hat{\mu}) := E_{M,N}(\hat{\mu}) &= \hat{f}^H A^{-1}(\hat{\mu}_\delta)\hat{f} + \hat{\mu}_0 \\ &= \hat{f}^H D^{-1} T^{-1}(\hat{\mu}_\delta) D^{-1}\hat{f} + \hat{\mu}_0 \\ &= v^H T^{-1}(\hat{\mu}_\delta) v + \hat{\mu}_0, \end{aligned}$$

where

$$v := D^{-1}\hat{f} = \left[-\frac{\hat{f}_{-N}}{N}, -\frac{\hat{f}_{-N+1}}{N-1}, \dots, \frac{\hat{f}_{N-1}}{N-1}, \frac{\hat{f}_N}{N} \right]^T \in \mathbb{C}^{2N}, \tag{2.11}$$

and where the matrix function $T : \mathbb{C}^{2M+1} \to \mathbb{C}^{2N\times 2N}$ is defined as

$$T(\hat{\mu}) := T_{M,N,\delta}(\hat{\mu}) := \begin{bmatrix} \hat{\mu}_0 + \delta & \hat{\mu}_1 & \dots & \hat{\mu}_{2N-1} \\ \hat{\mu}_{-1} & \ddots & & \vdots \\ \vdots & & \ddots & \hat{\mu}_1 \\ \hat{\mu}_{-2N+1} & \dots & \hat{\mu}_{-1} & \hat{\mu}_0 + \delta \end{bmatrix},$$

with the assumption that $\hat{\mu}_k := 0$ for $-2N \le k \le -M-1$ and $M+1 \le k \le 2N$ if $M+1 < 2N$.

Gradient For $-2N \le k \le 2N$, we define $T_k \in \mathbb{R}^{2N\times 2N}$ to be the matrix with all zero entries except for the k-th diagonal, with positive indices denoting the upper diagonals and negative indices denoting lower diagonals. In particular $T_0 = I$ is the identity matrix. We recall moreover that for an invertible parametric matrix $A : \mathbb{R} \to \mathbb{R}^{2N\times 2N}$, we have that $\partial_\alpha(A^{-1}(\alpha)) = -A^{-1}(\alpha)\left(\partial_a A(\alpha)\right)A^{-1}(\alpha)$. With these facts in mind we have for $-M \le k \le M$ that

$$\begin{aligned} \partial_{\hat{\mu}_k} E(\hat{\mu}) &= v^H \partial_{\hat{\mu}_k} T^{-1}(\hat{\mu}) v + \delta_{k0} \\ &= -v^H T^{-1}(\hat{\mu}) \left(\partial_{\hat{\mu}_k} T(\hat{\mu})\right)\ T^{-1}(\hat{\mu}) v + \delta_{k0} \\ &= -v^H T^{-1}(\hat{\mu}) T_k\ T^{-1}(\hat{\mu}) v + \delta_{k0}. \end{aligned} \tag{2.12}$$

Hessian For $0 \le h, k \le M$ we have

$$\begin{aligned} \partial_{\hat{\mu}_h}\partial_{\hat{\mu}_k} E(\hat{\mu}) &= \partial_{\hat{\mu}_h}\left(-v^H T^{-1}(\hat{\mu}) T_k\ T^{-1}(\hat{\mu}) v + \delta_{k0}\right) \\ &= -v^H \partial_{\hat{\mu}_h}\left(T^{-1}(\hat{\mu}) T_k\ T^{-1}(\hat{\mu})\right) v, \end{aligned}$$

and

$$\begin{aligned} &\partial_{\hat{\mu}_h}\left(T^{-1}(\hat{\mu}) T_k\ T^{-1}(\hat{\mu})\right) \\ &\quad = \left(\partial_{\hat{\mu}_h} T^{-1}(\hat{\mu})\right) T_k T^{-1}(\hat{\mu}) + T^{-1}(\hat{\mu})\left(\partial_{\hat{\mu}_h} T_k\right) T^{-1}(\hat{\mu}) + T^{-1}(\hat{\mu}) T_k \left(\partial_{\hat{\mu}_h} T^{-1}(\hat{\mu})\right) \\ &\quad = -T^{-1}(\hat{\mu}) T_h T^{-1}(\hat{\mu}) T_k T^{-1}(\hat{\mu}) - T^{-1}(\hat{\mu}) T_k T^{-1}(\hat{\mu}) T_h T^{-1}(\hat{\mu}) \\ &\quad = -T^{-1}(\hat{\mu})\left(T_h T^{-1}(\hat{\mu}) T_k + T_k T^{-1}(\hat{\mu}) T_h\right) T^{-1}(\hat{\mu}). \end{aligned}$$

Thus

$$\partial_{\hat{\mu}_h}\partial_{\hat{\mu}_k} E(\hat{\mu}) = v^H T^{-1}(\hat{\mu})\left(T_h T^{-1}(\hat{\mu}) T_k + T_k T^{-1}(\hat{\mu}) T_h\right) T^{-1}(\hat{\mu}) v. \tag{2.13}$$

2.2.2 Efficient computation for hermitian $\hat{\mu}$

In the following we will work with a vector $\hat{\mu}$ which satisfies $\hat{\mu}_{-k} = \overline{\hat{\mu}_k}$ for $1 \le k \le M$, and $\hat{\mu}_0 \in \mathbb{R}$. In this case $T^{-1}(\hat{\mu})$ is hermitian (i.e., $T^{-1}(\hat{\mu}) = \left(T^{-1}(\hat{\mu})\right)^H$), and in this case the formulas (2.12) and (2.13) can be simplified.

Under this assumption, setting $w := T^{-1}(\hat{\mu}) v$ implies that (2.12) can be written as

$$\partial_{\hat{\mu}_k} E(\hat{\mu}) = -w^H T_k w + \delta_{k0}, \tag{2.14}$$

while (2.13) becomes

$$\partial_{\hat{\mu}_h}\partial_{\hat{\mu}_k} E(\hat{\mu}) = w^H\left(T_h T^{-1}(\hat{\mu}) T_k + T_k T^{-1}(\hat{\mu}) T_h\right) w.$$

Moreover, we can simplify the multiplications involving the matrices T_k. Indeed, for any vector $w = [w_{-N}, \dots, w_N]^T \in \mathbb{C}^{2N}$ and for all $-N \le k \le N$ we have that

$$T_k w = \begin{cases} (w_{-N+k}, w_{-N+k+1}, \dots, w_N, 0, \dots, 0)^T, & k \ge 0 \\ (0, \dots, 0, w_{-N}, w_{-N+1}, \dots, w_{N-k},)^T, & k < 0 \end{cases} =: (w_{-N+k}, \dots, w_{N+k})^T,$$

where in the last equation we used again the convention that $w_j = 0$ if $j < -N$ or $j > N$. In other words, multiplication by T_k performs a shift of k steps in the corresponding direction.

Moreover, we define the operation flip : $\mathbb{C}^{2N} \to \mathbb{C}^{2N}$ by $\text{flip}(w)_\ell := w_{-\ell}$, $N \le \ell \le N$.
It follows that

$$\begin{aligned} w^H T_k w &= \sum_{j=-N}^{N} \overline{w_j}(T_k w)_j = \sum_{j=-N}^{N} \overline{w_j} w_{j+k} = \sum_{j=N}^{-N} \overline{w_{-j}} w_{-j+k} \\ &= \sum_{j=N}^{-N} \overline{\text{flip}(w)_j} w_{-j+k} = (\overline{\text{flip}(w)} * w)_k. \end{aligned}$$

Substituting this formula into (2.14), we get

$$\begin{aligned} \nabla_{\hat{\mu}} E(\hat{\mu}) &= \begin{bmatrix} \partial_{\hat{\mu}_{-M}} E(\hat{\mu}) \\ \vdots \\ \partial_{\hat{\mu}_0} E(\hat{\mu}) \\ \vdots \\ \partial_{\hat{\mu}_M} E(\hat{\mu}) \end{bmatrix} = - \begin{bmatrix} (\overline{\text{flip}(w)} * w)_{-M} \\ \vdots \\ (\overline{\text{flip}(w)} * w)_0 - 1 \\ \vdots \\ (\overline{\text{flip}(w)} * w)_M \end{bmatrix} \\ &= e_0 - (\overline{\text{flip}(w)} * w)_{-M \le j \le M}, \end{aligned} \tag{2.15}$$

where $e_0 \in \mathbb{R}^{2M+1}$ is the 0-the cardinal vector, i.e., $(e_0)_k = \delta_{k0}$, and the subscript means that we select only the elements of the vector with indices $-M \le j \le M$, and $w = T^{-1}(\hat{\mu})v$.

For the Hessian, let $T^{-1} = \{t_{ij}\}_{i,j=-N}^{N}$

$$\left(T^{-1}(T_k w)\right)_i = \sum_{j=-N}^{N} t_{ij}(T_k w)_j = \sum_{j=-N}^{N} t_{ij} w_{k+j}$$

$$\begin{aligned} (T_{-h} w) T^{-1}(T_k w) &= \sum_{l=-N}^{N} (T_{-h} w)_l \sum_{j=-N}^{N} t_{ij}(T_k w)_j = \sum_{l=-N}^{N} w_{-h+l} \sum_{j=-N}^{N} t_{ij} w_{k+j} \\ &= \sum_{l=-N}^{N} \sum_{j=-N}^{N} w_{-h+l} t_{ij} w_{k+j}. \end{aligned}$$

Then

$$\partial_{\hat{\mu}_h} \partial_{\hat{\mu}_k} E(\hat{\mu}) = \sum_{l=-N}^{N} \sum_{j=-N}^{N} [w_{-h+l} t_{ij} w_{k+j} + w_{-k+l} t_{ij} w_{h+j}]$$

2.2.3 Positive μ

We want now to impose conditions to enforce μ to be a real- and positive-valued function. This can be obtained by requiring that there is a real valued function $\sigma : \mathbb{R} \to \mathbb{R}$ such that $\mu(x) = \sigma(x)^2$ for all $x \in \mathbb{R}$. This can be expressed in terms of conditions on the Fourier coefficients of μ and σ. Indeed, assuming that $\hat{\sigma} := [\hat{\sigma}_{-P}, \dots, \hat{\sigma}_P]^T \in \mathbb{C}^{2P+1}$, $P \in \mathbb{N}$, with

$$\sigma(x) := \sum_{j=-P}^{P} \hat{\sigma}_j \phi_j(x),$$

we require $\hat{\mu} = \hat{\sigma} * \hat{\sigma}$, i.e.,

$$\hat{\mu}_k = (\hat{\sigma} * \hat{\sigma})_k = \sum_{j=-P}^{P} \hat{\sigma}_{k-j} \hat{\sigma}_j, \quad -M \le k \le M,$$

so that $\mu(x) = \sigma(x)^2$ for all $x \in \mathbb{R}$. Observe that the last conditions requires in particular that $M = 2P$. Moreover, we require that $\sigma_0 \in \mathbb{R}$ and $\sigma_j = \overline{\sigma}_{-j}$ for $1 \leq j \leq P$, so that $\sigma(x) \in \mathbb{R}$ for all x. This condition can be expressed by introducing $\hat{x} \in \mathbb{R}^{P+1}$, $\hat{y} \in \mathbb{R}^P$ and defining

$$\hat{\sigma}_k := \begin{cases} \hat{x}_{-k} - i\hat{y}_{-k}, & -P \leq k \leq -1 \\ \hat{x}_0, & k = 0 \\ \hat{x}_k + i\hat{y}_k, & 1 \leq k \leq P \end{cases} \tag{2.16}$$

We will thus parametrize the problem in terms of $\hat{z} := (\hat{x}, \hat{y}) \in \mathbb{R}^{2P+1}$ instead of $\hat{\sigma} \in \mathbb{C}^{2P+1}$.

With this change of variable, problem (2.9) becomes

$$\begin{cases} \frac{d}{dt}\underline{\hat{\sigma}} = -\nabla \tilde{F}_{M,N}(\underline{\hat{\sigma}}) & \forall t > 0 \\ \underline{\hat{\sigma}}(0) = \underline{\hat{\sigma}^0} \end{cases} \tag{2.17}$$

where $F_{M,N}(\hat{\sigma}) := E_{M,N}(\hat{\sigma} * \hat{\sigma})$. The discretized equation (2.10) is now replaced by the Implicit Euler discretization of (2.17), at each time step Newton method is used to resolve the non linearity. For this reason the gradient and the Hessian of $F_{M,N}$ need to be calculated.

Gradient:

Using the chain rule, for all $0 \leq j \leq P$ we have

$$\partial_{\hat{x}_j} E(\hat{\sigma} * \hat{\sigma}) = \left(\partial_{\hat{x}_j}(\hat{\sigma} * \hat{\sigma})\right) \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}),$$

where

$$\partial_{\hat{x}_j}(\hat{\sigma} * \hat{\sigma}) = [\partial_{\hat{x}_j}(\hat{\sigma} * \hat{\sigma})_{-M}, \ldots, \partial_{\hat{x}_j}(\hat{\sigma} * \hat{\sigma})_M]^T \in \mathbb{C}^{2M+1}.$$

For $0 \leq j \leq P$ and $-M \leq k \leq M$ we have

$$\partial_{\hat{x}_j}(\hat{\sigma} * \hat{\sigma})_k = \partial_{\hat{x}_j} \sum_{\ell=-P}^{P} \hat{\sigma}_{k-\ell}\hat{\sigma}_\ell = \sum_{\ell=-P}^{P} \left(\partial_{\hat{x}_j}\hat{\sigma}_{k-\ell}\right)\hat{\sigma}_\ell + \hat{\sigma}_{k-\ell}\left(\partial_{\hat{x}_j}\hat{\sigma}_\ell\right). \tag{2.18}$$

Now $\partial_{\hat{x}_j}\hat{\sigma}_{k-\ell} = 1$ if and only if $j = k-\ell$ or $j = -(k-\ell)$ (see (2.16)), i.e., $\ell = k-j$ or $\ell = k+j$. Similarly $\partial_{\hat{x}_j}\hat{\sigma}_\ell = 1$ if and only if $j = \ell$ or $j = -\ell$, which implies that $k - \ell = k - j$ or $k - \ell = k + j$. Observe that for $j = 0$ the elements of these two pairs coincide, i.e., $\ell = k-j = k+j$ and $k-\ell = k-j = k+j$. All together we get

$$\partial_{\hat{x}_j}(\hat{\sigma} * \hat{\sigma})_k = \begin{cases} \hat{\sigma}_{k-j} + \hat{\sigma}_{k+j} + \hat{\sigma}_{k-j} + \hat{\sigma}_{k+j} = 2\left(\sigma_{k+j} + \sigma_{k-j}\right), & 1 \leq j \leq P, \\ \hat{\sigma}_k + \hat{\sigma}_k = 2\sigma_k, & j = 0. \end{cases}$$

For all $j = 0, \ldots, P$, we define the vector $S_j \in \mathbb{C}^{2M+1}$ as

$$\begin{aligned} S_j^T &:= \left[\partial_{\hat{x}_j}(\hat{\sigma} * \hat{\sigma})_k\right]_{k=-M}^{M} \\ &= 2\begin{cases} [\hat{\sigma}_{-M+j} + \hat{\sigma}_{-M-j}, \ldots, \hat{\sigma}_{M+j} + \hat{\sigma}_{M-j}], & 1 \leq j \leq P, \\ [\hat{\sigma}_{-M}, \hat{\sigma}_{-M+1}, \ldots, \hat{\sigma}_M], & j = 0, \end{cases} \end{aligned} \tag{2.19}$$

and this gives

$$\partial_{\hat{x}_j} E(\hat{\sigma} * \hat{\sigma}) = S_j^T \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}),$$

i.e., the vector $\nabla_{\hat{x}} E(\hat{\sigma} * \hat{\sigma}) \in \mathbb{C}^{P+1}$ can be computed as

$$\nabla_{\hat{x}} E(\hat{\sigma} * \hat{\sigma}) = \begin{bmatrix} S_0^T \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \\ \vdots \\ S_P^T \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \end{bmatrix} = S^T \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \tag{2.20}$$

where

$$S := \left[S_0, S_1, \ldots, S_P\right] \in \mathbb{C}^{(2M+1)\times(P+1)}.$$

To compute the gradient with respect to $\hat{y}$ we proceed similarly. Replacing $\hat{x}$ with $\hat{y}$ in (2.18), we have (see again (2.16)) that $\partial_{\hat{y}_j}\hat{\sigma}_{k-\ell} = i$ for $j = k-\ell$, i.e. $\ell = k-j$, and $\partial_{\hat{y}_j}\hat{\sigma}_{k-\ell} = -i$ for $j = -(k-\ell)$, i.e., $\ell = j+k$. Thus the term $\partial_{\hat{y}_j}\hat{\sigma}_{k-\ell}\hat{\sigma}_\ell$ gives two terms $i\hat{\sigma}_{k-j}$ and $-i\hat{\sigma}_{k+j}$. For the second term we have instead that $\partial_{\hat{y}_j}\hat{\sigma}_\ell = i$ for $j = \ell$, and $\partial_{\hat{y}_j}\hat{\sigma}_\ell = -i$ for $j = -\ell$, and thus the term $\hat{\sigma}_{k-\ell}\partial_{\hat{y}_j}\hat{\sigma}_\ell$ gives the two terms $i\hat{\sigma}_{k-j}$ and $-i\hat{\sigma}_{k+j}$. Thus for $1 \le j \le P$ and $-M \le k \le M$ it holds

$$\partial_{\hat{y}_j}(\hat{\sigma} * \hat{\sigma})_k = i\hat{\sigma}_{k-j} - i\hat{\sigma}_{k+j} + i\hat{\sigma}_{k-j} - i\hat{\sigma}_{k+j} = 2i\left(-\hat{\sigma}_{k+j} + \hat{\sigma}_{k-j}\right), \quad 1 \le j \le P,$$

In this case we define $j = 1, \dots, P$ the vector $R_j \in \mathbb{C}^{2M+1}$ as

$$R_j^T := \left[\partial_{\hat{y}_j}(\hat{\sigma} * \hat{\sigma})_k\right]_{k=-M}^{M} = 2i[-\hat{\sigma}_{-M+j} + \hat{\sigma}_{-M-j}, \dots, -\hat{\sigma}_{M+j} + \hat{\sigma}_{M-j}], \tag{2.21}$$

and the matrix

$$R := \left[R_1, \dots, R_P\right] \in \mathbb{C}^{(2M+1)\times P},$$

and this gives

$$\nabla_{\hat{y}} E(\hat{\sigma} * \hat{\sigma}) = R^T \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}). \tag{2.22}$$

Recalling that $\hat{z} = [\hat{x}, \hat{y}] \in \mathbb{R}^{2P+1}$, we can combine (2.20) and (2.22) and obtain

$$\nabla_{\hat{z}} E(\hat{\sigma} * \hat{\sigma}) = [S, R]^T \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}), \tag{2.23}$$

where $[S, R] \in \mathbb{C}^{(2M+1)\times(2P+1)}$.

Hessian:

We go on to compute the second order derivatives.

x-derivatives:

For $0 \le j, h \le P$ we use (2.20) to get

$$\begin{aligned}
\partial_{\hat{x}_h}\partial_{\hat{x}_j} E(\hat{\sigma} * \hat{\sigma}) &= \partial_{\hat{x}_h}\left(S_j^T \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma})\right) \\
&= \left(\partial_{\hat{x}_h} S_j^T\right) \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) + S_j^T \cdot \partial_{\hat{x}_h}\left(\nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma})\right) \\
&= \left(\partial_{\hat{x}_h} S_j^T\right) \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) + S_j^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \cdot \partial_{\hat{x}_h}(\hat{\sigma} * \hat{\sigma}) \\
&= \left(\partial_{\hat{x}_h} S_j^T\right) \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) + S_j^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \cdot S_h
\end{aligned}$$

where $H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \in \mathbb{C}^{(2M+1)\times(2M+1)}$ is the Hessian matrix of E computed in $\hat{\mu} = \hat{\sigma} * \hat{\sigma}$. It remains to compute the first term. For this we have from (2.19) that

$$\partial_{\hat{x}_h} S_j^T = 2\partial_{\hat{x}_h} \begin{cases} [\hat{\sigma}_{-M+j} + \hat{\sigma}_{-M-j}, \dots, \hat{\sigma}_{M+j} + \hat{\sigma}_{M-j}], & 1 \le j \le P, \\ [\hat{\sigma}_{-M}, \dots, \hat{\sigma}_M], & j = 0. \end{cases}$$

The case $j = 0$ can be compute directly. Instead, for $j > 0$ and $-M \le k \le M$ we have that $\partial_{\hat{x}_h}(\hat{\sigma}_{k+j} + \hat{\sigma}_{k-j}) = 1$ exactly when

$$\begin{aligned}
&h = k + j \Rightarrow k = h - j \\
&h = -(k+j) \Rightarrow k = -h - j \\
&h = k - j \Rightarrow k = h + j \\
&h = -(k-j) \Rightarrow k = -h + j.
\end{aligned}$$

Observe that, since $0 \le j, h \le P$, we always have $-M \le k \le M$ in the combinations above. Moreover, also in this case the duplicated indices need to be counted only once, and this happens when $h = 0$, so that $h + j = -h + j$ and $-h - j = h - j$.

Denoting as $e_j \in \mathbb{R}^{2M+1}$ the vector with entry 1 in position j, we thus have for $0 \le h, j \le P$ that

$$\partial_{\hat{x}_h} S_j^T = 2 \begin{cases} e_{h-j}^T + e_{j-h}^T + e_{h+j}^T + e_{-h-j}^T, & 1 \le j, h \le P, \\ e_{-j}^T + e_j^T, & 1 \le j \le P, h = 0 \\ e_h^T + e_{-h}^T, & j = 0. \end{cases}$$

Setting $\xi := \nabla_{\hat{x}} E(\hat{\sigma} * \hat{\sigma}) \in \mathbb{R}^{2P+1}$ we can thus compute

$$\left(\partial_{\hat{x}_h} S_j^T\right) \xi = 2 \begin{cases} \xi_{h-j} + \xi_{j-h} + \xi_{h+j} + \xi_{-h-j}, & 1 \le j, h \le P, \\ \xi_j + \xi_{-j}, & 1 \le j \le P, h = 0 \\ \xi_h + \xi_{-h}, & j = 0. \end{cases}$$

We can group these indices into two matrices $M_{xx,1}(\xi)$ (index $h-j$ and $j-h$), $M_{xx,2}(\xi)$ (index $h+j$ and index $-h-j$) and obtain

$$M_{xx,1}(\xi) = \begin{bmatrix} 2\xi_0 & \xi_1 + \xi_{-1} & \xi_2 + \xi_{-2} & \dots & \xi_P + \xi_{-P} \\ \xi_1 + \xi_{-1} & 2\xi_0 & \xi_1 + \xi_{-1} & \dots & \xi_{P-1} + \xi_{1-P} \\ \xi_2 + \xi_{-2} & \xi_1 + \xi_{-1} & 2\xi_0 & \dots & \xi_{P-2} + \xi_{2-P} \\ \vdots & \vdots & \vdots & & \vdots \\ \xi_P + \xi_{-P} & \xi_{P-1} + \xi_{1-P} & \xi_{P-2} + \xi_{2-P} & \dots & 2\xi_0 \end{bmatrix},$$

$$M_{xx,2}(\xi) = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & \xi_2 + \xi_{-2} & \xi_3 + \xi_{-3} & \dots & \xi_{1+P} + \xi_{-P-1} \\ 0 & \xi_3 + \xi_{-3} & \xi_4 + \xi_{-4} & \dots & \xi_{2+P} + \xi_{-P-2} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & \xi_{P+1} + \xi_{-P-1} & \xi_{P+2} + \xi_{-P-2} & \dots & \xi_{2P} + \xi_{-2P} \end{bmatrix},$$

so that $M_{xx}(\xi) := \left[\left(\partial_{\hat{x}_h} S_j^T\right)(\xi)\right]_{h,j=0}^{P} = 2\left(M_{xx,1}(\xi) + M_{xx,2}(\xi)\right)$. Observe that $M_{xx,1}(x)$ is a Toeplitz matrix and $M_{xx,2}(x)$ is an Hankel matrix without the first row and column.

We can conclude that

$$\partial_{\hat{x}_h} \partial_{\hat{x}_j} E(\hat{\sigma} * \hat{\sigma}) = M_{xx} \left(\nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma})\right) + S^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) S$$

Mixed derivatives:

For $1 \le h \le P$ and $0 \le j \le P$ we use again (2.20) to get

$$\begin{aligned} \partial_{\hat{y}_h} \partial_{\hat{x}_j} E(\hat{\sigma} * \hat{\sigma}) &= \partial_{\hat{y}_h} \left(S_j^T \cdot \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma})\right) \\ &= \left(\partial_{\hat{y}_h} S_j^T\right) \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) + S_j^T \cdot \partial_{\hat{y}_h} \left(\nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma})\right) \\ &= \left(\partial_{\hat{y}_h} S_j^T\right) \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) + S_j^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \cdot \partial_{\hat{y}_h} (\hat{\sigma} * \hat{\sigma}) \\ &= \left(\partial_{\hat{y}_h} S_j^T\right) \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) + S_j^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \cdot R_h \end{aligned}$$

For the first term in the sum we have from (2.19) that

$$\partial_{\hat{y}_h} S_j^T = 2 \partial_{\hat{y}_h} \begin{cases} [\hat{\sigma}_{-M+j} + \hat{\sigma}_{-M-j}, \dots, \hat{\sigma}_{M+j} + \hat{\sigma}_{M-j}], & 1 \le j \le P, \\ [\hat{\sigma}_{-M}, \dots, \hat{\sigma}_M], & j = 0. \end{cases}$$

Also in this case if $j = 0$ the computation is easier and we get that

$$\partial_{\hat{y}_h} \hat{\sigma}_k = 2 \begin{cases} -i, & h = -k \\ i, & h = k \end{cases}, \quad -M \le k \le M, 1 \le h \le P.$$

If instead $j > 0$ and $-M \le k \le M$ we have that

$$\partial_{\hat{y}_h} \left(\hat{\sigma}_{k+j} + \hat{\sigma}_{k-j}\right) = \begin{cases} i, & h = k + j \Rightarrow k = h - j \\ -i, & h = -(k+j) \Rightarrow k = -h - j \\ i, & h = k - j \Rightarrow k = h + j \\ -i, & h = -(k-j) \Rightarrow k = -h + j. \end{cases}$$

In this case there are no duplicated indices since $j, h > 0$.

Reasoning as before, for $1 \le h \le P$ and $0 \le j \le P$ we thus have

$$\partial_{\hat{y}_h} S_j^T = 2i \begin{cases} e_{h-j}^T - e_{j-h}^T + e_{h+j}^T - e_{-h-j}^T, & 1 \le j, h \le P, \\ e_h^T - e_{-h}^T, & j = 0, \end{cases}$$

and defining the matrices $M_{yx,1}(\xi), M_{yx,2}(\xi) \in \mathbb{C}^{P \times (P+1)}$ by

$$M_{yx,1}(\xi) = \begin{bmatrix} \xi_1 - \xi_{-1} & 0 & \xi_{-1} - \xi_1 & \cdots & \xi_{1-P} - \xi_{P-1} \\ \xi_2 - \xi_{-2} & \xi_1 - \xi_{-1} & 0 & \cdots & \xi_{2-P} - \xi_{P-2} \\ \xi_3 - \xi_{-3} & \xi_2 - \xi_{-2} & \xi_1 - \xi_{-1} & \cdots & \xi_{3-P} - \xi_{P-3} \\ \vdots & \vdots & \vdots & & \xi_{-1} - \xi_1 \\ \xi_P - \xi_{-P} & \xi_{P-1} - \xi_{1-P} & \xi_{P-2} - \xi_{2-P} & \cdots & \xi_1 - \xi_{-1} \quad 0 \end{bmatrix},$$

$$M_{yx,2}(\xi) = \begin{bmatrix} 0 & \xi_2 - \xi_{-2} & \xi_3 - \xi_{-3} & \cdots & \xi_{1+P} - \xi_{-P-1} \\ 0 & \xi_3 - \xi_{-3} & \xi_4 - \xi_{-4} & \cdots & \xi_{2+p} - \xi_{-p-2} \\ 0 & \xi_4 - \xi_{-4} & \xi_5 - \xi_{-5} & \cdots & \xi_{3+p} - \xi_{-p-3} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & \xi_{P+1} - \xi_{-P-1} & \xi_{P+2} - \xi_{-P-2} & \cdots & \xi_{2P} - \xi_{-2P} \end{bmatrix},$$

so that $M_{yx}(\xi) := \left[\left(\partial_{\hat{y}_h} S_j^T \right) \xi \right]_{1 \le h \le P, 0 \le j \le P} = 2i \left(M_{yx,1}(\xi) + M_{yx,2}(\xi) \right)$.

We have thus

$$\partial_{\hat{y}_h} \partial_{\hat{x}_j} E(\hat{\sigma} * \hat{\sigma}) = M_{yx} \left(\nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \right) + R^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \cdot S$$

y-derivatives:

For $1 \le j, h \le P$ we use instead (2.22) to get as before

$$\partial_{\hat{y}_h} \partial_{\hat{y}_j} E(\hat{\sigma} * \hat{\sigma}) = \left(\partial_{\hat{y}_h} R_j^T \right) \nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) + R_j^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \cdot R_h,$$

For the derivative of R_j we use (2.21) to get for $1 \le j \le P$ that

$$\partial_{\hat{y}_h} R_j^T = 2i \partial_{\hat{y}_h} [-\hat{\sigma}_{-M+j} + \hat{\sigma}_{-M-j}, \ldots, -\hat{\sigma}_{M+j} + \hat{\sigma}_{M-j}],$$

where for $1 \le j, h \le P$ and $-M \le k \le M$ it holds

$$\partial_{\hat{y}_h} \left(-\hat{\sigma}_{k+j} + \hat{\sigma}_{k-j} \right) = \begin{cases} -i, & h = k + j \Rightarrow k = h - j \\ i, & h = -(k+j) \Rightarrow k = -h - j \\ i, & h = k - j \Rightarrow k = h + j \\ -i, & h = -(k-j) \Rightarrow k = -h + j. \end{cases}$$

This gives for $1 \le j, h \le P$

$$\begin{aligned} \partial_{\hat{y}_h} R_j^T &= 2i \left(-ie_{h-j}^T - ie_{j-h}^T + ie_{h+j}^T + ie_{-h-j}^T \right) \\ &= 2 \left(e_{h-j}^T + e_{j-h}^T - e_{h+j}^T - e_{-h-j}^T \right). \end{aligned}$$

In this case we define the $P \times P$ matrices $M_{yy,1}(\xi)$ (indices $h - j$ and $j - h$) and $M_{yy,2}(\xi)$ (indices $h + j$ and $-h - j$) as

$$M_{yy,1}(\xi) = \begin{bmatrix} 2\xi_0 & \xi_1 + \xi_{-1} & \xi_2 + \xi_{-2} & \cdots & \xi_{P-1} + \xi_{1-P} \\ \xi_1 + \xi_{-1} & 2\xi_0 & \xi_1 + \xi_{-1} & \cdots & \xi_{P-2} + \xi_{2-P} \\ \xi_2 + \xi_{-2} & \xi_1 + \xi_{-1} & 2\xi_0 & \cdots & \xi_{P-3} + \xi_{3-P} \\ \vdots & \vdots & \vdots & & \vdots \\ \xi_{P-1} + \xi_{1-P} & \xi_{P-2} + \xi_{2-P} & \xi_{P-3} + \xi_{3-P} & \cdots & 2\xi_0 \end{bmatrix},$$

$$M_{yy,2}(\xi) = \begin{bmatrix} \xi_2 + \xi_{-2} & \xi_3 + \xi_{-3} & \xi_4 + \xi_{-4} & \cdots & \xi_{1+P} + \xi_{-P-1} \\ \xi_3 + \xi_{-3} & \xi_4 + \xi_{-4} & \xi_5 + \xi_{-5} & \cdots & \xi_{2+P} + \xi_{-P-2} \\ \xi_4 + \xi_{-4} & \xi_5 + \xi_{-5} & \xi_6 + \xi_{-6} & \cdots & \xi_{3+P} + \xi_{-P-3} \\ \vdots & \vdots & \vdots & & \vdots \\ \xi_{P+1} + \xi_{-P-1} & \xi_{P+2} + \xi_{-P-2} & \xi_{P+3} + \xi_{-P-3} & \cdots & \xi_{2P} + \xi_{-2P} \end{bmatrix},$$

so that $M_{yy}(\xi) := \left[\left(\partial_{\hat{y}_h} R_j^T\right)\xi\right]_{h,j=1}^{P} = 2\left(M_{yy,1}(\xi) - M_{xx,2}(\xi)\right)$. We have

$$\partial_{\hat{y}_h}\partial_{\hat{y}_j} E(\hat{\sigma} * \hat{\sigma}) = M_{yy}\left(\nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma})\right) + R^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \cdot R$$

Observe that $M_{yy,1}(\xi)$ can be obtained from $M_{xx,1}(\xi)$ by removing the first row and columns, and the same holds for $M_{yy,2}$ and $M_{xx,2}$.

Full Hessian:

Combining the three terms, we get the full Hessian matrix

$$\begin{aligned} H_{\hat{z}} E(\hat{\sigma} * \hat{\sigma}) &= \left[\partial_{\hat{z}_h}\partial_{\hat{z}_j} E(\hat{\sigma} * \hat{\sigma})\right]_{0 \le h,j \le 2P+1} \\ &= M\left(\nabla_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma})\right) + [S, R]^T \cdot H_{\hat{\mu}} E(\hat{\sigma} * \hat{\sigma}) \cdot [S, R]. \end{aligned}$$

where

$$M(\xi) := \begin{bmatrix} M_{xx}(\xi) & M_{yx}(\xi)^T \\ M_{yx}(\xi) & M_{yy}(\xi) \end{bmatrix}.$$

2.3 Discussion

We implemented our method in Matlab. Although the code is still not complete, it was already possible to extensively verify that the optimizer is able to converge with the theoretically predicted rate to a local minimum. Moreover, the code includes an adaptive update of the time stepping in the Implicit Euler solver.

We consider our experiments with a very simple example, where we define $f(x) = -sin2\pi x$ and $u(x) = x - \frac{1}{2}$, so according to (1.14) the expected μ should be

$$\mu(x) = \frac{1}{2\pi}(1 - cos2\pi x)$$

The results so far

To solve the problem we discretize it with the method defined in Section 2.1.1. In particular, we consider here values $N = 3, N = 20$, correspondingly $M = 4, M = 10$, $\delta = 10^{-5}$, Newton tolerance 10^{-3}, and time step 0.1 which increases by factor of 1.05 whenever Newton converges.The method computes a sequence of approximation for μ, and we are going to analyze the quality and properties of this sequence of approximants. In particular, in the case of $N = 3, M = 4$, we show in Figure 2.1 (left upper corner) the approximation of f, μ, u. In the right-top corner, instead, we visualize the decay of the steps taken by the Euler method (i.e, the distance between consecutive points on the numerical trajectory), and the norm of the gradient of the optimization functional along the same trajectory and both decay as expected. Also as one can see the eigenvalues of the Hessian are quite large and positive, indicating that the optimizer is converging to a unique stationary points, but the solution is not the desired one as you can see in left upper corner of the figure. This indicates that there should be some issues in our derivations of the discrete problem, which is however correctly solved. In the same figure, the lower part we are presenting the experiment with $N = 20, M = 10$ and with the same optimization parameters. And the behaviours are almost the same, however we still do not get the desired solution.

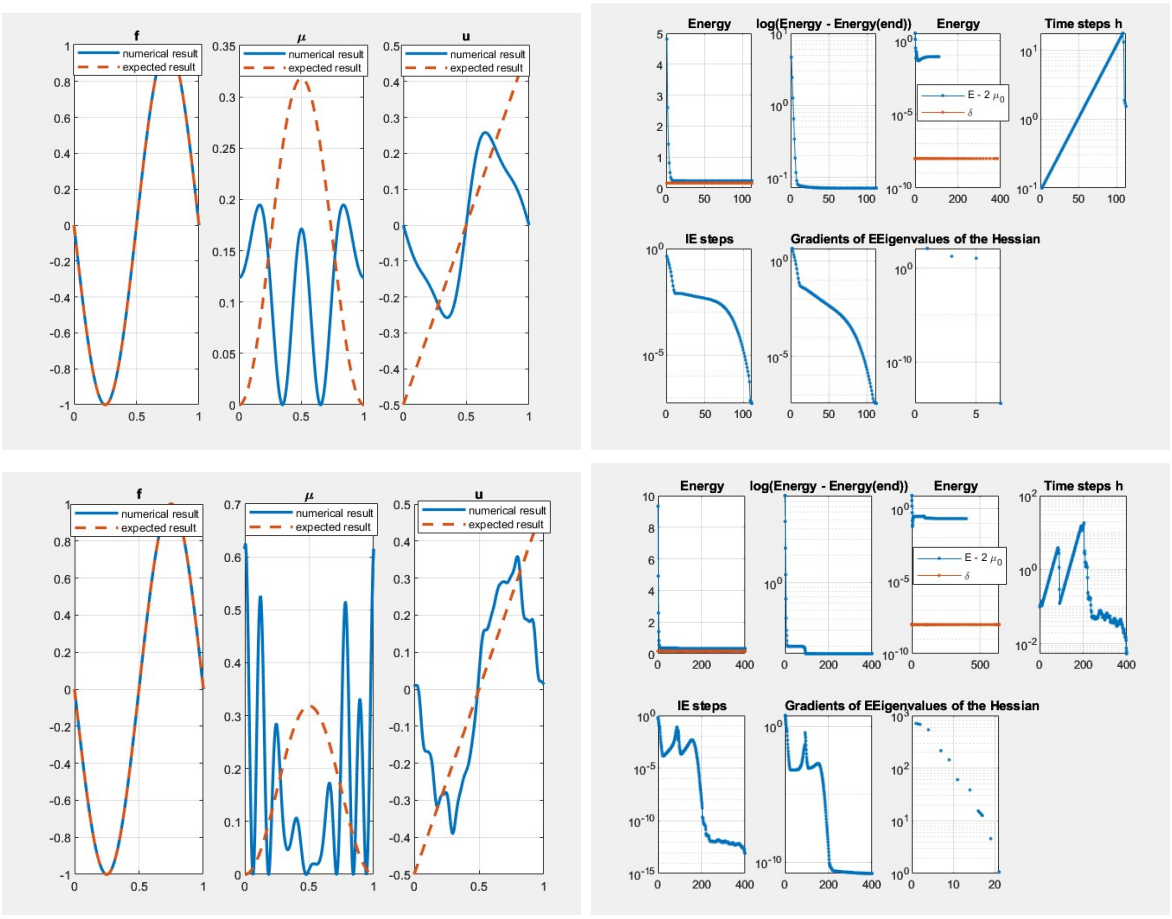

Figure 2.1: The upper part of the figure illustrates the case when $N = 3, M = 4$. On the right-hand side, you can observe the resulting μ and u, compared with their true values. Meanwhile, on the left-hand side, we have plotted key quantities essential for analyzing our solver's performance. In the lower part of the figure, we repeat the same analysis for the case of $N = 20, M = 10$

Now, in Figure 2.2, we run the experiment again with $N = 20, M = 10$ but this time the initial data is a perturbed version of the true solution. The optimizer is behaving well, but yet we do not get the solution.

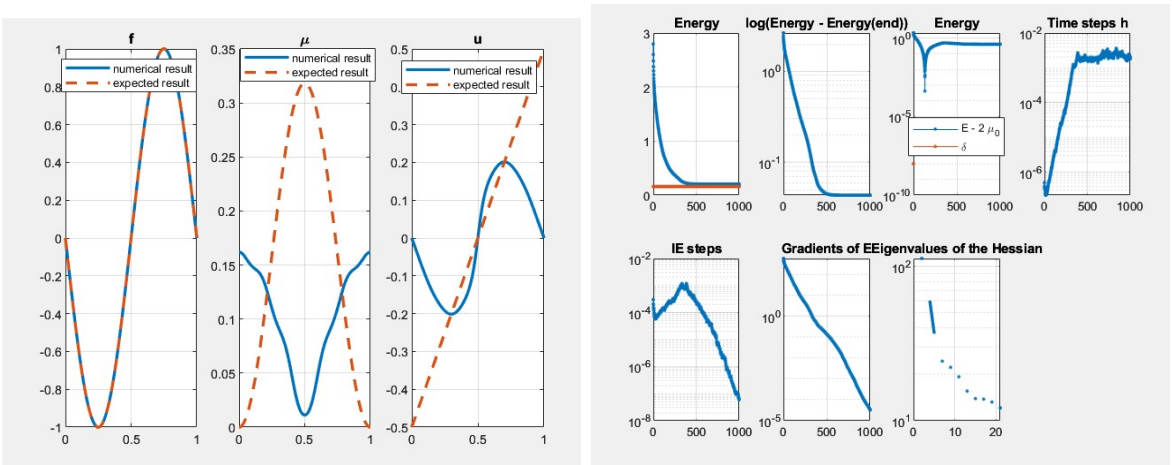

Figure 2.2: The figure presents the outcomes obtained when running the code with initial data that closely resembles the actual solution

The current work is devoted to identify and fix possible problems that lead to this discrepancy. However, observe that the solution is strictly positive in the domain, meaning that the constraints imposed on the solution are working as needed.

2.3.1 An observation

After a careful revision, it turned out that the matrix $T(\hat{\mu})$ in (2.5) is not exactly a Toeplitz matrix, we can explain the form of the new matrix with the following example where $N = 3, M = 4$.

The Toeplitz matrix we were working on is a $2N \times 2N$ matrix generated as follow

$$Toep([\mu_0, \mu_{-1}, \mu_{-2}, \mu_{-3}, \mu_{-4}, 0], [\mu_0, \mu_1, \mu_2, \mu_3, \mu_4, 0]) \text{ (with } (2N - M - 1) \text{ zeros added)}$$

$$= \begin{bmatrix} \mu_0 & \mu_1 & \mu_2 & \mu_3 & \mu_4 & 0 \\ \mu_{-1} & \mu_0 & \mu_1 & \mu_2 & \mu_3 & \mu_4 \\ \mu_{-2} & \mu_{-1} & \mu_0 & \mu_1 & \mu_2 & \mu_3 \\ \mu_{-3} & \mu_{-2} & \mu_{-1} & \mu_0 & \mu_1 & \mu_2 \\ \mu_{-4} & \mu_{-3} & \mu_{-2} & \mu_{-1} & \mu_0 & \mu_1 \\ 0 & \mu_{-4} & \mu_{-3} & \mu_{-2} & \mu_{-1} & \mu_0 \end{bmatrix}$$

To construct the new matrix, we first construct a Toeplitz matrix of dimension $(2N+1) \times (2N+1)$ as follow

$$Toep([\mu_0, \mu_{-1}, \mu_{-2}, \mu_{-3}, \mu_{-4}, 0, 0], [\mu_0, \mu_1, \mu_2, \mu_3, \mu_4, 0, 0]) \text{ (with } (2N - M) \text{ zeros added)}$$

$$= \begin{bmatrix} \mu_0 & \mu_1 & \mu_2 & \mu_3 & \mu_4 & 0 & 0 \\ \mu_{-1} & \mu_0 & \mu_1 & \mu_2 & \mu_3 & \mu_4 & 0 \\ \mu_{-2} & \mu_{-1} & \mu_0 & \mu_1 & \mu_2 & \mu_3 & \mu_4 \\ \mu_{-3} & \mu_{-2} & \mu_{-1} & \mu_0 & \mu_1 & \mu_2 & \mu_3 \\ \mu_{-4} & \mu_{-3} & \mu_{-2} & \mu_{-1} & \mu_0 & \mu_1 & \mu_2 \\ 0 & \mu_{-4} & \mu_{-3} & \mu_{-2} & \mu_{-1} & \mu_0 & \mu_1 \\ 0 & 0 & \mu_{-4} & \mu_{-3} & \mu_{-2} & \mu_{-1} & \mu_0 \end{bmatrix}$$

Then we remove the column and the row that correspond to u_0, *i.e.* column $N+1$ and row $N+1$, then we get

$$T = \begin{bmatrix} \mu_0 & \mu_1 & \mu_2 & \mu_4 & 0 & 0 \\ \mu_{-1} & \mu_0 & \mu_1 & \mu_3 & \mu_4 & 0 \\ \mu_{-2} & \mu_{-1} & \mu_0 & \mu_2 & \mu_3 & \mu_4 \\ \mu_{-4} & \mu_{-3} & \mu_{-2} & \mu_0 & \mu_1 & \mu_2 \\ 0 & \mu_{-4} & \mu_{-3} & \mu_{-1} & \mu_0 & \mu_1 \\ 0 & 0 & \mu_{-4} & \mu_{-2} & \mu_{-1} & \mu_0 \end{bmatrix}$$

Now, when we substitute this new T and do the appropriate changes in the Gradient and the Hessian, we get the following results

As depicted in Figure 2.3, the upper part illustrates the results for the case where $N = 3$ and $M = 4$, and it is evident that we are still quite distant from reaching the true solution. On the other hand, in the scenario of $N = 20$ and $M = 10$, the norm of the gradient initially exhibits a decreasing trend, but then experiences a sudden jump. This means we need more thorough investigations.

Lastly, when we execute this modified code with initial data that is close to the actual solution, the optimizer's performance was poor (Figure 2.4).

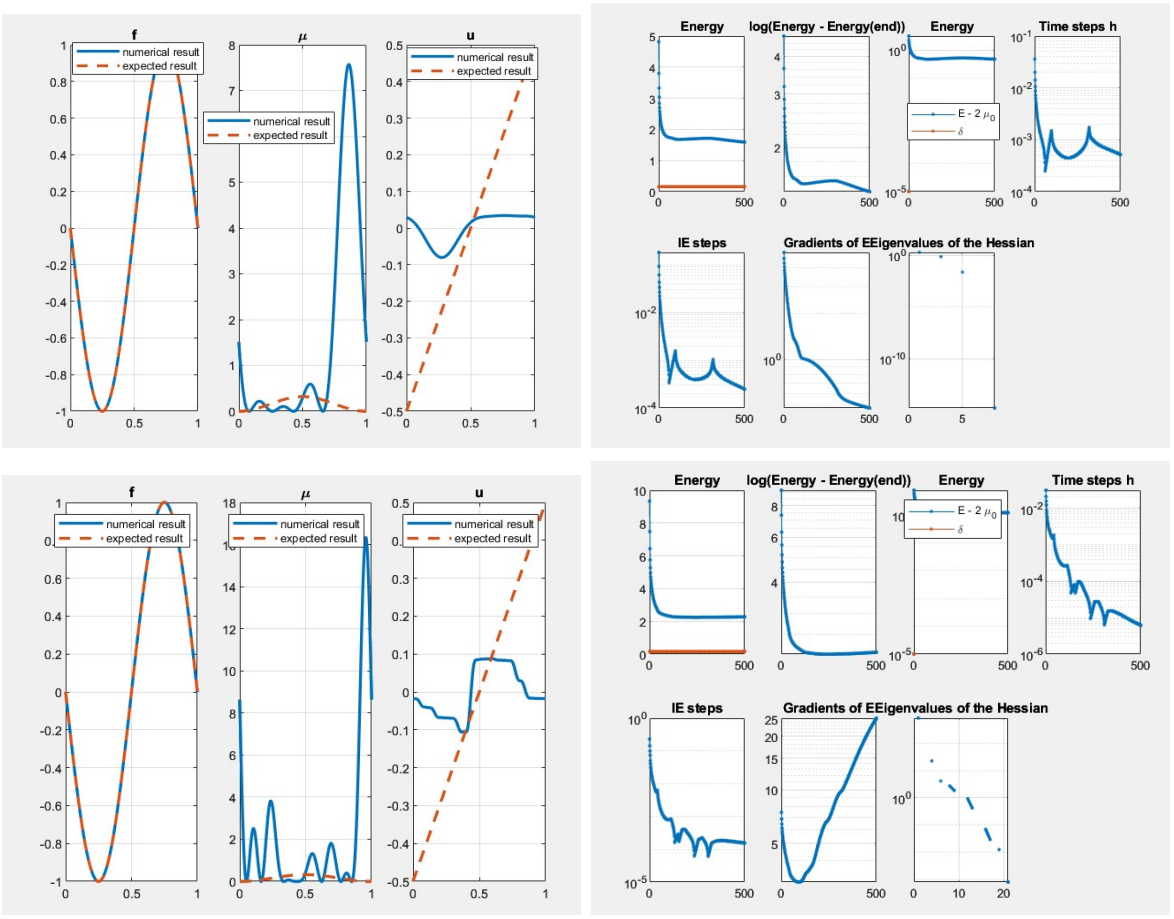

Figure 2.3: In this figure, we present results obtained when the matrix T is modified. The upper part of the figure illustrates the case when $N = 3, M = 4$. On the right-hand side, you can observe the resulting μ and u, compared with their true values. Meanwhile, on the left-hand side, we have plotted key quantities essential for analyzing our solver's performance. In the lower part of the figure, we repeat the same analysis for the case of $N = 20, M = 10$

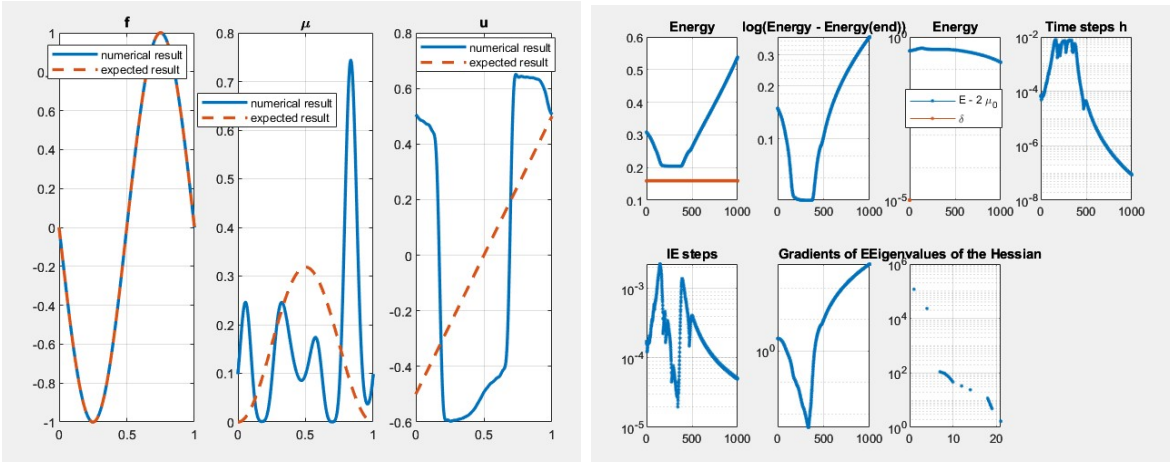

Figure 2.4: The results for the modified code with initial data that closely resembles the actual solution

Chapter 3

UQ and MC

Uncertainty Quantification (UQ) [39] is the combination of probability theory and statistical methods with real-world applications. A UQ problem usually involves one or more mathematical models for the study of interest, with uncertainties regarding the correct form or parameter values of those models. These uncertainties are often addressed probabilistically. Often, the probabilistic theory behind UQ methods is relatively straightforward but becomes complicated when combined with the complexities of the application. For example, the prediction of a quantity of interest mighty be measured by an elementary mathematical object, such as an *expected value*; but the complexity is how to compute the expected value with an integral over a million-dimensional parameter space.

Typical UQ problems can be: prediction, model and software verification and validation, parameter estimation, and inverse problems.

3.1 UQ in Parametric PDEs

The general setting of our work is having a parametric PDEs with uncertainties in the parameters. The following example (taken from [39] will demonstrate the main idea.

Example: Consider an elliptic boundary value problem on a connected Lipschitz domain $\mathcal{X} \subset \mathbb{R}^n$ $(n = 2 or 3)$:

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f && \text{in } \mathcal{X} \\ u &= b && \text{on } \partial\mathcal{X} \end{aligned}$$

The tensor field $\kappa : \mathcal{X} \to \mathbb{R}^{n\times n}$ describes the permeability of the material to the fluid. The source term $f : \mathcal{X} \to \mathbb{R}$, and $b : \partial\mathcal{X} \to \mathbb{R}$ is the pressure specified on the boundary of $\mathcal{X}$.

From the theory of PDEs, we know that, for each given positive definite and bounded κ, the problem above has a unique weak solution $u \in H_0^1(\mathcal{X})$ for each $f \in H^{-1}(\mathcal{X})$. In real life application in general, the 'precise' κ, f, and b might be uncertain, and hence our knowledge of the solution $u = u(\kappa, f, b)$ is uncertain as well. If κ, f, and b are treated as random variables following probability distributions, then also u becomes a random variable. Thus, one is might be interested in properties of random variables, such as mean, variance, deviation probabilities, etc. This is known as the *forward propagation of uncertainty.*

Now assume we have a solution map u from a parameter space $\mathcal{P} \subset \mathbb{R}^{n_p}$, $n_p \in \mathbb{N}$, to a Hilbert space $\mathcal{H}$, i.e.,

$$\begin{aligned} u :& \mathcal{P} \to \mathcal{H} \\ & p \mapsto u(p). \end{aligned}$$

We are interested in quantifying the effect of the input parameter p on the solution $u(p)$. We may assume that the parameter p is distributed according to a prior probability distribution ρ_p on $\mathcal{P}$, and we would like to compute the expected (or mean) solution

$$\mathbb{E}(u, \rho_p) := \int_{\mathcal{P}} u(p) d\rho_p(p). \tag{3.1}$$

In some cases, one may also be interested in higher order moments

$$\mathbb{E}(u^s, \rho_p) := \int_{\mathcal{P}} u^s(p) d\rho_p(p), \quad s \in \mathbb{N},$$

Computing the integral (3.1) is impossible in general, since we do not know the map u explicitly and since $\mathcal{P}$ may be high dimensional (i.e., $n_p \gg 1$). It is thus of interest to try to approximate (3.1) with some numerical discretization, and the most commonly used one is the **Monte Carlo method**. In the following section we introduce Monte Carlo Simulation.

3.2 Monte Carlo Simulation

Monte Carlo (MC) [40] is a computational method, it uses generated sample from a given distribution to estimate some features of this distribution (for example, the moments of the distribution).

Let X be a random variable of a given distribution $P_X(x)$, and assume we can withdraw a sample from this distribution

$$\xi_n = \{x_1, x_2, \ldots, x_n\}$$

Denote by

$$F(P_X)$$

the feature that we are interested to compute (*e.g.* the mean, the variance). And denote by $P_n(x)$ the **empirical distribution** of the sample ξ_n (*i.e.* it is the probability distribution that assigns $\frac{1}{n}$ to each one of the values x_i). Then

$$F(P_X) \simeq F(P_n)$$

is the *Monte Carlo* approximation of the specific feature.

Thus the idea of MC is to approximate the feature $F(P_X)$ by the corresponding feature of the empirical distribution of the generated sample.

Now back to our problem, we collect $N \in \mathbb{N}$ samples $p_1, \ldots, p_N$ sampled from $\mathcal{P}$ according to ρ_p, for each sample we compute the solution $u(p_i)$, $1 \leq i \leq N$, and we use the approximation

$$\mathbb{E}_{MC}(u) := \frac{1}{N} \sum_{i=1}^{N} u(p_i) \tag{3.2}$$

We denote by $\rho_{p,N}$, the empirical distribution

$$\rho_{p,N} := \frac{1}{N} \sum_{i=1}^{N} \delta_{p_i},$$

where δ_{p_i} is a measure concentrated in p_i. Indeed, we have

$$\mathbb{E}_{MC}(u) = \frac{1}{N} \sum_{i=1}^{N} u(p_i) = \int_{\mathcal{P}} u(p) d\rho_{p,N}(p) = \mathbb{E}(u, \rho_{p,N}). \tag{3.3}$$

Under some reasonable assumption on u (i.e. on the PDE) it is known that there is $c > 0$ such that

$$|\mathbb{E}(u, \rho_p) - \mathbb{E}(u, \rho_{p,N})| \leq cN^{-1/2}. \tag{3.4}$$

This means that the method is convergent, that the converge rate is independent from the dimension n_p of $\mathcal{P}$, but also that the rate is quite slow in terms of N. In other terms, for a good approximation we need a large N, which means that we need to solve the PDE for a substantial number of times. This is potentially very expensive.

To overcome this problem, the plan is to use **data generation** algorithms.

For an efficient generation of MC ensemble, the idea is as follows: we fix a (possibly low dimensional) space $\mathcal{Q} \subset \mathbb{R}^{n_q}$ with a probability distribution ρ_q, and we aim at defining a map

$$\hat{u} : \mathcal{Q} \to \mathcal{H}$$

with the property that the corresponding mean

$$\mathbb{E}(\hat{u}, \rho_q) = \int_{\mathcal{Q}} \hat{u}(q) d\rho_q(q) \tag{3.5}$$

is a good approximation of $\mathbb{E}(u, \rho_p)$ in (3.1).

In practice, this will be realized by requiring that the Monte Carlo approximation of $\mathbb{E}(\hat{u}, \rho_q)$, i.e. the empirical mean, be a good approximation of the Monte Carlo approximation of $\mathbb{E}(u, \rho_p)$, since both of these are quantities that we are able to compute, in contrast with the exact integrals (3.1) and (3.5). To this end, we aim at proving that there are a constant $c > 0$ and a rate $t > 0$ such that

$$|\mathbb{E}(u, \rho_{p,N}) - \mathbb{E}(\hat{u}, \rho_{q,N})| \leq c_2 N^{-t}, \tag{3.6}$$

for any $p_1, \ldots, p_N$ drawn from $(\mathcal{P}, \rho_p)$ and any $q_1, \ldots, q_N$ from $(\mathcal{Q}, \rho_q)$.

Moreover, sampling $q_1, \ldots, q_N$ from $\mathcal{Q}$ is simple (we fixed ρ_q) and we need to properly design the map $\hat{u}$ so that its evaluation $\hat{u}(q_i)$, $1 \leq i \leq N$ is fast and computationally cheap. If these conditions are satisfied, in practice we will just compute the Monte Carlo approximation $\mathbb{E}(\hat{u}, \rho_{q,N})$, and we will obtain a good approximation of $\mathbb{E}(u, \rho_p)$. Indeed, using (3.4) and (3.6) we get the bound

$$\begin{aligned}
|\mathbb{E}(u, \rho_p) - \mathbb{E}(\hat{u}, \rho_{q,N})| &\leq |\mathbb{E}(u, \rho_p) - \mathbb{E}(u, \rho_{p,N})| + |\mathbb{E}(u, \rho_{p,N}) - \mathbb{E}(\hat{u}, \rho_{q,N})| \\
&\leq c_1 N^{-1/2} + c_2 N^{-t}, \\
|\mathbb{E}(u, \rho_p) - \mathbb{E}(\hat{u}, \rho_{q,N})| &\leq c_3 N^{-s}.
\end{aligned}$$

One of the key advantage of using deep leaning generative model in these applications is that we do not need to be accurate in each element of the ensemble, but instead we need the average to be approximated correctly. And as we show above, this can be achieved when we enlarge the number of ensemble. And here come the main advantage of data generation, because once the generative model is trained then it is very cheap to generate a substantial number of ensembles.

Chapter 4

Deep Learning and Data Generation Models

In recent years, deep learning has revolutionized numerous fields, from computer vision and natural language processing to healthcare and autonomous systems. By leveraging vast amounts of data and complex neural network architectures, deep learning models have achieved high accuracy and capabilities. The concept of *data generation* is the center of the success of these models. The main idea of data generation is that synthetic data is generated such that it resembles the original datasets.

This chapter explores some deep learning and data generation models, delving into the methods that drive their effectiveness. In particular, in the last section, we focus on the generative models that we use for our ultimate goal, namely, Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). Additionally, the chapter highlights the critical role of hyperparameter tuning in optimizing neural networks. Proper tuning is essential to maximize model performance, ensuring that deep learning systems operate efficiently and accurately in diverse applications.

Before exploring neural networks and deep learning, it is essential to establish the foundational concepts of Machine Learning. We'll outline the key definitions and explain some basic terminologies.

4.1 Machine Learning (ML)

Machine learning is a collection of techniques that enable computer systems to "learn" from training data, allowing them to make predictions on new, unseen data without requiring explicit programming [37].

Research in machine learning is advancing quickly, driven by academic institutions and corporate departments. This rapid growth is due to the transformative impact of ML across a wide range of fields and applications. For instance, in healthcare, ML algorithms are revolutionizing diagnostics and personalized medicine. In finance, they are enhancing fraud detection and algorithmic trading. In everyday technology, ML powers everything from recommendation systems in streaming services to autonomous vehicles. As the demand for intelligent systems continues to rise, so does the innovation and development in machine learning, shaping the future of countless industries.

Machine Learning serves as a *central hub*, connecting and integrating with numerous other disciplines. It intersects with *Optimization* to improve algorithmic efficiency, and with *Artificial Intelligence* to enhance decision-making and automation. In *Computer Vision* and *Robotics*, ML drives advancements in object recognition and autonomous control. It relies on *Information Theory* for effective data processing and *Data Mining* for discovering patterns in large datasets. *Statistics* underpins the theoretical foundations of ML, while *Big Data* provides the vast amounts of information required for model training. *Cognitive Science* informs the development of learning algorithms by modeling human cognition, and *Game Theory* contributes to strategies for decision-making in competitive environments.

Machine learning fundamentally aims to transform data-driven experience into computational knowledge. The learning process involves several key components [37]. The domain set ($\mathcal{X}$) rep-

resents all possible instances that require classification, while the label set ($\mathcal{Y}$) represents all potential classifications. Training occurs on a finite sequence of labeled samples/training set $\mathcal{S} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, drawn from the product space $\mathcal{X} \times \mathcal{Y}$.

The learning algorithm's objective is to produce a prediction function $h : \mathcal{X} \to \mathcal{Y}$ that can accurately classify previously unseen instances. This function approximates an unknown true labeling function f, which represents the ground truth the algorithm attempts to discover. Success is measured by the generalization error, defined as:

$$L_{\mathcal{D},f}(h) = \mathbb{P}_{x\sim D}[h(x) \neq f(x)] = D(\{x : h(x) \neq f(x)\}).$$

This error quantifies the probability that h will misclassify a random instance drawn from the underlying distribution $\mathcal{D}$.

Empirical Risk Minimization (ERM)

Empirical Risk Minimization (ERM) serves as a foundational principle in statistical learning that underpins many machine learning approaches. A learning algorithm takes as input a training set $\mathcal{S}$, sampled from an unknown distribution $\mathcal{D}$ and labeled by a target function f, with the goal of producing a predictor $h_{\mathcal{S}} : \mathcal{X} \to \mathcal{Y}$. The algorithm aims to minimize the error relative to the unknown $\mathcal{D}$ and f. The learner compute the training error, which is the error the classifier makes on the training data itself:

$$L_{\mathcal{S}} = \frac{|\{i \in \{1, 2, \ldots, m\} : h(x_i) \neq y_i\}|}{m}$$

This error is called *empirical error/ empirical risk*. Given that the training sample is the learner's only view of the world, it is logical to seek a solution that performs well on this data. The approach of finding a predictor h that minimizes the error on the training set, $L_{\mathcal{S}}$, is known as Empirical Risk Minimization (ERM)

Error Decomposition

We break down the error of an ERM predictor into two parts as follows [37]: let $h_{\mathcal{S}}$ represent an ERM hypothesis. We can then express it as:

$$L_{\mathcal{D}}(h_{\mathcal{S}}) = \epsilon_{app} + \epsilon_{est}$$

where

$$\epsilon_{app} = min_{h\in\mathcal{H}} L_{\mathcal{D}}(h), \quad \epsilon_{est} = L_{\mathcal{D}}(h_{\mathcal{S}}) - \epsilon_{app}$$

The Approximation Error (ϵ_{app}): it represents the minimum risk achievable by a predictor within a given hypothesis class, which measures the risk incurred due to limiting the model to a specific hypothesis class. Importantly, the approximation error is independent of sample size and is determined solely by the choice of hypothesis class. Expanding the hypothesis class can reduce this error. Under the realizability assumption, the approximation error is zero, but in the agnostic case, it can be significant.

The estimation Error (ϵ_{est}): represents the gap between the approximation error and the actual error of the ERM predictor. This arises because the empirical risk (training error) is only an approximation of the true risk, meaning the predictor that minimizes empirical risk may not minimize the true risk. The quality of this estimation depends on both the size of the training set and the complexity of the hypothesis class. For a finite hypothesis class, the estimation error increases logarithmically with the size of the class $|\mathcal{H}|$ and decreases with the number of training samples m.

When aiming to minimize total risk, we encounter a trade-off known as the *bias-complexity trade-off*. If we choose a very large hypothesis class $\mathcal{H}$, the approximation error decreases, but the estimation error may increase, leading to *overfitting*—where the model fits the training data too closely and fails to generalize. Conversely, choosing a very small hypothesis class reduces the estimation error, but can

increase the approximation error, resulting in *underfitting*—where the model is too simple and fails to capture patterns in the data.

Learning theory explores how complex we can make $\mathcal{H}$ while keeping the estimation error at a manageable level.

Model selection and validation

How would we select the best algorithm or hyperparameters? there are several approches but here we use the simplest way; by sampling an additional set of samples, independent of the training set, and using the empirical error on this *validation set* as our estimator. Formally, let $V = \{(x_1, y_1), \ldots, (x_{m_v}, y_{m_v})\}$ to be a set of m_v samples from $\mathcal{D}$ that is independent from the training set. We have the following theorem

Theorem 6 *Let h be some predictor and assume that the loss function is in $[0, 1]$. Then, for every $\delta \in (0, 1)$, with probability at least $1 - \delta$ over the choice of a validation set V of size m_v we have*

$$|L_V(h) - L_{\mathcal{D}}(h)| \leq \sqrt{\frac{\log(2/\delta)}{2m_v}}$$

Validation set of Model selection

Here we explain how to use the validation set in model selection [37]. First we train different algorithms (or the same algorithm with different hyperparameters) on the training set and then we yield into

$$\mathcal{H}' = \{h_1, \ldots, h_r\}$$

a set of the output predictors of the different algorithms. Then, to choose a single predictor of $\mathcal{H}'$ w sample a fresh validation set and choose the predictor that minimizes the error over the validation set.

The process is some what similar to train over a finite hypothesis class, the only different is that $\mathcal{H}'$ is not fixed in advance, but rather it depends on the training set.

Now, from Theorem 6 together with the union bound we have the following theorem

Theorem 7 *Let $\mathcal{H}'$ be an arbitrary set of predictors and assume that the loss function is in $[0, 1]$. Assume that a validation set V of size m_v is sampled independent of $\mathcal{H}'$. Then, with probability of at least $1 - \delta$ over the choice of V we have*

$$\forall h \in \mathcal{H}',\ |h_{\mathcal{D}}(h) - L_V(h)| \leq \sqrt{\frac{\log(2|\mathcal{H}'|/\delta)}{2m_v}}$$

Cross-Validation (CV)

So far the process of the use of validation set assumes that we split the data into three parts: *training set*, *validation set*, and *test set* (as described in Figure 4.1). The test set is not involved in the choice of the optimal hypothesis h^*. However, sometimes the data is not plentiful and we cannot afford to drop part of it to build the validation set. The solution to this is to use the concept of *k-Fold Cross Validation*.

In the k-fold cross validation the original training set of m samples is partitioned into k subsets (folds) of size m/k. For each fold: · the algorithm is trained on the union of the other folds, and then · the error of its output is estimated using the selected fold (see Figure 4.2 for illustration). Finally, the average of all these errors is the estimate of the

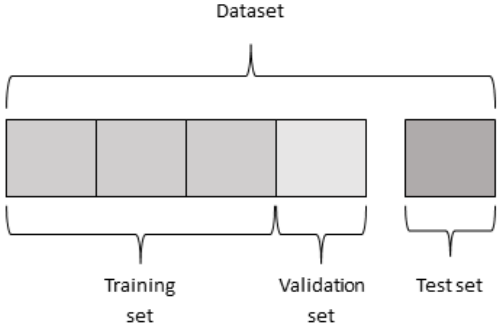

Figure 4.1: Dataset split into: training, validation, and testing

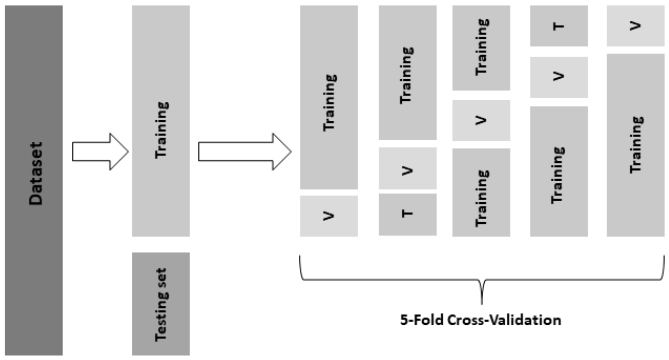

Figure 4.2: 5-Fold Cross-Validation (T for training set, V for validation set)

true error. A special case is when $k = m$, is called *leave-one-out* (LOO) [37].

Algorithm 1 describes the steps of K-fold Cross Validation for model selection taken from [37]

Algorithm 1 Algorithm for 5-Fold Cross-Validation for model selection

Input:

- Training set $\mathcal{S} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$.
- Set of parameter values Θ.
- Learning algorithm A.
- Integer k.

Partition S into $S_1, S_2, \ldots, S_k$.
foreach $\theta \in \Theta$
for $i = 1, \ldots, k$ **do**
    $h_{i,\theta} = A(S \setminus S_i; \theta)$
end for
$error(\theta) = \frac{1}{k} \sum_{i=1}^{k} L_{S_i}(h_{i,\theta})$
Output:

- $\theta^* = \text{argmin}_\theta[error(\theta)]$
- $h_{\theta^*} = A(S; \theta^*)$

Stochastic Gradient Descent (SGD)

Minimizing a differentiable convex function $f(w)$ with respect to the weight vector w is a fundamental problem in optimization, underpinning many machine learning algorithms. While classical gradient descent provides an exact update direction using the full gradient, it becomes computationally prohibitive for large-scale problems where data dimensions and sample sizes are immense. To address this challenge, stochastic gradient descent (SGD) offers a practical alternative by estimating the gradient using small, randomly selected subsets of data, thus significantly reducing computational overhead.

Remark 6 *Here are some advanced schemes of SGD that are commonly used in ML:*

- *Adagrad: it adapts the learning rate for each parameter independently. There are also Adadelta and RMSprop other improved version of Adagrad.*

- *Adam (Adaptive Moment Estimation): it also computes adaptive learning rate for each parameter. It combines ideas from Adagrad and momentum.*

4.2 Neural Networks

Neural networks (NNs) are computational models inspired by the architecture of the human brain. In these models, *neurons* are represented as nodes within a directed graph, while the connections between these nodes are depicted as *edges*. These edges symbolize the pathways through which information flows between neurons. This structure allows NNs to process and learn from data in a manner somewhat analogous to how the brain processes information [37]. An example of such a neural network structure is illustrated in Figure 4.3.

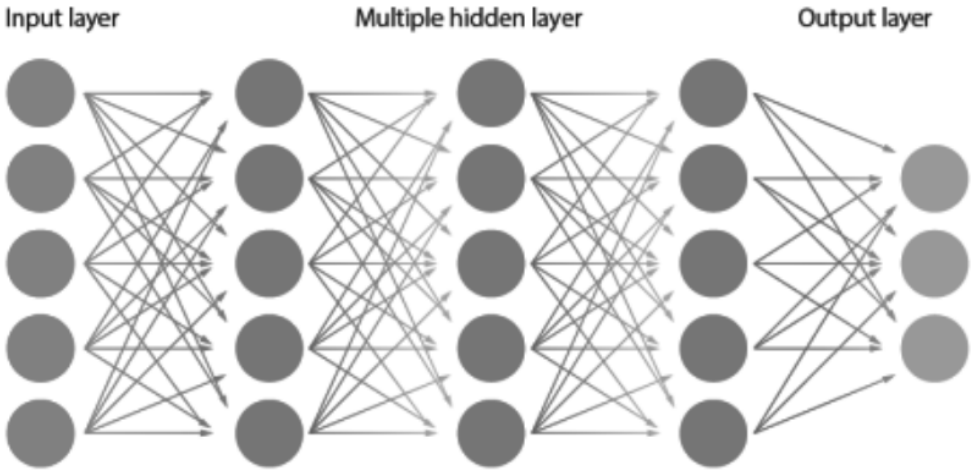

Figure 4.3: An example of a neural network (taken from [24])

NNs were initially proposed in the 1940s and 1950s. Their first practical applications emerged in the 1980s and 1990s, though these early applications were quite elementary. It was not until the 2010s, with the advancement of deep architectures that neural networks began to achieve impressive performances.

A **feedforward** neural network is a type of neural network characterized by a graph with no cycles, meaning that data flows in only one direction. Typically, the network is organized into *layers*, where each layer contains several neurons. Each neuron in a given layer receives input from the neurons in the preceding layer.

Formally [37], we can represent a neural network as a graph $G = (V, E)$ where V is the set of neurons and E is the set of connections between neurons, represented as directed edges. And a weight function $w : E \to \mathbb{R}$ that is defined over the edges, where the weights w are the parameters to be learned. And there is the *activation function* $\sigma : \mathbb{R} \to \mathbb{R}$ which is a nonlinear function that is applied to the output of a neuron to introduce non-linearity into the network, enabling it to learn complex patterns. Common activation functions include: *Sigmoid*: $\sigma(x) = \frac{1}{1+e^{-x}}$, which maps inputs to $(0, 1)$, *Hyperbolic Tangent (tanh)*: $\sigma(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$, which maps inputs to $(-1, 1)$, and *Rectified Linear Unit (ReLU)*: $\sigma(x) = \max(0, x)$, which maps negative inputs to zero and keeps positive inputs unchanged.

Let us represent the network as the union of the layers: $V = \bigcup_{t=0}^{T} V_t$, where T is the number of layers or the depth of the network:

- V_0 is the input layer, V_T is the output layer, and $V_1, \ldots, V_{T-1}$ are the hidden layers.
- Each neuron receives a weighted sum of outputs from the previous layer and applies an activation function σ to produce its output.

The output of a neuron j in layer $t+1$ can be expressed as:

$$o_{t+1,j}(x) = \sigma\left(w_{0j}^{(t+1)} + \sum_{i=1}^{|V_t|} w_{ij}^{(t+1)} o_{t,i}(x)\right) \tag{4.1}$$

where $w_{ij}^{(t+1)}$ is the weight of the connection from neuron i in layer t to neuron j in layer $t+1$, $w_{0j}^{(t+1)}$ is the bias term, and σ is the activation function (see Figure 4.4).

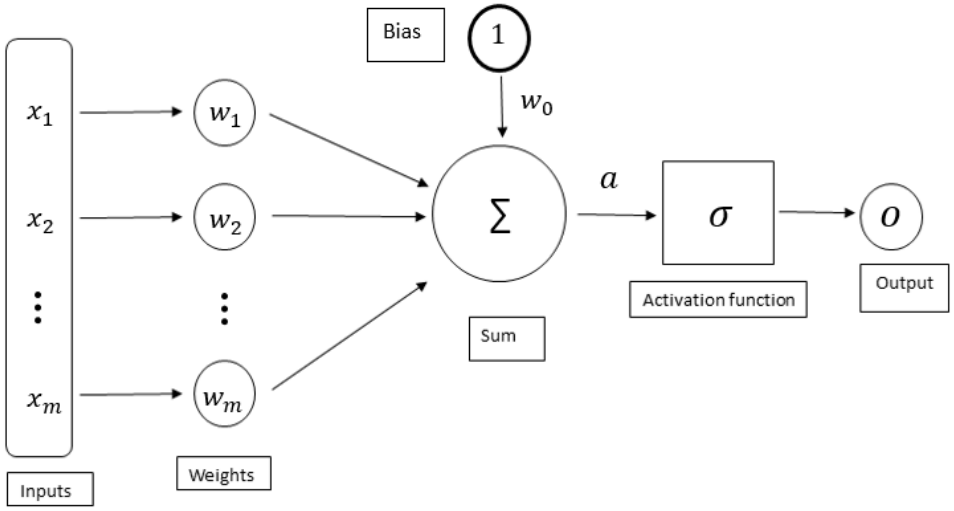

Figure 4.4: An example of a neural network with regularization layers

This process of passing input data through a neural network to generate an output is called *forward propagation*. It begins at the input layer (V_0), where the input data is processed. The output from this layer is then passed to the next layer (V_1), where it is further computed and sent to subsequent layers (V_2, V_3, and so on) until it reaches the final output layer (V_T). This sequential computation across layers continues until the network produces the final output.

Remark 7 *(Regularization layers)*

4.2.1 Training of the Neural network

Once we have specified a neural network by (V, E, σ, w), we define a function $h_{V,E,\sigma,w} : \mathbb{R}^{|V_0|-1} \to \mathbb{R}^{|V_T|}$ that maps input data to output predictions. The combination of the components (V, E, σ) is the network's architecture, while w represents the learnable parameters.

For a given loss function $\Delta(h_w(x), y)$ that measures the discrepancy between the prediction $h_w(x)$ and the true value y, the learning task is to find parameters w that minimize the expected loss:

$$L_{\mathcal{D}}(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\Delta(h_w(x), y)] \tag{4.2}$$

where $\mathcal{D}$ is the data distribution.

Backpropagation

Backpropagation is the primary algorithm used to compute the gradient of the loss function with respect to the network weights, which is then used in gradient descent optimization.

The algorithm consists of two main phases:

1. **Forward Pass**: Compute the output of each neuron layer by layer, starting from the input.

2. **Backward Pass**: Compute the gradient of the loss with respect to each weight by propagating the error backward through the network.

For a neural network with a loss function such as squared error $\Delta(h_w(x), y) = \frac{1}{2}||h_w(x) - y||^2$, the backpropagation algorithm efficiently calculates the gradient $\nabla_w L_{\mathcal{D}}(w)$ through the chain rule of calculus.

The key insight of backpropagation is that the gradient computation can be decomposed and reused, starting from the output layer and working backward:

1. For the output layer, calculate the error: $\delta_T = o_T - y$
2. For each previous layer t, propagate the error: $\delta_t = (W_{t+1}^T \delta_{t+1}) \cdot \sigma'(a_t)$
3. The gradient for each weight is then: $\frac{\partial L}{\partial w_{ij}^{(t)}} = \delta_{t,j} \cdot o_{t-1,i}$

where a_t represents the weighted inputs to layer t, σ' is the derivative of the activation function. With the gradients calculated, the weights can be updated using gradient descent:

$$w^{(t+1)} = w^{(t)} - \eta \nabla_w L_{\mathcal{D}}(w^{(t)}) \tag{4.3}$$

where η is the learning rate.

Through this iterative process, the neural network learns to minimize the loss function and improve its predictions on the training data.

Convolutional Neural Networks (CNN)

Traditional fully connected feedforward neural networks face significant limitations when processing structured data like images. When images are flattened into one-dimensional vectors, these networks not only require an excessive number of parameters due to their full connectivity structure, but they also lose crucial spatial relationships and local patterns inherent in the image data, potentially leading to inefficiency and overfitting issues. Convolutional Neural Networks (CNNs) were developed to address these challenges through three key architectural innovations: *local connectivity*, which allows each neuron to connect only to a small region of the input; *weight sharing*, which reduces parameters by reusing the same weights across different input locations; and *multiple feature maps* that capture different patterns in the data. Additionally, CNNs employ pooling layers that reduce dimensionality while preserving important features, thereby enhancing computational efficiency. For further details about CNNs refer to [21].

4.3 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a method that is particularly useful for data compression. In PCA both the compression and the reconstruction of the original data are achieved using linear transformations. The lower dimensional data should be a good approximation of the higher dimensional representations. This compression can enhance computational efficiency and reduce storage requirements.

The goal of the PCA is to find the linear map that minimizes the mean square error between the original data and the corresponding reconstructed data (see Figure 4.5). Mathematically,

Assume we have the data points $x_1, x_2, \ldots, x_m \in \mathbb{R}^d$ and let $W \in \mathbb{R}^{n \times d}, (n < d)$ be a mapping

$$\begin{aligned} W : \mathbb{R}^d &\to \mathbb{R}^n \\ x &\mapsto y = Wx \end{aligned}$$

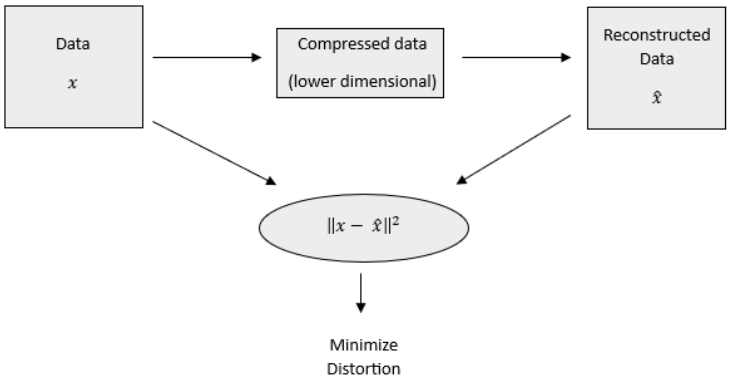

Figure 4.5: Principal Component Analysis (PCA)

where $y = Wx \in \mathbb{R}^n$ is a lower dimensional representation of $x \in \mathbb{R}^d$. Now, let $U \in \mathbb{R}^{d \times n}, (n < d)$ be defined as

$$\begin{aligned} U : \mathbb{R}^n &\to \mathbb{R}^d \\ y &\mapsto \hat{x} = Uy \end{aligned}$$

the inverse mapping used to recover an approximation $\hat{x} = Uy = UWx$ of x.

The goal: Find the lower dimensional representation that better approximate the data while minimizing the square distance between x and $\hat{x}$

$$\underset{\substack{W \in \mathbb{R}^{n \times d} \\ U \in \mathbb{R}^{d \times n}}}{\operatorname{argmin}} \sum_{i=1}^{m} \|x_i - UWx_i\|_2^2 \tag{4.4}$$

Lemma 1 *There exists an optimal solution (U^*, W^*) of 4.4 where*

- *the columns of U^* are orthonormal (i.e. $(U^*)^T U^* = I$),*
- $W^* = (U^*)^T$

the details of the proof can be found in [37].

Now we can rewrite the optimization problem in 4.4 as follows

$$\underset{U \in \mathbb{R}^{d \times n} : U^T U = I}{\operatorname{argmin}} \sum_{i=1}^{m} \|x_i - UU^T x_i\|_2^2 \tag{4.5}$$

For $x \in \mathbb{R}^d$ and U such that $U^T U = I$ we have

$$\begin{aligned} \|x - UU^T x\|_2^2 &= \|x\|_2^2 - x^T UU^T x \\ &= \|x\|_2^2 - trace(x^T UU^T x) = \|x\|_2^2 - trace(U^T x x^T U) \end{aligned}$$

then we can write our optimization problem as follows:

$$\underset{U \in \mathbb{R}^{d \times n} : U^T U = I}{\operatorname{argmax}} trace\left(U^T (\sum_{i=1}^{m} x_i x_i^T) U\right) = \underset{U \in \mathbb{R}^{d \times n} : U^T U = I}{\operatorname{argmax}} trace\left(U^T A U\right) \tag{4.6}$$

where $A = \sum_{i=1}^{m} x_i x_i^T$ is a symmetric positive semi-definite matrix, and hence it can be written as

$$A = VDV^T$$

where D is a diagonal matrix and $V^T V = VV^T = I$. The elements of the diagonal of D are the eigenvalues of A and the columns of V are the corresponding eigenvectors. In Algorithm 2 a pseudocode of the PCA from [37].

Now we can assert the following theorem that state the solution of 4.6 is the matrix U whose columns are the n eigenvectors of A that are corresponding to the largest n eigenvalues (the proof in [37]).

Algorithm 2 Pseudocode for PCA

1: **Input:**

- $X \in \mathbb{R}^{m \times d}$: a matrix that contains m samples $\mathbb{R}^d$.
- n: number of components.

2: **Algorithm:**

- Compute $A = X^T X$.
- Perform eigenvalue decomposition of A.
- Let $u_1, \ldots, u_n$ be the eigenvectors of A corresponding to the largest eigenvalues.

3: **Output:** $u_1, \ldots, u_n$.

Theorem 8 *Let $x_1, x_2, \ldots, x_m$ be arbitrary vectors in $\mathbb{R}^d$. Let $A = \sum_{i=1}^{m} x_i x_i^T$. And let $u_1, u_2, \ldots, u_n$ be n eigenvectors of A corresponding to the largest n eigenvalues of A. Then a solution of the PCA optimization 4.6 is the to set U to the matrix whose columns are $u_1, u_2, \ldots, u_n$ and to set $W = U^T$.*

4.4 Hyperparameter Tuning

Before exploring hyperparameter tuning techniques, it is essential to distinguish between *hyperparameters* and *model parameters* [29]. Model parameters are determined by the nature of the data and are learned during the training process. These parameters cannot be controlled directly, as their values depend entirely on the dataset. For instance, in a linear equation (see the equation bellow), parameters like the slope m and the intercept C are coefficients that are learned from the data. Hyperparameters, on the other hand, are parameters that control the behavior of the model or algorithm. Unlike model parameters, they are adjustable and can be fine-tuned to optimize the model's performance (for example, λ in the linear equation bellow is a hyperparameter). In essence, hyperparameters can be thought of as the model's *configuration*. A machine learning algorithm can truly excel when the optimal combination of hyperparameters is selected. While choosing the right algorithm for the problem is essential, fine-tuning the hyperparameters is equally crucial for achieving the best possible performance.

$$\sum (y_i - (mx_i - C))^2 + \lambda|\omega|$$

Steps for Hyperparameter Tuning:

1. **Model Selection**: Begin by selecting the appropriate model or algorithm that aligns with the problem at hand.
2. **Define Hyperparameter Space**: Identify the set of hyperparameters associated with the chosen model. Construct the *hyperparameter space* by specifying the ranges or values over which the hyperparameters will be explored.
3. **Choose a Search Method**: Select a suitable search strategy for tuning the hyperparameters, such as grid search, random search, or more advanced techniques like Bayesian optimization (we will discuss these techniques in the following sections).
4. **Apply Cross-Validation**: Implement a cross-validation strategy, such as k-fold cross-validation, to ensure that the model performance is evaluated reliably across different data subsets.
5. **Model Evaluation**: Assess the performance of the model using an appropriate evaluation metric. For classification problems, this could be the accuracy, calculated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

4.4.1 Classic Hyperparameters Tuning Techniques in ML

We briefly explore some generic and classic approches of hyperparameter tuning, namely: *Manual Search, Grid Search,* and *Randomized Search.*

For the Manual Search, as the name suggest, the user manually selects and tests different combinations of hyperparameters based on intuition or prior experience. While this method can be useful when the hyperparameter space is simple, it is often impractical for more complex models where there are many hyperparameters and possible values. Manual search can be time-consuming and inefficient but may still be valuable when the user has a strong understanding of the model's behavior.

Grid Search is a systematic, exhaustive search process where the algorithm evaluates all possible combinations of hyperparameters within a predefined grid. The grid consists of hyperparameter values that the user specifies beforehand, and the model is trained and evaluated for every combination in this grid. Although this approach ensures that every possible combination is tested, it can be computationally expensive, especially when dealing with large models or high-dimensional hyperparameter spaces.

Randomized Search, unlike Grid Search, does not evaluate the model for every possible combination of hyperparameters. Instead, it randomly selects combinations of hyperparameters from the specified ranges and tests them. This approach is much more efficient in terms of computation time, especially when the hyperparameter space is large or when the model is complex. Although it does not guarantee the optimal combination is found, it often yields near-optimal results with fewer evaluations.

4.4.2 Bayesian Optimization

Bayesian Optimization offers a more time- and memory-efficient approach to hyperparameter tuning compared to the classical methods. In essence, it enables efficient discovery of the best options by exploiting the knowledge gained from past evaluations.

It achieves this by balancing two key phases: exploration and exploitation.

1. **Exploration Phase:** During exploration, the algorithm searches different regions of the hyperparameter space to gather insights into how the objective function behaves. The aim is to investigate untested areas, potentially identifying promising regions where the model could perform well, even if those areas have not been evaluated before.

2. **Exploitation Phase:** In the exploitation phase, the focus shifts to optimizing hyperparameters in the regions that have already demonstrated good performance. The algorithm leverages the data from the exploration phase to refine the search, concentrating on regions that are more likely to yield improved results based on past observations.

Bayesian optimization efficiently balances both phases to find the optimal hyperparameters with minimal computational cost. To apply it in practice, we start by defining an objective function that includes the ML model, which returns a score based on cross-validation results.

Popular tools for implementing Bayesian optimization include *Hyperopt*, *Optuna*, and *SMAC*.

We mention that integer parameters are usually treated in this context as integers, which are then rounded at each iteration at the closest integer value.

4.4.3 Tuning Hyperparameters and Layers in Deep Learning (DL)

The optimization of hyperparameters in deep learning is essential for the neural network performance and generalization capability [33]. Bayesian optimization framework works efficiently for hyperparameter tuning in deep learning.

The architecture and performance of neural networks are governed by several key hyperparameters:

1. Network Architecture hyperparameters:

    - The dimension of hidden layers (number of neurons) is proportional to task complexity, requiring careful balance to avoid underfitting or unnecessary computational complexity.

- The depth of the neural network (number of hidden layers) depending on the complexity of the problem.
- Adding regularization layers to prevent overfitting risk. For instance Batch Normalization and dropout layer (See Figure 4.6).

2. Training hyperparameters:
   - Learning rate which affects convergence dynamics, with higher rates we accelerate the training at the risk of optimization instability.
   - Batch size
   - Training epoch
   - Optimizer selection
3. Activation functions

The process of hyperparameter tuning, while computationally intensive, is fundamental to achieving optimal model performance and generalization capabilities.

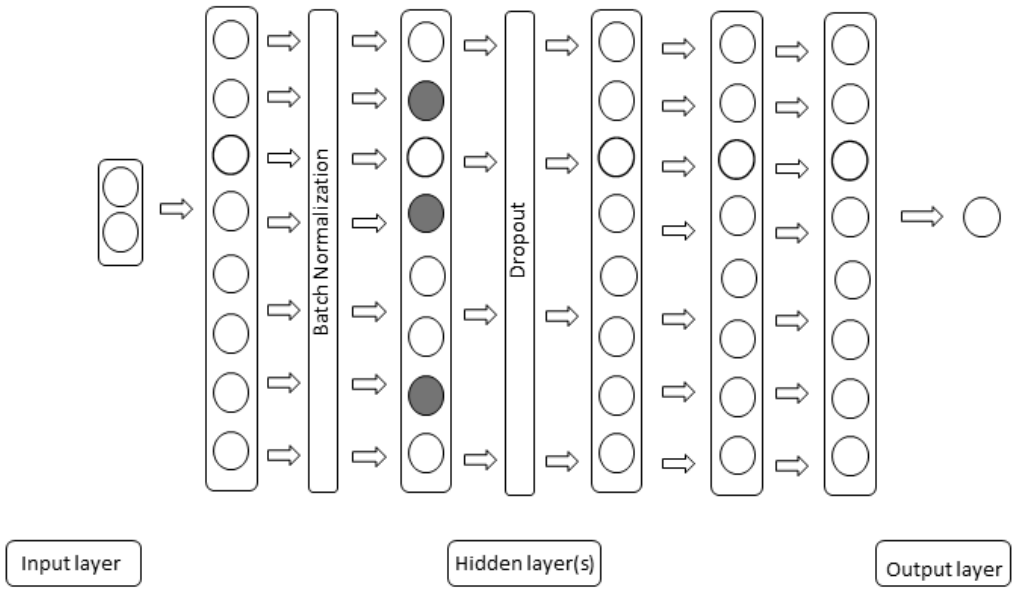

Figure 4.6: An example of a neural network with regularization layers

4.5 Data Generation Models

Data generation becomes an important field that seeks to model the inherent distribution of existing data to produce new yet similar data. This field is rapidly evolving and has extensive applications in areas such as molecule design, image editing, text generation, and speech synthesis [46].

4.5.1 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) represent a revolutionary class of artificial intelligence algorithms that have taken the field of machine learning by storm. Introduced by Ian Goodfellow and his colleagues in 2014 [22], GANs have demonstrated impressive capabilities in generating realistic and diverse data, such as images, audio, and even text.

The GAN framework involves a dynamic interaction between the generator and discriminator networks. The generator's objective is to produce data instances that are indistinguishable from real

data, whereas the discriminator is responsible for identifying genuine data from generated data. This process resembles a game between the two networks, where the generator constantly improves its ability to produce realistic data in response to the feedback provided by the discriminator. Over time, this iterative process ideally leads to the generation of data that is so realistic that even the discriminator struggles to tell it apart from real data. However, achieving this balance can be challenging and requires careful tuning of hyperparameters, architecture, and training techniques.

GANs have been applied to a wide array of tasks with impressive results. As in many different AI frameworks such as, e.g., photo realistic image generation [8], image-to-image translation [50], text-to-image translation [48], photo editing [31], super resolution [26], and image inpainting [30].

Despite their remarkable capabilities, GANs do come with challenges such as training instability and mode collapse, which we will be discussing in this section.

4.5.2 First introduction to GANs

In this paper [22], the authors investigate a specific scenario where the generative model produces samples through a multilayer perceptron by introducing random noise. Likewise, the discriminative model is constructed as a multilayer perceptron. In this particular instance, both models can be trained exclusively using the well-established backpropagation technique. Additionally, we can extract samples from the generative model using forward propagation.

As we discussed so far, GANs are designed to tackle the fundamental question: Given a dataset of objects sharing a certain level of consistency—like images of cats, handwritten Chinese characters, or Van Gogh paintings—can we artificially produce similar objects? Mathematically [47], when we say the objects in dataset $\mathcal{X}$ exhibit a consistent pattern, we imply they are samples drawn from a common probability distribution μ in $\mathbb{R}^d$ (assuming objects are points in $\mathbb{R}^d$).

The notion of 'similar objects' relates to their likeness within the context of probability distribution. Two datasets are considered similar if they are derived from the same or nearly identical probability distribution. In essence, our training dataset $\mathcal{X}$ is a subset of $\mathbb{R}^d$, composed of samples originating from the probability distribution μ (characterized by density $p_{data}(x)$). Our objective is to discover a probability distribution ν (with density $p_g(x)$) that closely approximates μ. By generating samples from ν, we obtain artificial objects that closely resemble those in $\mathcal{X}$.

The adversarial game described above can be formulated *mathematically* by minimax of a target function between the *discriminator function* $D(x;\theta_d)$ where $D : \mathbb{R}^d \to [0,1]$ is a differentiable function depends on parameters θ_d, D yields a single scalar output, and the *generator function* $G(z;\theta_d)$ where $G : \mathbb{R}^d \to \mathbb{R}^d$ is also a differentiable function. $D(x)$ signifies the likelihood that x originated from the data rather than p_g. The training involves maximizing D's accuracy in labeling both training examples and samples from G. Simultaneously, G is trained to minimize $log(1 - D(G(z)))$. In order to grasp the distribution p_g of data x generated by the generator, we define a prior on input noise variables $p_z(z)$, In [22] the target *loss function* is proposed to be:

$$V(D,G) = \mathbb{E}_{x\sim p_{data}} \log D(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D(G(z))), \tag{4.7}$$

where $\mathbb{E}_\mu(f) = \int f d\mu$ is the expectation functional of f with respect to the probability measure μ. Hence, GANs solve the minimax problem:

$$\min_G \max_D V(D,G) = \min_G \max_D \left\{ \mathbb{E}_{x\sim p_{data}} \log D(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D(G(z))) \right\} \tag{4.8}$$

As stated in [22], in practice, it is necessary to adopt an iterative, numerical method to implement the game. Fully optimizing D within the inner loop of training is impractical due to its high computational cost and the risk of overfitting on finite datasets. As an alternative, we employ a strategy where we iteratively alternate between optimizing D for k steps and optimizing G for one step. This approach ensures that D remains close to its optimal solution, provided that G's changes are gradual. The step-by-step process is outlined in Algorithm 3 as presented formally in [22].

Algorithm 3 Training generative adversarial networks using minibatch stochastic gradient descent. The hyperparameter k, which denotes the number of iterations to update the discriminator, was set to 1, representing the most computationally efficient choice.

for number of training iterations **do**
  for k steps **do**
    Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
    Update the discriminator:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \right]$$

  end for
  Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
  Update the generator:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$$

end for

Now we address some theoretical results stated in [22], and for more details for the proofs, refer to their paper.

Proposition 9 *For G fixed, the optimal discriminator D is*

$$D^*_G(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(x)}$$

Setting $C(G) = \max_D V(G, D)$, and we have the following theorem

Theorem 9 *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that moment, $C(G)$ attains a value of $-log4$.*

As in the proof of the above theorem, with simple calculations we get:

$$C(G) = -log(4) + 2JS(p_{data}|p_g) \tag{4.9}$$

where JS is the Jensen-Shannon divergence. Hence, GANs are actually minimizing the Jensen-Shannon divergence between p_{data} and p_g.

Once the optimization of G and D is completed, the generator G can be used to sample objects by first sampling z according to the distribution of the input, and then computing $G(z)$.

4.5.3 Understanding GANs training dynamics

As we said earlier, GANs are still very difficult to train. In [4] authors focus on rigorous examination of issues like instability, saturation during GANs training, and mode collapse (it occurs when the generator of the GAN starts producing a limited set of very similar or identical samples). As refer in the paper [22], when the discriminator gets better, the updates to the generator get consistently worse. This situation can only occur when the distributions are not continuous or possess separate supports. A potential reason for the distributions lacking continuity is if their supports are lying on low-dimensional manifolds.

In theorem 2.1, 2.2 in [2], they prove that in both cases where the two distributions have disjoint support and where their support lie on low-dimensional manifolds, then there is exist a **perfect discriminators** which are smooth and constant almost everywhere (by perfect discriminator we mean, $p_{data}[D(x) = 1] = 1$ and $p_g[D(x) = 0] = 1$.

First issue that arise is that, as the discriminator gets better, the gradient of the generator vanishes (as stated and proved in Theorem 2.4 [4]). So, to prevent gradient vanishing in situations where the discriminator is highly confident, a different gradient step is used for the generator, as follow

$$\nabla_{\theta_g} \mathbb{E}_{z\sim p(z)} \left[-\log D(G(z)) \right]$$

Running experiments with this $-\log D$ cost function, authors observed that the gradient norms grow quickly (for details and proof see Theorem 2.6 in [3]).

The authors in [3] suggest to use and alternative divergence or distance for the GANs, they chose to use Wasserstein distance (see (1.7)), and it was the first time to introduce *Wasserstein GANs* (WGANs).

4.5.4 First introduction to WGANs

WGANs address the primary training challenges encountered in traditional GANs. Specifically, when training WGANs, there is no need to carefully manage the balance between training the discriminator and generator. Additionally, WGANs reduce the common issue of mode collapse that is frequently observed in GANs. The provided example in [4] (Example 1) demonstrates how seemingly straightforward sequences of probability distributions converge when considering the Wasserstein distance, but fail to converge when using different distances and divergences (the comparison was between Wasserstein distance, Total Variation distance, Kullback-Leibler divergence, and Jensen-Shannon divergence).

With mild conditions we can show that Wasserstein distance $W(p_{data}, p_{\theta_g})$ is a continuous loss function on θ_g, which makes it much more sensible cost function for the GANs problem than at least the Jensen-Shannon divergence, as we see in the following theorem, , but first we need to set the following assumption

Assumption 1. Let $g : \mathcal{Z} \times \mathbb{R} \to \mathcal{X}$ be locally Lipschitz between finite dimensional vector spaces. We will denote $g_\theta(z)$ it is evaluation on coordinates (z, θ). We say that g satisfies assumption 1 for a certain probability distribution p over $\mathcal{Z}$ if there are local Lipschitz constants $L(\theta, z)$ such that

$$\mathbb{E}_{z\sim p} \left[L(\theta, z) \right] < +\infty$$

Theorem 10 *Consider a fixed distribution p_r defined over the space $\mathcal{X}$. Additionally, let Z be a random variable, for example, following a Gaussian distribution, defined over a separate space denoted as $\mathcal{Z}$. We have a function $g : \mathcal{Z} \times \mathbb{R}^d \to \mathcal{X}$, which we will denote as $g_\theta(z)$, where z represents the first coordinate and θ the second. Let p_θ represent the distribution of the random variable $g_\theta(Z)$. Then:*

- *If g is continuous in θ, so is $W(p_r, p_\theta)$.*
- *If g is locally Lipschitz and satisfies regularity assumption 1, then $W(p_r, p_\theta)$ is continuous everywhere, and differentiable almost everywhere.*
- *the statements above are false for the Jensen-Shannon divergence $JS(p_r, p_\theta)$ and all the KLs*

This makes the Wasserstein distance to have nicer properties when optimized. Indeed, the function g is in practice represented by a neural network with parameters θ that need to be optimized, usually by gradient-based methods. The theorem then guarantees that $W(p_r, p_\theta)$ can be used as a loss function to be minimized, since the smoothness of g carries over to that of W. This is in constrast with the JS divergence. Nonetheless, finding the infimum in equation (1.8) poses a significant computational challenge. So let us recall Kantorovich-Rubinstein duality [44] (also see (1.4))

$$W_1(p_{data}, p_{\theta_g}) = \sup_{||f||_L \leq 1} \mathbb{E}_{x \sim p_{data}}[f(x)] - \mathbb{E}_{x \sim p_{\theta_g}}[f(x)] \tag{4.10}$$

Moreover, the topology induced by the Wasserstein distance is the weakest among the other divergences and distances (also refer to Section 1.3.1), giving it the edge for practical real-world applications. We state the following theorem (detailed proof can be found in [2]).

Theorem 11 *Let P denote a distribution defined on a compact space $\mathcal{X}$, and let $(P_n)_{n \in \mathbb{N}}$ be a sequence of distributions on the same space $\mathcal{X}$. Considering all limits as n approaches infinity, the following statements hold:*

1. *The following statements are equivalent:*
   - *$\delta(p_n, p) \to 0$ with δ the total variation distance.*
   - *$JS(p_n, p) \to 0$ with JS the Jensen-Shannon divergence.*
2. *The following statements are equivalent:*
   - *$W(p_n, p) \to 0$.*
   - *$p_n \xrightarrow{\mathcal{D}} p$ where $\xrightarrow{\mathcal{D}}$ represent convergence in distribution for random variables*
3. *$KL(p_n||p) \to 0$ or $KL(p||p_n) \to 0$ imply the statement in 1.*
4. *The statements in 1 imply the statements in 2.*

We now address the following theorem which consider differentiating $W_1(p_r, p_\theta)$ [3]

Theorem 12 *Let p_r be any distribution. Consider p_θ as the distribution of $g_\theta(Z)$, where Z is a random variable with density p and g_θ a function satisfying assumption 1. Then, there is a solution $f : \mathcal{X} \to \mathbb{R}$ to the problem*

$$\max_{||f||_L \leq 1} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{x \sim p_\theta}[f(x)]$$

and we have

$$\nabla_\theta W_1(p_r, p_\theta) = \mathbb{E}_{z \sim p(z)}[\nabla_\theta f(g_\theta(z))]$$

when both terms are well-defined.

Observe that when we substitute the condition $||f||_L \leq 1$ with $||f||_L \leq K$ (K-Lipschitz with a constant K), we obtain $W_1(p_r, p_\theta)$ up to a multiplicative constant. Consequently, if we work with a parametric set of functions $\{f_w\}_{w \in \mathcal{W}}$ that all satisfy the K-Lipschitz condition for some K, we can work with the following problem:

$$\max_{w \in \mathcal{W}} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p(z)}[f_w(g_\theta(z))] \tag{4.11}$$

we call f_w the 'critic', it is an analogue to the discriminator in GANs. In order to maintain parameters w lie in a compact space $\mathcal{W}$ and hence enforce the Lipschitz constraint, in [3] they use *weight clipping* after each gradient update. Here we display the algorithm for WGANs

The very first benefit of WGANs is that it provides an estimate of EM that correlates well with the quality of the generated samples. And The better the critic, the higher quality the gradients we use to train the generator. So we no longer need to balance generator and discriminator's training [3].

Algorithm 4 WGAN

Require: α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic. w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ has not converged **do**
  for $t = 0, \ldots, n_{critic}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim p_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot RMSProp(w, g_w)$
    $w \leftarrow clip(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot RMSProp(\theta, g_\theta)$
end while

4.5.5 Improving WGANs Training

The proposed WGAN represents a step towards achieving more stable GAN training. However, it can encounter issues where it generates low-quality samples or struggles with convergence. In [23] they have identified that these problems frequently arise from the employment of weight clipping in WGAN, which is employed to enforce a Lipschitz constraint on the critic. The authors propose an alternative way to enforce the Lipschitz constraint, which is through *gradient penalty*. A function is considered 1-Lipschitz if and only if its gradients have a magnitude no greater than 1 everywhere. Therefore, they consider the direct restriction of the gradient norm of the critic's output with respect to its input. The new objective functional as in [23] is

$$L = \mathbb{E}_{x\sim p_g}\left[f(x)\right] - \mathbb{E}_{x\sim p_r}\left[f(x)\right] + \lambda\, \mathbb{E}_{z\sim p(z)}\left[(||\nabla_z f(z)||_2 - 1)^2\right] \tag{4.12}$$

where the first two terms is the original critic loss, and the last term is the gradient penalty.

The proposed approach offers an edge over weight clipping by enhancing both the training speed and the quality of generated samples. And it preserves the property of correlating the sample quality and the convergence toward a minimum.

4.5.6 How Accurately Do WGANs Approximate the Wasserstein Distance?

As we have seen, Wasserstein distance is expressed using Kantorovich duality as the difference between expected values of a potential function under real data and model distributions. As mentioned in [28], this introduces at least three sources of errors in the approximation of the Wasserstein distance: the approximated discriminator and constraints, the estimation of the expectation value, and the optimization. The authors consider the c-transform formulation, which allows for more accurate estimation of the Wasserstein distance. But the surprising finding in their research is that the approach that most accurately approximates the Wasserstein distance does not yield the best looking images in the generative setting.

To describe the objective function in the $c-$transform technique of approximating the Wasserstein distance, recall definition 1, then we can compute the $c-$transform over the minibatches as

$$\phi_\omega^c(y_i) \approx \widehat{\phi_\omega^c}(y_i) = \min_j\{c(x_j, y_i) - \phi_\omega(x_j)\} \tag{4.13}$$

The objective is written as

$$\max_{\omega}\left\{\frac{1}{N}\sum_{i=1}^{N}\phi_{\omega}(x_i)+\frac{1}{N}\sum_{i=1}^{N}\widehat{\phi_{\omega}^{c}}(y_i)\right\} \tag{4.14}$$

Authors in [28] runs many experiments, and the outcomes clearly demonstrate that, at each iteration, the $c-$transform provides more accurate estimates of the Wasserstein distances compared to gradient penalty or weight clipping methods. However, the resulting images are blurry, whereas the highest-quality images are generated using the gradient penalty method, while weight clipping has not yet reached a convergence.

The authors raises many questions upon the results they got. First, the question of whether the precise Wasserstein-1 distance between batches is indeed the quantity that should be considered in the context of generative modeling?

Furthermore, it is intriguing to observe how the gradient penalty method excels in the generative scenario, despite its somewhat less precise approximation of the Wasserstein-1 distance, as indicated by their experiments. This raises the question of what attributes make it such an effective objective in the generative context?

In our work we want to use the formula of calculating the Wasserstein-1 distance as in Proposition 8, and see how it will impact the performance of WGANs.

4.5.7 Autoencoders

Autoencoders (AEs) were initially introduced in [34] as a type of neural network designed to reconstruct their input. Their primary objective is to learn an informative representation of data in an unsupervised manner, which can be applied to tasks like clustering. The formal problem, as defined in [5]: AE is composed of two components: an **encoder** function $e_\theta : \mathcal{X} \to \mathcal{Z}$ and a **decoder** function $d_\phi : \mathcal{Z} \to \mathcal{X}$, minimizing the following loss (the difference between the input x and its reconstruction $\hat{x}$)

$$||x - \hat{x}||_2 = ||x - d_\phi(e_\theta(x))||_2 \tag{4.15}$$

The encoder transforms the input data from a high-dimensional space to a **latent space**, which may have a smaller dimension than the input. This achieves data compression when the latent space is lower-dimensional. Conversely, the decoder reconstructs the data by converting it from the latent space back to the original high-dimensional space, Figure 4.7 illustrate the autoencoder model.

When the encoder e_θ and the decoder d_ϕ are restricted to be linear operations, this results in a linear autoencoder [5]. In that case, it produces the same latent representation as **Principal Component Analysis (PCA)** [32]. PCA is a linear transformation statistical technique. It transforms the original variables into a new set of variables (the principal components) that are uncorrelated and ordered by the amount of variance they explain. Thus, an autoencoder extends the concept of PCA by not just identifying a low-dimensional hyperplane where the data lie, but also by learning a nonlinear manifold.

However, AEs cannot be used for data generation. While an AE can compress input data by eliminating redundancy (encoder), the non-regularized latent space prevents the decoder from generating valid input data or creating new, diverse samples that differ statistically from the training data.

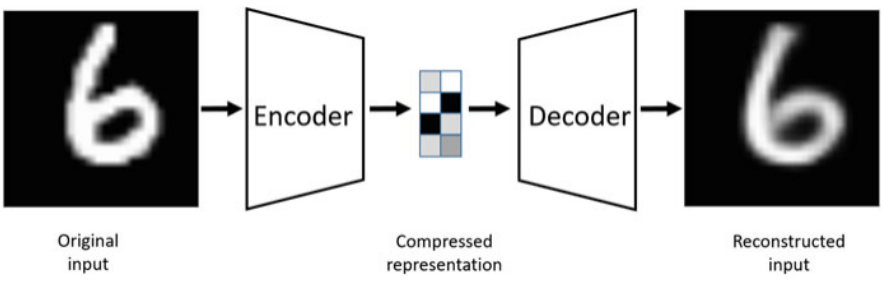

Figure 4.7: An Autoencoder example [6]

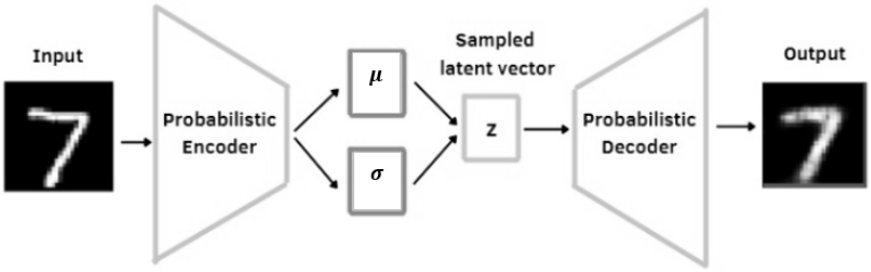

Figure 4.8: An example to illustrate the structure of VAEs

Different methods in the literature tackle this issue of AEs regularization, however, the major improvement in the capabilities of AEs is done by **Variational Autoencoders (VAEs)**.

4.5.8 Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) [25], following the concept of **Variational Bayes Inference**, are generative models that aim to represent the process of data generation through a probabilistic framework. The probabilistic nature of the VAE can be described as follows

- a **probabilistic decoder:** we assume a generative model distribution for each input data x_i conditioned on an unobserved random latent variable z_i, represented as $p_\theta(x_i|z_i)$, where θ the parameters governing the distribution,
- similarly, a **probabilistic encoder:** assume a **posterior** distribution over the latent variable given a data point x_i, denoted as $q_\phi(z_i|x_i)$, with ϕ being the parameters of this distribution,
- and finally, **latent space reparameterization:** assume a **prior** distribution for the latent variable z_i, given by $p_\theta(z_i)$, and enforce the latent representation to be Normal distribution

  $$z = \mu\epsilon + \sigma$$

  where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$

Thus, the encoder of a VAE generates parameters ($x \mapsto [\mu_x, \sigma_x]$) for a predefined distribution in the latent space for each input (see example in Figure 4.8). Once trained, VAE can be used as a generative model, we can sample random variables from the prior distribution and input them into the decoder. Since the decoder was trained to generate x from $p_\theta(x_i|z)$, it will produce meaningful new samples (See Figure 4.9 for example).

The loss function of the VAEs is as follows

$$||x - d_\phi(z)||^2 + D_{KL}(\mathcal{N}(\mu_x, \sigma_x), \mathcal{N}(0, \mathbf{I})) \tag{4.16}$$

One of the major weaknesses of VAEs becomes evident when examining the quality of reconstructed images: the resulting images are typically blurry.

4.5.9 Wasserstein Autoencoders (WAEs)

In the paper [42], the authors introduce the Wasserstein Autoencoder (WAE), a novel algorithm designed to create a generative model for data distribution. The WAE model aims to minimize a penalized form of the Wasserstein distance between the model distribution and the target distribution, resulting in a distinct regularizer compared to that used in VAEs. This regularizer encourages the encoded training distribution $Q_Z := \mathbb{E}_{P_X}[Q(Z|X)]$ to align with the prior P_Z.

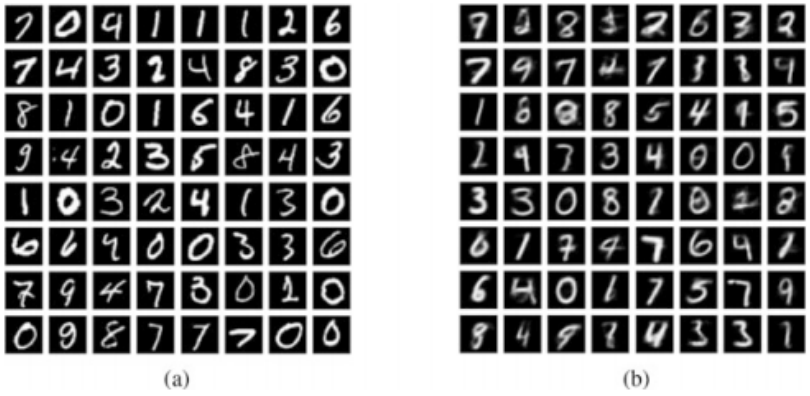

Figure 4.9: **(a)** Sample from the original MNIST dataset. **(b)** VAE generated MNIST images [6]

The experiments conducted by the authors demonstrate that WAE retains the favorable properties of VAEs, such as stable training, the utilization of an encoder-decoder setup, and the preservation of a well-regularized latent space structure. Additionally, WAE produces higher-quality samples, approaching the standard set by GANs.

The goal of the WAE is to minimize the Wasserstein distance $W_c(P_X, P_G)$ between the data distribution P_X and the latent variable model P_G, based on a novel autoencoder formulation. Let's recall Kantorovich's formulation of the optimal transport (OT) problem (Section 1.1.2):

$$W_c(P_X, P_G) := \inf_{\Gamma \in \mathcal{P}(X \sim P_X, Y \sim P_G)} \mathbb{E}_{(X,Y)\sim\Gamma}[c(X, Y)]$$

For simplicity, the authors first focus on non-random decoders, i.e., generative models $P_G(X|Z)$ that deterministically map Z to $X = G(Z)$ for a given map $G : \mathcal{Z} \to \mathcal{X}$.

Under this model, the OT problem simplifies: instead of finding a coupling Γ between two random variables (one distributed according to P_X and the other according to P_G), it is sufficient to find a conditional distribution $Q(Z|X)$ with Z marginal $Q_Z(Z) := \mathbb{E}_{X\sim P_X}[Q(Z|X)]$ matches the prior distribution P_Z.

The main theorem of the paper states:

Theorem 13 *For P_G as defined above with deterministic $P_G(X|Z)$ and any function $G : \mathcal{Z} \to \mathcal{X}$,*

$$\inf_{\Gamma \in \mathcal{P}(X \sim P_X, Y \sim P_G)} \mathbb{E}_{(X,Y)\sim\Gamma}[c(X, Y)] = \inf_{Q: Q_Z = P_Z} \mathbb{E}_{P_X} \mathbb{E}_{Q(Z|X)}[c(X, G(Z))]$$

where Q_Z is the marginal distribution of Z when $X \sim P_X$ and $Z \sim Q(Z|X)$.

The objective of the WAE is given by:

$$D_{WAE}(P_X, P_G) := \inf_{Q(Z|X)\in\mathcal{Q}} \mathbb{E}_{P_X} \mathbb{E}_{Q(Z|X)}[c(X, G(Z))] + \lambda \cdot \mathcal{D}_Z(Q_Z, P_Z)$$

where $\mathcal{Q}$ is any nonparametric set of probabilistic encoders, $\mathcal{D}_Z$ is an arbitrary divergence between Q_Z and P_Z, and $\lambda > 0$ is a hyperparameter.

The authors propose two different penalties for $\mathcal{D}_Z(Q_Z, P_Z)$:

GAN-based: $\mathcal{D}_Z(Q_Z, P_Z) = D_{JS}(Q_Z, P_Z)$, which introduces a discriminator in the latent space $\mathcal{Z}$ to separate "true" points sampled from P_Z and "fake" points sampled from Q_Z. This results in WAE-GAN. Many works in the literature [49][10][43][7] combine the adversarial training of the GANs with VAE architectures, this returns to the fact that, despite the instability of the training of GANs (as seen in Section 4.5.3), the samples generated by GANs are more compelling.

Even though WAE-GAN reverts to a min-max problem, it shifts the adversarial component from the input space $\mathcal{X}$ to the latent space $\mathcal{Z}$. Additionally, the prior P_Z can often have a simple, single-mode shape (such as a Gaussian distribution), making the task of matching distributions easier than in traditional GANs, which typically deal with unknown, complex, and potentially multi-modal distributions.

When used with Wasserstein-2 distance (the case when $c(x, y) = ||x - y||_2^2$), WAE-GAN is the same as the Adversarial Autoencoder (AAE) proposed in [27].

MMD-based: $\mathcal{D}_Z(Q_Z, P_Z) = MMD(P_Z, Q_Z)$, where MMD is the maximum mean discrepancy, referred to as WAE-MMD.

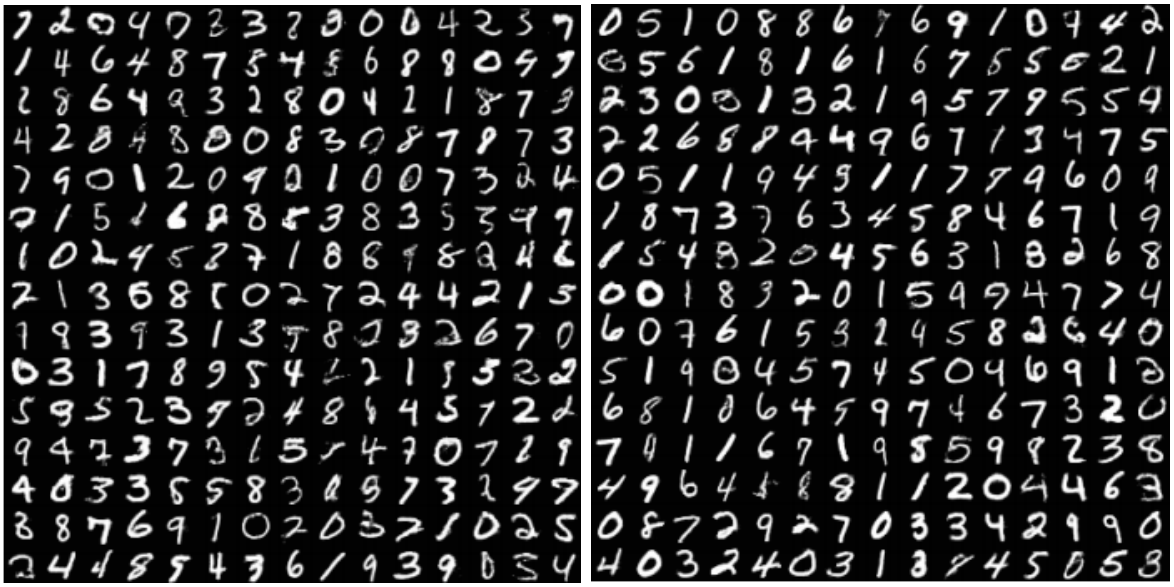

Figure 4.10: VAE (on the left), WAE-GAN(on the right) trained on MNIST dataset [42]

Our progress so far includes implementing the WAE-GAN model, as detailed in Chapter 5.

To conclude this section, [42] classical unregularized AEs focus solely on minimizing reconstruction loss. This causes different training points to be encoded into separate, non-overlapping areas randomly scattered throughout the latent space $\mathcal{Z}$, creating gaps where the decoder $P_G(X|Z)$ has not been trained. As a result, the encoder $Q(Z|X)$ does not yield a useful representation, making sampling from the latent space Z challenging.

VAEs address this by minimizing a variational bound on the KL-divergence $D_{KL}(P_X||P_G)$, which includes both the reconstruction cost and a regularizer. The regularizer measures how different the encoded distribution of each training sample is from the prior P_Z. However, this does not ensure that the overall encoded distribution $Q(Z|X)$ matches P_Z as effectively as WAE does.

4.5.10 Conditional VAE

In [38] the authors introduce Conditional Variational Autoencoders (CVAEs) as deep conditional generative models that address the challenge of structured output prediction by learning to perform probabilistic inference and generate diverse predictions. The model learns to represent the distribution of high-dimensional output space as a generative model conditioned on input observations, using three key variables: input x, output y, and latent z. CVAE functions as a conditional directed graphical model where input observations modulate the prior on Gaussian latent variables that generate the outputs, with training performed through maximizing the conditional log-likelihood using Stochastic Gradient Variational Bayes (SGVB) framework.

The Conditional generative process: for a given observation x, z that is drawn from a prior distribution $p_\theta(z|x)$ and the output y is generated from the distribution $p_\theta(y|x, z)$.

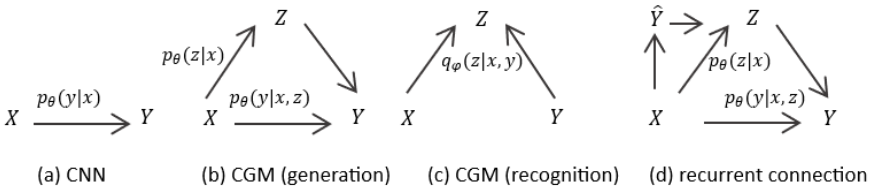

Figure 4.11

4.5.11 Choose the right Hyperparameters for a VAE

Hyperparameters to consider when tuning a VAE [12] include the latent dimension, β coefficient, and the network architecture in general as in section 4.4.3.

The *latent dimension* represents the size of the vector that encodes the compressed version of the input data. This parameter is crucial because it determines how much information is captured during encoding, influencing the diversity of the generated samples. A larger latent dimension allows the model to capture more details, but it can also lead to overfitting and instability. On the other hand, a smaller latent dimension forces the VAE to learn more efficiently by focusing on essential features, but it may limit the expressiveness and quality of the output. The goal when tuning the latent dimension is to balance the complexity of the model with its ability to capture the essential features of the data without overfitting. As with all hyperparameter tuning, this involves finding the right trade-off between complexity and performance.

When using a regularized version of VAE, such as β-VAE, the β *coefficient* becomes another important parameter to tune. The β coefficient balances the trade-off between the reconstruction loss and the Kullback-Leibler (KL) divergence loss in the loss function (see Equation 4.17). A higher β value encourages a more regularized latent space, potentially leading to better interpretability, but it can degrade the quality and diversity of the reconstructed outputs. Conversely, a lower β coefficient allows for better reconstruction fidelity and richer outputs but may result in the latent space collapsing, where the model ignores the prior distribution.

$$||x - d_\phi(z)||^2 + \beta \cdot D_{KL}(w\mathcal{N}(\mu_x, \sigma_x), \mathcal{N}(0, \mathbf{I})) \tag{4.17}$$

Lastly, the *network architecture* plays a significant role in determining the complexity and flexibility of the VAE, similar to the considerations discussed in section 4.4.3 for neural network tuning. The architecture dictates how well the VAE can handle various data types and complexities. Choosing the optimal VAE architecture involves carefully considering the characteristics and dimensions of the training data. Exploring different architectures, such as convolutional layers (CNNs) for image data, can be highly beneficial in enhancing the model's ability to learn complex patterns. Therefore, testing various architectures and configurations is crucial for optimizing the performance of a VAE.

Chapter 5

Data Generation for Ensemble Construction

After establishing the foundational background and tools in the preceding chapters, we now arrive at the main chapter of the thesis. We first show the general framework of the kind of problems that we are interested. We show the general set up for implementation and then we show a range of experimental results, covering different architectures and configurations of the deep learning models that were discussed earlier in chapter 4.

5.1 General Framework

We consider parametric PDEs where certain parameters are modeled as random variables following known probability distributions. Formally, the problem can be stated as

$$\begin{cases} \mathcal{L}(u;p) = f & \text{in } \Omega \\ \mathcal{B}(u;p) = g & \text{on } \partial\Omega, \end{cases} \tag{5.1}$$

where $\Omega \subset \mathbb{R}^d$ is an open and bounded set with a sufficiently regular boundary, and $\mathcal{P} \subset \mathbb{R}^P$ is a compact parameter set. Furthermore, $\mathcal{L}$ is a differential operator, $\mathcal{B}$ represents boundary conditions, $p \in \mathcal{P}$ is the parameter that follows a probability distribution $\mathcal{D}$, f represents source terms, and g represents boundary data. The goal is to determine the unknown solution $u : \Omega \times \mathcal{P} \to \mathbb{R}^d$, or simply some statistics, for example the mean

$$\mathbb{E}(u)(x) := \int_{\mathcal{P}} u(x,p) d\mathcal{D}(p).$$

Unlike stochastic PDEs, the equation remains deterministic for each fixed parameter value, but the randomness in parameters induces uncertainty in the solution.

When parameters are random, each solution instance corresponds to a single parameter realization. However, in practice, understanding the overall behavior and statistics of solutions across the parameter space is essential. This naturally leads to uncertainty quantification (UQ) methods, which are discussed in detail in Chapter 3.

Monte Carlo (MC) methods provide a straightforward approach to estimate solution statistics by sampling parameters and averaging corresponding solutions. However, traditional MC approaches require solving the full PDE for each parameter sample, making them computationally expensive for large sample sizes (as discussed in Chapter 3).

Our primary objective is to ensure that the mean of the generated solutions closely approximates the reference mean. This allows for some flexibility in the accuracy of individual solutions within the MC ensemble, as long as the overall mean is enhanced.

The key contribution of this work is demonstrating that Variational Autoencoders (VAEs) can efficiently learn this mean of the solution mapping from a few snapshots (i.e., solutions with given

parameter samples). Once trained, the VAE enables rapid generation of approximate solutions, substantially improving the efficiency of Monte Carlo estimation.

Our main approach focuses on enhancing the VAE model by selecting suitable architectures and performing hyperparameter tuning. For this purpose, we employ the Bayesian Optimization (BO) technique, which is detailed in Chapter 4.

To construct the MC ensembles, we explore various VAE models presented in Chapter 4. In this chapter, we present the results of implementing β-VAE (using both feedforward and convolutional neural network structures) and WAE-VAE.

General architecture of the generative models

Here, we provide details on the general architecture applied to the generative models, with more specific details depending on the case test and the results shown.

We remark that in the following description some hyperparameters are specified in a range, rather than with single values: This means that the corresponding exact value will be determined by an hyperparameter tuning procedure.

Encoder: The encoder begins with an input layer containing d neurons, followed by a ReLU activation function (section 4.2). It then has a hidden layer with the number of neurons in the range of $[64, 512]$, also followed by a ReLU activation layer. A dropout layer (see remark 7) with a dropout rate between $0.2 - 0.3$ is added next. A second hidden layer with the number of neurons in $[64, 512]$ is then added, followed by another ReLU activation layer. Finally, the latent space layer with a latent dimension in $[16, 32]$ is included.

Decoder: The decoder starts with an input layer that has a number of neurons equal to the latent dimension, followed by a ReLU activation. It then has a first hidden layer with $[64, 512]$ neurons, followed by a ReLU activation function. A dropout layer with a $0.2 - 0.3$ dropout rate is added, followed by a second hidden layer with $[64, 512]$ neurons and a ReLU activation layer. An output layer with d neurons is then included. The choice of the last activation function for the output depends on the problem being studied. For example, if the solution is in $(0, 1)$, a sigmoid activation function is used.

In the case of the WAE-GAN, an additional neural network is required to represent the discriminator that works in the latent space (Chapter 4, Section 5.3):

Discriminator: The discriminator has the same structure as the decoder, with the only difference being that the output layer has a dimension of 1 and is followed by a sigmoid activation function.

The values of the hyperparameters are then chosen by Bayesian optimization, we give details for each example later.

Dataset preparation

We prepare the training set with high-fidelity solutions of the parametric PDE. For that we use FEniCS [9], which is a computation platform for solving PDEs with the finite element method.

In more detail, we use the following procedure.

To construct a single dataset, we fix a number $N \in \mathbb{N}$ of samples and a repetition index $j = 1, \ldots, n_{sam}$. This second index is used to account for the effect of randomization, as will be explained in the following. We then perform the following operations:

1. **Parameter Space Sampling:** We sample N parameters $\mathcal{P}_N := \{p_1, p_2, ..., p_N\}$ from $\mathcal{X}$ according to the given probability distribution $\mathcal{D}$. We remark that the n parameters are sampled independently.

2. **High-Fidelity Solution Generation:** We compute the correspoding solution of the PDE (5.1). More specifically, we first derive a corresponding weak formulation, and consider the corresponding Finite Element discretizatio implemented in FEniCS. Since the PDE is deterministic once the value of the parameter is fixed, we can exectute the solver once for each parameter $p_i \in \mathcal{P}_N^j$, $i = 1, \ldots, N$, obtaining a corresponding FEM solution $u_i \in \mathbb{R}^d$, where $d \in \mathbb{N}$ is the number

of degrees of freedom of the FEM solution, depending on the mesh that has been used for the single problem. These values are collected in a set

$$\mathcal{U}_N^j := \{u_1, \dots, u_N\}. \tag{5.2}$$

3. **Dataset Construction:** Using these parameter samples and corresponding solution (vectors), we define a dataset with input set $\mathcal{P}_N^j$ and an output set $\mathcal{U}_N^j$.

We then assemble different datasets for training incrementally accurate models and for testing them. More specifically, we proceed as follows:

1. **Reference dataset:** A testing dataset $(\mathcal{P}_{ref}, \mathcal{U}_{ref})$ with size $N = 10000$ is generated for testing purposes. Since the exact solution of the PDE (5.1) is in general not available, this is considered to be the reference solution. Further details on the testing of the model are provided in the next sections.

2. **Training datasets:**

   (a) **Construction:** We construct training datasets $\left(\mathcal{P}_N^j, \mathcal{U}_N^j\right)$ with incremental values $N \in \{100, 200, \dots, 1000\}$. We remark that these datasets are nested, meaning that for each $j = 1, \dots, n_{sam}$, the parameter samples are chosen as nested sets

   $$\mathcal{P}_{100}^j \subset \mathcal{P}_{200}^j \subset \cdots \subset \mathcal{P}_{1000}^j,$$

   where each dataset is obtained incrementally from the previous one by adding 100 new samples.

   (b) **Training and validation splitting:** For each dataset $\left(\mathcal{P}_N^j, \mathcal{U}_N^j\right)$ we further consider a standard training-validation splitting. Namely, each dataset is partitioned into a training and a validation subset. The training set comprises $80\% - 95\%$ of the total data, with the remaining portion reserved for validation. In this case, since the hyperparameters have already been optimized via BO, we use the validation set only to observe the decay of the loss and verify the convergence of the method. This split ensures robust model evaluation while maintaining sufficient training data.

Construction of the approximated MC solutions

The generative models and the corresponding approximations of the MC mean are constructed as follows: For each $j = 1, \dots, n_{sam}$, and for each value of $N = 100, \dots, 1000$ and $j = 1, \dots, n_{sam}$, a single generative model M_N^j is trained. This training follows the architecture specifications and hyperparameter definitions and optimizations that are specific to each problem. Each model represents a map $M_N^j : \mathbb{R}^{latent} \to \mathbb{R}^d$, where we recall that $\mathbb{R}^{latent}$ is the space of latent parameters of the generative model.

We would like to remark that we tested also an incremental training stategy. Namely, for each j, the weights obtained at the end of the training of the model M_{100}^j are used as initial weights for training the model M_{200}^j, and the same for all other values on N. This initialization procedure, however, did not produce significantly advantages, while impeding a parallel training of the different models. We did thus not use this method.

We can then use these trained models for computing approximations of the MC mean. We define a number $L \in \mathbb{N}$ of samples and a number $k = 1, \dots, n_{rep}$ of repetitions of the generation.

For each $k = 1, \dots, n_{rep}$, we then sample L independent values $Z_L^k := \{z_1, \dots, z_L\} \in \mathbb{R}^{latent}$ from the latent space, and evaluate the model M_N^j on Z_L^k to obtain a set of generated solutions

$$\hat{\mathcal{U}}_{N,k}^j := \{M_N^j(z_i) : z_i \in Z_L^k\} \subset \mathbb{R}^d.$$

If the generative model is sufficiently accurate, we expect each vector $M_N^j(z_i)$ to resemble an actual (FEM) solution of (5.1). However, we remark that this needs not to be the case: First, there is no explicit dependence on a given parameter $p \in \mathcal{P}$; Second, the model is trained to generate vectors that collectively resemble the empirical distribution of the training dataset, and not necessarily to match single instances of the training set.

For each of these model-generation pair, the approximated MC mean (see (3.3)) is then defined as the empirical mean of all vectors in $\hat{\mathcal{U}}_{N,k}^j$, i.e.,

$$\hat{E}_{N,j,k} = \frac{1}{\left|\hat{\mathcal{U}}_{N,k}^j\right|} \sum_{\hat{u} \in \hat{\mathcal{U}}_{N,k}^j} \hat{u} \in \mathbb{R}^d.$$

We stress that this quantity is averaged over the samples, not over space: It is in fact a d-diemensional vector. Moreover, this procedure gives a value for each dataset size $N = 100, \dots, 1000$, each different realization index $j = 1, \dots, n_{sam}$ of the dataset, and each repetition $k = 1, \dots, n_{rep}$ of the generation process. These two randomization processes (over the dataset and over the generation) ensures a fairer comparison of the different models.

Evaluation of the models

To evaluate the performance of our methodology, we establish the following validation framework.

First, we construct the MC approximations obtained with the FEniCS solutions. Namely, for each dataset $\mathcal{U}_N^j$ (see (5.2)) we define the MC approximation

$$E_{N,j} = \frac{1}{\left|\mathcal{U}_N^j\right|} \sum_{u \in \mathcal{U}_N^j} u \in \mathbb{R}^d.$$

Morever, for the reference dataset $\mathcal{U}_{ref}$ we define the reference MC mean as

$$E_{ref} = \frac{1}{|\mathcal{U}_{ref}|} \sum_{u \in \mathcal{U}_{ref}} u \in \mathbb{R}^d.$$

As mentioned above, this empirical mean serves as the high-fidelity approximation of the true mean, that is generally not available.

These approximate MC values $E_{N,j}$ (depending on the finite dataset) and $\hat{E}_{N,j,k}$ (depending on the dataset and the generative models), and the almost exact MC value E_{ref} are used for assessing the qualities of the generative models.

Namely, using the normalized 2-norm over $\mathbb{R}^d$ we define the error between the reference soltution and those based on the finite datasets as

$$e_{N,j} := \frac{1}{\sqrt{d}} \left\| E_{ref} - \hat{E}_{N,k,j} \right\|_2,$$

and the error between the reference solution and those computed by the generative models as

$$\hat{e}_{N,j,k} := \frac{1}{\sqrt{d}} \left\| E_{ref} - \hat{E}_{N,k,j} \right\|_2.$$

This last family of errors $\hat{e}_{N,j,k}$ can be further aggregated over $k = 1, \dots, n_{rep}$, obtaining an estimation of the mean and variance over the different generations.

Training Configuration

Loss Function Components: In Chapter 4, VAEs and their loss functions are defined, which fundamentally consist of two parts: the *reconstruction loss* and the *regularization loss*. In our implementation of the β-VAE, since we do not need to be accurate for each individual solution, we use the

Sum of Squared Mean Differences (SSMD) for the reconstruction loss which is defined by: given the solution data u_i and the reconstructed solutions $\hat{u}_i$

$$SSMD(u, \hat{u}) = \sum_{j=1}^{d} \left(\frac{1}{n} \sum_{i=1}^{n} (u_{ij} - \hat{u}_{ij}) \right)^2$$

where d is the dimension of u and n is the batch size.

For the regularization component, we use the Kullback-Leibler divergence weighted by the parameter β:

$$SSMD(u, \hat{u}) + \beta \cdot KL(P_q || P_z) \tag{5.3}$$

For the WAE-GAN model, the loss function is described in Section 5.3, where the reconstruction loss is Wasserstein distance where the cost function $c(x, y) = ||x - y||^2$ and the Jensen–Shannon divergence is used for the regularization term.

Optimizer Setup: For all the results we show we use Adam optimizer with learning rate chosen by Bayesian optimization and with weight decay 10^{-5}.

Hyperparameter Tuning: The key hyperparameters tuned include the number of neurons in the hidden layers, the latent space dimension, the dropout rate, the learning rate, the batch size for training, and the regularization weight β (for β-VAE). We run Bayesian optimization (see Section4.4.2) to choose the optimal choice of hyperparameter combination. In here we will show the results of different combinations that yield from multiple running of BO.

5.2 Test case: Parametric diffusion equation

In this section we define the test case that will be used to test our method. We define a mathematical formulation of the problem, and the details for its discretization.

Definition of the continuous problem

We test our scheme on a two-dimensional diffusion equation with variable diffusion coefficients. Diffusion equations are used to describe the distribution of a quantity, such as heat, particles, or chemicals, within a physical system over time. It is given by the PDE

$$\begin{aligned} \operatorname{div}(D(x)\nabla u(x)) = 0 \in \Omega = [0,1] \times [0,1] \\ u(\mathbf{x}) = 1, \ \mathbf{x} = (0, y) \\ u(\mathbf{x}) = 0, \ \mathbf{x} = (1, y) \\ (\nabla u(x))^T n = 0, \ \mathbf{x} = (x, 0) \ \& \ \mathbf{x} = (x, 1) \end{aligned}$$

with $u(x)$ representing the quantity being diffused, such as temperature or concentration, and $D(x)$ being the diffusion coefficient. For the purpose of initial testing of the methods, we consider the piecewise constant diffusion coefficient $D : \Omega \to \mathbb{R}$ defined by

$$D(x) = \begin{cases} D_1, & x \in \Omega_1 \\ D_2, & x \in \Omega_2 \\ D_3, & x \in \Omega_3 \\ D_4, & x \in \Omega_4 \end{cases}$$

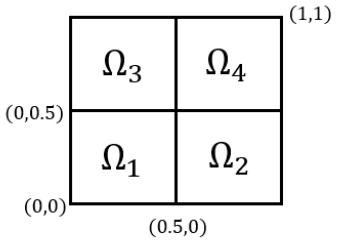

Figure 5.1: Subdomains of Ω

where $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ are subdomains of $\Omega = [0,1] \times [0,1]$ (as shown in Figure 5.1).

The parameter space for this test case is $\mathcal{P} = [10^{-3}, 1]^4 \subset \mathbb{R}^4$, $p = (D_1, D_2, D_3, D_4)^T \in \mathcal{P}$, equipped with the log-uniform distribution $\mathcal{D}$. We point out that further testing should be applied to study the limiting behavior of the problem when the domain partition is refined, corresponding to an increase in the dimension of the parameter space.

As discussed before, for a given diffusion coefficient p the equation is deterministic, and we compute the corresponding solution of the variational formulation, which is derived in a standard way (see e.g. [11]).

FEniCS solutions

Once the variational form of the diffusion equation is established, we proceed to its numerical solution using FEniCS. We begin by defining a unit square mesh with 21×21 elements.

The function space for the solution and test functions is chosen to be first-order Lagrange polynomials ("P1" elements). Next, the boundary conditions, as specified in the test case, are implemented through DirichletBC in FEniCS, ensuring the solution satisfies the prescribed values on the domain boundaries. For each value of a sampled parameter, we solve the variational form of the equation using the FEniCS solver (more details on FEniCS can be found in [9]).

Dataset and generation specifications

We generate the dataset following the specifications of Section 5.1. More precisely, we use the values of the paramters defined in Table 5.1.

	n_{sam}	L	n_{rep}
Value	4	1000	10

Table 5.1: Dataset and generation parameters.

Choice of Hyperparameters and Training

We run Bayesian Optimization with the hyperparameters ranges defined in Table 5.2 (where the parameter β is used only in β-VAE). We recall that integer parameters are treated as continuous ones,

	N. layer 1	N. layer 2	$\log_{10}$(lr)	batch size	latent dim	β
Range	(64, 512)	(64, 512)	(-4, -2)	(32, 128)	(16, 64)	(50, 500)

Table 5.2: Range of search for the Bayesian Optimization of the hyperparameters.

and rounded at each evaluation of the model (see Section 4.4.2). In the following, we will thus just report these integer results.

The error measure that is minimized by BO is the validation loss. We choose to use the dataset with $N = 700$ as a validation set.

5.3 Results

We present now the results obtained with two different architectures: β-VAE (Section 4.5.11) and WAE (Section).

5.3.1 β-VAE

Running the BO as described in Section 5.2 gives the following optimal combination of the hyperparameters (denote HP_{VAE} in the following). Real values are rounded to two significant digits.

ID	N. layer 1	N. layer 2	$\log_{10}$(lr)	batch size	latent dim	β
HP_{VAE}	266	348	-2.14	39	60	82.89

Table 5.3: Optimal hyperparameters for β-VAE.

To summarize the training procedure, we report in figure 5.2 the evolution of the training and validation losses, and their components, when trained for 1500 epochs. To simplify the visualization we report the results for the representative datasets sizes $N = 100, 500, 1000$, and for the four repetitions (i.e., $j = 1, 2, 3, 4$ and $n_{sam} = 4$). For all panels it is immediate to see that the train loss is dominated by the MSE components, while the KL component is more efficiently reduced. Moreover, the total train and validation losses have similar final values, indicating that no overfitting or underfitting is occurring. Finally, an increase in the dataset size N leads to a quicker convergence, and the loss remain stable after reaching its plateu value, indicating that the value of the learning rate is appropriately chosen. Considering all these observations, we may assess that the training of these models is effective across all dataset sizes and repetitions. We remark moreover that the plot corresponding to omitted values of N show completely analogous results.

We can now employ these models in the data generation and MC computation pipeline described in the previous sections. The results are reported in Figure 5.3. In this case, each row corresponds to a different parameter sampling repetition (i.e., to $j = 1, \dots, 4$), while in each plot the value N plotted along the x-axis. For each of these values of N, the blue and orange curves report the errors $E_{N,j}$ (corresponding the Fenics solution with finite dataset) and $\hat{E}_{N,j,k}$ (corresponding to the generative solution). For the Fenics errors, the four values $E_{N,j}$, $j = 1, \dots, 4$, are agggregated, and all the four panels report the same curves (showing a mean and standard deviation of the error). For the error of the generative models, instead, the value of $\hat{E}_{N,j,k}$ is aggregated across $k = 1, \dots, n_{rep} = 10$, and the corresponding mean and standard deviations are reported in the figure.

Figure 5.3 clearly show that both method present a convergence of order $1/\sqrt{N}$ to the reference solution. This is theoretically expected for the MC solution generated with exact PDE solves, but it is a significant achievement for the generated solutions. In particular, this shows the (numerical) convergence of the method: Datasets of increased size tend to produce more accurate results. Even more, the rate of decay is of the order expected from a MC approximant.

However, we can still clearly observe a non-monotonic decay of the error. This is a potential limitation of the method, since there is no guarantee that a larger dataset will reduce the error. Moreover, the results vary significantly across the four differt repetitions. First, this is an indication of a lack of robustness of the method. We expect that mitigating these two limitations will require to develop a more robust process for the tuning of the hyperparameters, and possibly to gain additional insights into the inner design of these VAEs.

Nevertheless, there are single values instances (notably, the case $j = 1$ in the top left panel) for which β-VAE clearly outperforms the standard MC solution, demonstrating the effectiveness of the method. This is a promising indication for the possibility of further developments.

5.3.2 WAE-GAN

Running the BO as described in Section 5.2 gives this time the following optimal combination of the hyperparameters (denote HP_{WAE} in the following). Real values are rounded to two significant digits.

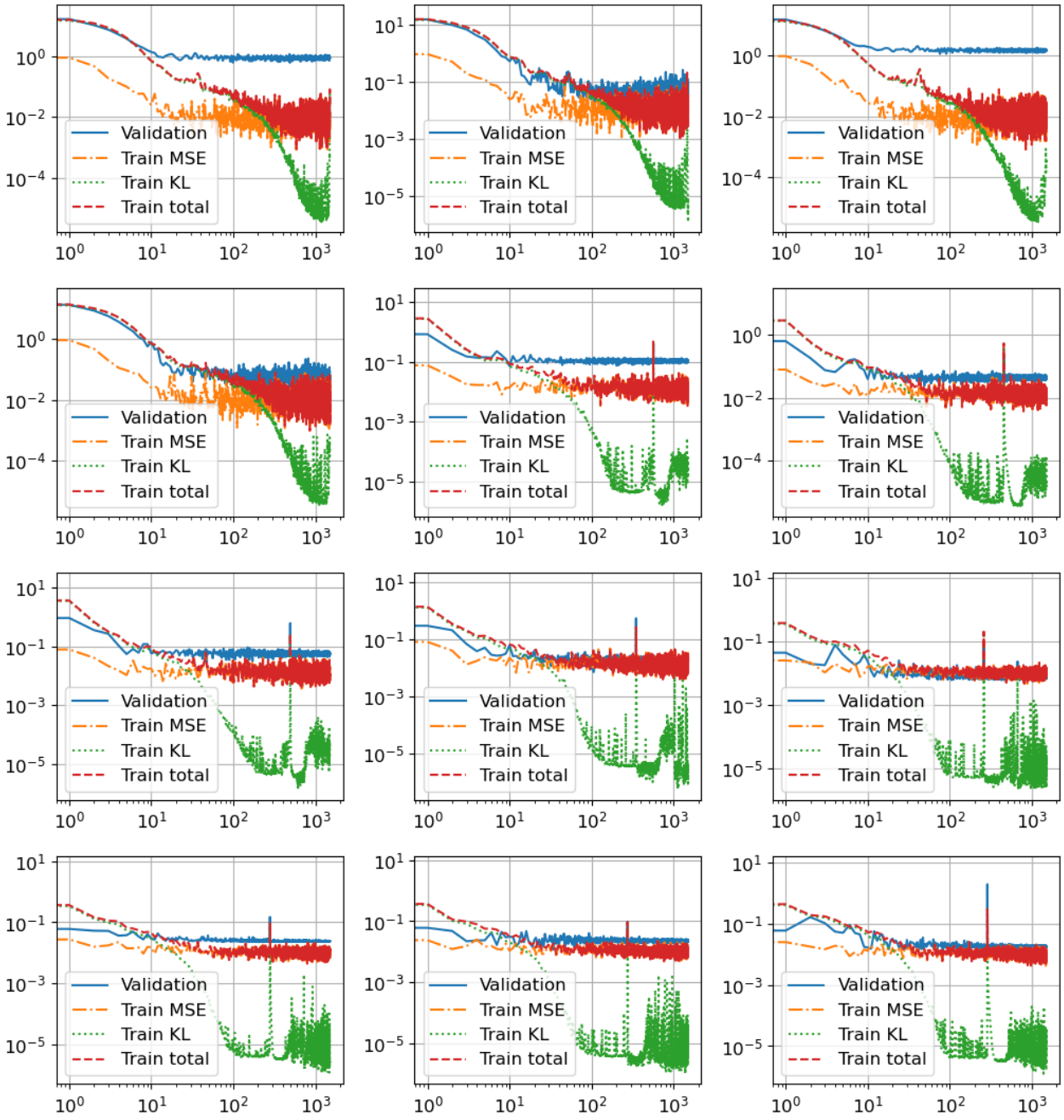

Figure 5.2: Loss evolution during training for VAE. The figure visualizes the average train and validation losses, as a function of the training epoch. The training loss is further decomposed into its RMSE add KL components. Each panel corresponds to a different training set: The columns correspond to dataset sizes $N = 100, 500, 1000$, while the columns correspond to different repetitions.

ID	N. layer 1	N. layer 2	$\log_{10}$(lr)	batch size	latent dim
HP_{WAE}	128	128	-4	34	19

Table 5.4: Optimal hyperparameters for WAE.

The training in this case was not effective, failing to reach convergence even just on the training set. To demonstrate this fact, we show in Figure 5.4 the behavior of the two components of the training loss for the first 500 epochs. The training has been stopped without reaching 1500 epochs since no convergence was observed. It is clear that the two components are diverging.

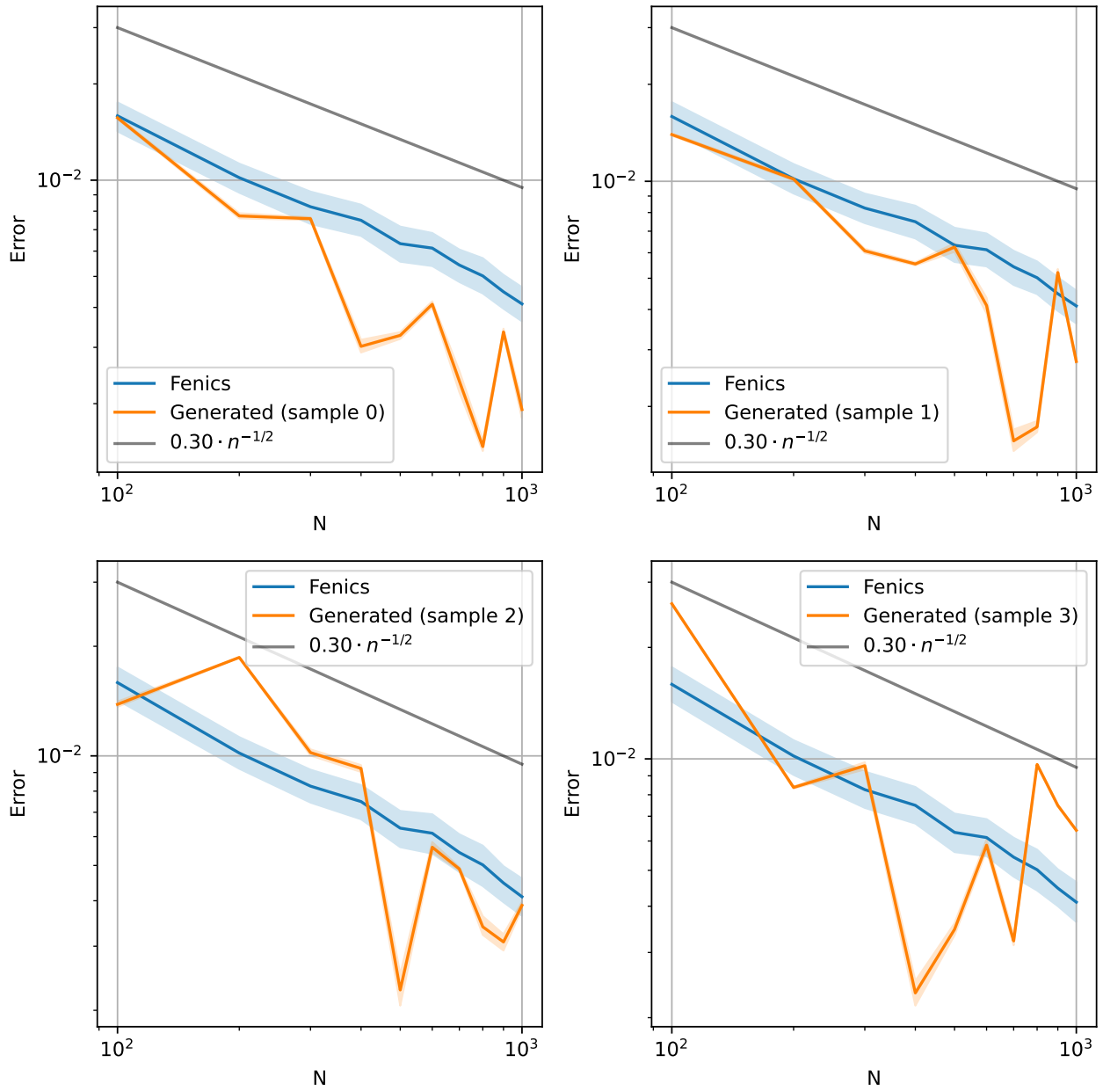

Figure 5.3: Error with respect to the reference solution as a function of the number of samples in the training set. Each figure reports the error between the Fenics solution and the reference solution (blue lines), and between the generated solution and the reference solution (orange lines). The four panels correspond to four different repetitions (i.e., different samples of parameters). For each setting, the lines report the errors as means (solid lines) and standard deviations (shaded area) over the repetitions. The gray lines represent the theoretical expected decay of the MC error (scaled with a suitable constant to improve the visualization).

To further verify that no convergence is achieved, we used the resulting models in the genration pipeline. The corresponding results are reported in Figure 5.5. We see here a clear divergence of the generative MC mean from the reference solution. In this case, the hardness of training of the generative model poses a significant limitation to its use.

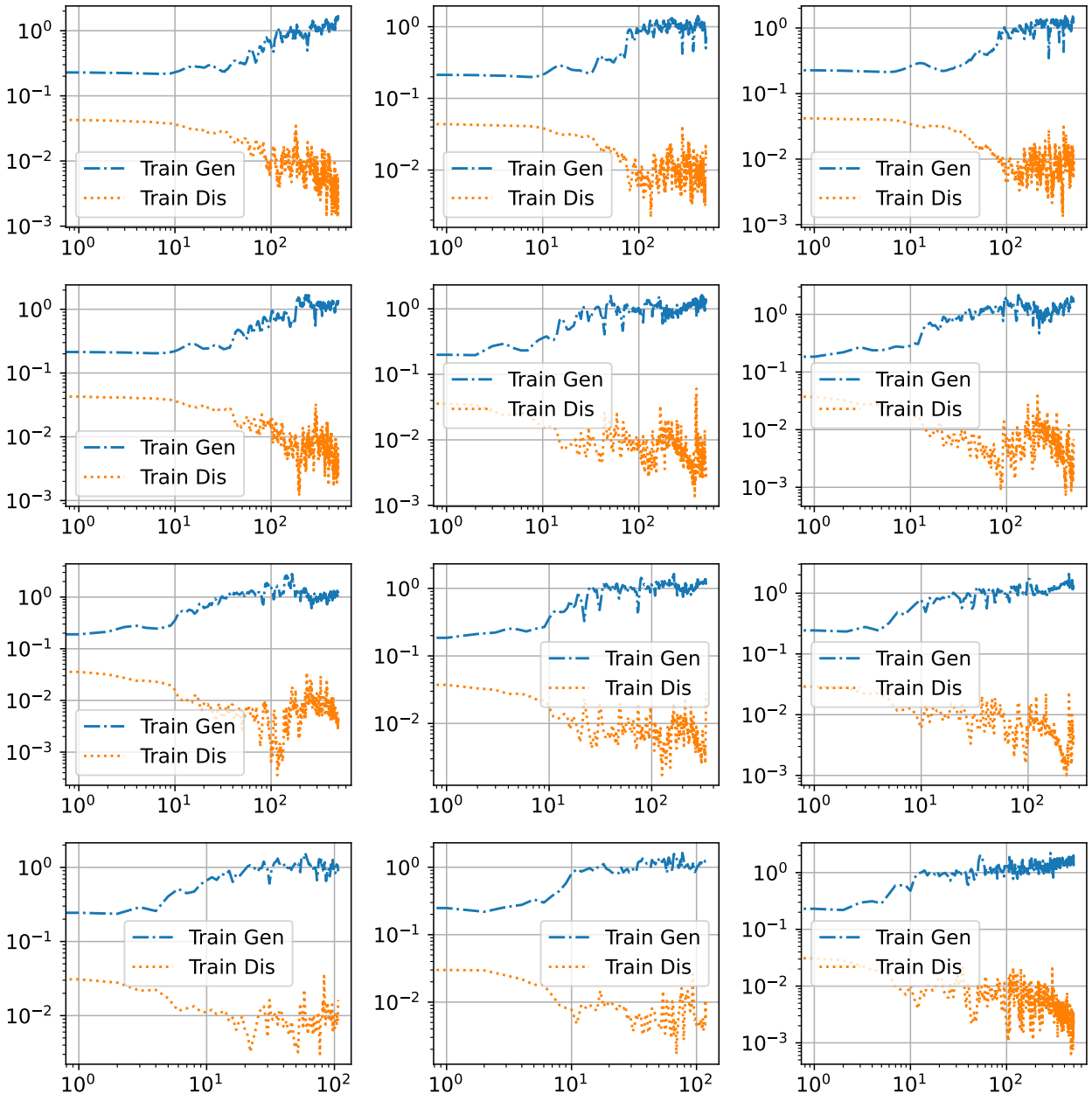

Figure 5.4: Loss evolution during training for WAE. The figure visualizes the average train losses, as a function of the training epoch. The training loss is further decomposed into its generative and discriminative components. Each panel corresponds to a different training set: The columns correspond to dataset sizes $N = 100, 500, 1000$, while the columns correspond to different repetitions.

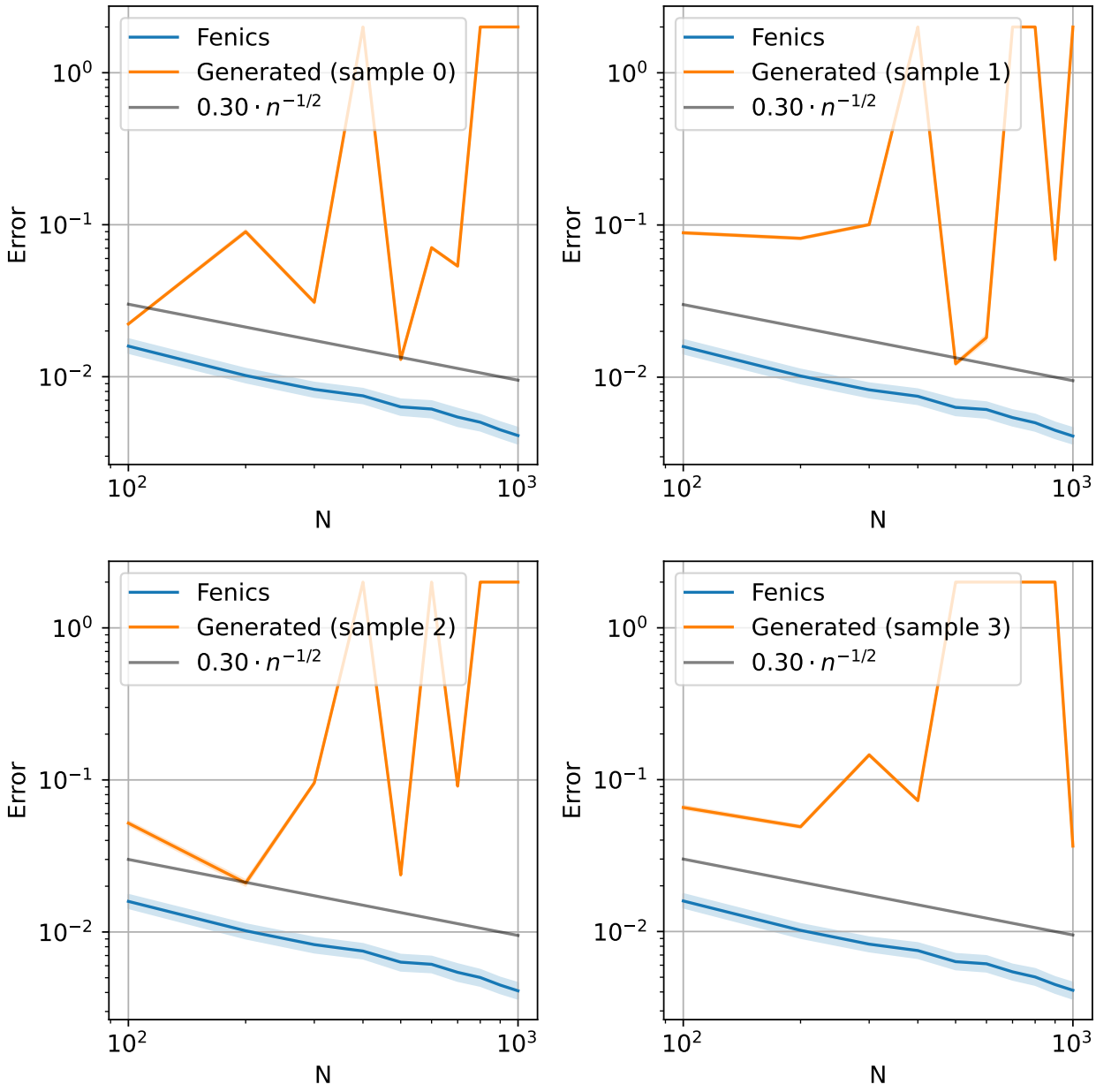

Figure 5.5: Error with respect to the reference solution as a function of the number of samples in the training set. Each figure reports the error between the Fenics solution and the reference solution (orange lines), and between the generated solution and the reference solution (blue lines). The four rows correspond to four different repetitions (i.e., different samples of parameters). For each setting, the lines report the errors as means (solid lines) and standard deviations (shaded area) over the N_{rep} repetitions. The gray lines represent the theoretical expected decay of the MC error (scaled with a suitable constant to improve the visualization).

Chapter 6

Conclusions and future work

In this thesis, we explored the potential of using generative models, specifically Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), to improve the computational efficiency of uncertainty quantification (UQ) in parametric scalar partial differential equations (PDEs). The primary motivation for this work was to address the high computational cost associated with traditional Monte Carlo simulations for uncertainty propagation in complex models, particularly when high-dimensional and computationally expensive problems are involved. By focusing on generating Monte Carlo ensembles with accurately matched empirical moments, our approach sought to provide an alternative to the exhaustive model evaluations required by conventional Monte Carlo methods.

The results presented in Chapter 5 demonstrated that the integration of VAEs into uncertainty quantification tasks shows promising potential. Preliminary experiments with β-VAE and WAE-GAN revealed that generative models could be trained to approximate the desired distributions of PDE solutions with significant computational savings compared to traditional Monte Carlo simulations. However, while the approach is promising, it is clear that further work is required to refine and improve the method.

There are several avenues for future research that can further enhance the findings of this thesis. First, the approximation method presented in Chapter 2, which aims to compute the Wasserstein-1 distance more efficiently, still requires further refinement. The convergence of the spectral solution method has not yet been achieved, and additional work is necessary to ensure the accuracy and stability of the solution. When successfully developed, this method would likely offer possibly significant advantages in the training of Wasserstein generative models. Furthermore, the methods for generating Monte Carlo ensembles have so far only been applied to relatively simple test problems. For more complex and higher-dimensional problems, it will be crucial to scale the methods effectively while ensuring that they retain their ability to produce reliable approximations.

Another area for future work is the training of the generative models. Although VAEs and GANs can efficiently generate data, the complexity and extensive training required, even for simpler test problems, highlight the need for improved training protocols. These protocols could focus on better controlling the convergence of the models and reducing the computational burden associated with their training.

In summary, this thesis presents a novel approach to uncertainty quantification that integrates generative models with the Wasserstein distance, offering a potential route to more efficient and accurate uncertainty propagation in the context of parametric PDEs. While the current results show promise, it is clear that further advancements in the computational methods, model training, and application to more challenging problems are needed to fully realize the potential of this approach. Nonetheless, the work presented here serves as a foundation for future research that could significantly enhance the efficiency and accuracy of uncertainty quantification in a broad range of scientific and engineering disciplines.

Bibliography

[1] L. Ambrosio, K. Deckelnick, G. Dziuk, M. Mimura, V. A. Solonnikov, H. M. Soner, and L. Ambrosio. *Lecture Notes on Optimal Transport Problems*. Springer, 2003.

[2] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations*, 2017.

[3] M. Arjovsky and L. Bottou. Towards Principled Methods for Training Generative Adversarial Networks. In *International Conference on Learning Representations*, 2017.

[4] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein Generative Adversarial Networks. *ICML*, 2017.

[5] P. Baldi. Autoencoders, Unsupervised Learning, and Deep Architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49. JMLR Workshop and Conference Proceedings, 2012.

[6] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[7] D. Berthelot, T. Schumm, and L. Metz. BEGAN: Boundary Equilibrium Generative Adversarial Networks. *CoRR*, 2017.

[8] A. Brock, J. Donahue, and K. Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis, 2018.

[9] Documentation. FEniCS Project. `https://fenicsproject.org/`.

[10] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville. Adversarially learned inference. In *International Conference on Learning Representations*, 2017.

[11] A. Ern and J.-L. Guermond. *Theory and practice of finite elements*, volume 159 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2004.

[12] R. Etukuru, A. Kumar, and M. Ismail. How can you choose the right Hyperparameters for a Variational Autoencoder?

[13] L. C. Evans. Partial Differential Equations and Monge-Kantorovich Mass Transfer. *Current developments in mathematics*, 1997(1):65–126, 1997.

[14] L. C. Evans and W. Gangbo. *Differential Equations Methods for the Monge-Kantorovich Mass Transfer Problem*. American Mathematical Soc., 1999.

[15] E. Facca. *Biologically Inspired Formularion of Optimal Transport Problems*. Phd thesis, Università degli Studi di Padova, Dip. Matematica "Tullio Levi Civita", 2018.

[16] E. Facca, F. Cardin, and Putti, Mario. Towards a Stationary Monge–Kantorovich Dynamics: The Physarum Polycephalum Experience. *SIAM J. Appl. Math.*, 78(2):651–676, 2018.

[17] E. Facca, S. Daneri, F. Cardin, and M. Putti. Numerical Solution of Monge–Kantorovich Equations via a Dynamic Formulation. *Journal of Scientific Computing*, 82(3):1–26, 2020.

[18] E. Facca, S. Daneri, F. Cardin, and Putti, Mario. Numerical Solution of Monge-Kantorovich Equations via a dynamic formulation . *J Sci Comput*, Accepted for publication, 2020.

[19] E. Facca and F. Piazzon. Transport energy. *arXiv preprint arXiv:1909.04417*, 2019.

[20] M. Feldman and R. J. McCann. Uniqueness and Transport Density in Monge's Mass Transportation Problem. *Calculus of Variations and Partial Differential Equations*, 15(1):81–113, 2002.

[21] I. Goodfellow. Deep Learning, 2016.

[22] I. J. 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*, 2014.

[23] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved Training of Wasserstein Gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 5769–5779, USA, 2017. Curran Associates Inc.

[24] IBM. What is a neural network?

[25] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *ICLR*, 2014.

[26] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2016.

[27] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial Autoencoders. *ArXiv*, abs/1511.05644, 2015.

[28] A. Mallasto, G. Montúfar, and A. Gerolin. How Well Do WGANs Estimate the Wasserstein Metric?, 2020.

[29] S. Pandian. A Comprehensive Guide on Hyperparameter Tuning and its Techniques, 2024. Analytics Vidhya.

[30] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. Context encoders: Feature learning by inpainting, 2016.

[31] G. Perarnau, J. van de Weijer, B. Raducanu, and J. M. Álvarez. Invertible conditional gans for image editing, 2016.

[32] E. Plaut. From principal subspaces to principal components with linear autoencoders. *arXiv preprint arXiv:1804.10253*, 2018.

[33] Rendyk. Tuning the Hyperparameters and Layers of Neural Network Deep Learning, 2024. Analytics Vidhya.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation, Parallel Distributed Processing, Explorations in the Microstructure of Cognition, ed. De Rumelhart and j. Mcclelland. Vol. 1. 1986. *Biometrika*, 71:599–607, 1986.

[35] F. Santambrogio. Absolute Continuity and Smmability of Transport Densities: Simpler Proofs and New Estimates. *Calculus of variations and partial differential equations*, 36:343–354, 2009.

[36] F. Santambrogio. *Optimal Transport for Applied Mathematicians*, volume 87 of *Calculus of Variations, PDEs, and Modeling*. Birkhäuser, Basel, Oct. 2015.

[37] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press, 2014.

[38] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[39] T. J. Sullivan. *Introduction to Uncertainty Quantification*, volume 63. Springer, 2015.

[40] M. Taboga. *The Monte Carlo method*. Kindle Direct Publishing, 2021. Online appendix.

[41] A. Tero, R. Kobayashi, and T. Nakagaki. A Mathematical Model for Adaptive Transport Network in Path Finding by True Slime Mold. *Journal of theoretical biology*, 244(4):553–564, 2007.

[42] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein Auto-Encoders. *ICLR*, 2018.

[43] D. Ulyanov, A. Vedaldi, and V. Lempitsky. It takes (only) two: Adversarial Generator-Encoder Networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[44] C. Villani. *Optimal Transport*, volume 338 of *Old and New*. Springer Science & Business Media, Berlin, Heidelberg, Oct. 2008.

[45] C. Villani. *Topics in Optimal Transportation*, volume 58. American Mathematical Soc., 2021.

[46] S. Wang, Y. Du, X. Guo, B. Pan, Z. Qin, and L. Zhao. Controllable Data Generation by Deep Learning: A Review. *Association for Computing Machinery*, 56, 2024.

[47] Y. Wang. A Mathematical Introduction to Generative Adversarial Nets (GAN). *Mathematical Modelling and Numerical Simulation with Applications*, 3, 2023.

[48] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks, 2016.

[49] J. Zhao, M. Mathieu, and Y. LeCun. Energy-based Generative Adversarial Networks. In *International Conference on Learning Representations*, 2017.

[50] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.