HERIOT-WATT UNIVERSITY

Machine learning methods for uncertainty quantification in subsurface reservoirs

by

Shing Cheng Chan Chang

A thesis submitted for the degree of Doctor of Philosophy

in the

School of Energy, Geoscience, Infrastructure and Society

December 2018

HERIOT-WATT UNIVERSITY

Abstract

School of Energy, Geoscience, Infrastructure and Society

Doctor of Philosophy

by Shing Cheng Chan Chang

We investigate current challenges in the reservoir engineering pipeline that can be addressed using recent machine learning techniques. Our emphasis is on improving the performance of uncertainty quantification tasks which are ubiquitous in subsurface reservoir simulations. In one work, we accelerate multiscale methods by embedding a neural network surrogate for the fast computation of the custom basis functions, replacing the need to solve the local elliptic problems normally required to obtain them. In a different work, we address current challenges in obtaining geological parametrizations that can capture complex geological structures. We adopt a neural network parametrization using a recent unsupervised learning technique, obtaining an effective parametrization that can reproduce high-order statistics of flow responses. In a follow-up work, we introduce a method for post-hoc conditioning of the neural network parametrization to generate conditional realizations by training a second neural network to sample from a Bayesian posterior and coupling it with the original network. In our final work, we introduce a framework for exemplar-based parametric synthesis of geological images based on a recent kernel method, obtaining a neural network parametrization of the geology using a single exemplar image.

Dedicated to my family

"I have always believed that scientific research is another domain where a form of opti mism is essential to success: I have yet to meet a successful scientist who lacks the ability to exaggerate the importance of what he or she is doing, and I believe that someone who lacks a delusional sense of significance will wilt in the face of repeated experiences of multiple small failures and rare successes, the fate of most researchers."

Daniel Kahneman, *Thinking, Fast and Slow.*

Acknowledgements

I would like to thank my professors at Instituto Universitario Aeronáutico: Mario Derrico and Carlos Sacco for their constant and invaluable support; Roberto Guibert, Carlos Kozameh, Pedro Pury, Gustavo Scarpin, Luis Serrano and Andres Liberatto for the inspiring lectures during my undergraduate studies; and German Weht for being a guide and a good friend.

I would like to thank Organización Multidisciplinaria de Apoyo a Profesores y Alumnos del Paraguay for showing me the beauty of mathematics through awesome math contests, where I learned that the hardest questions in science are often the easiest to understand, and conversely, many seemingly difficult problems often contain simple solutions.

I am deeply grateful to my supervisor Ahmed H. Elsheikh for granting me this wonderful opportunity to come and study in a First World quality institution, as well as for his support throughout this journey – including being my guarantor when I was looking for a flat, which was not a trivial thing to obtain. I also thank my colleague Nagoor Khan for being a great flatmate and friend.

Last but not least, I would like to thank my mom for her relentless support in all stages of my life – no achievements would have been possible without her; my grandparents for teaching me the value of education; and my uncle, my sister, and of course my friends, who despite the long distance and time zone manage to keep in constant touch with me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In an ideal world, mathematical models do not contain simplifications, model parameters are fully known, and measurements are clean. In reality, the world is rife with uncertainties. Reservoir engineering needs to account for incomplete descriptions of the subsurface, sparse and noisy measurements, and model simplifications. Airplane design needs to be robust to air conditions, distribution of passengers, and small differences in manufactured parts. A restaurant needs to respond to changing demands based on the time of the day, day of the week, and month. Many real-world problems do not have a deterministic answer, instead they can only be answered effectively using a statistical framework. For example, a sensical answer to "how many customers is a restaurant expecting tomorrow" should not be a single number, but a bound ("between 80 and 120"), or even better, a probability distribution. In short, any actionable prediction should be accompanied with a sense of the uncertainties in the information that it relied on.

Uncertainty quantification in an engineering setting typically relies on the evaluation of expensive numerical simulations for a large number of realizations of the unknown or uncertain parameters. This often entails the availability of large datasets of parameters, measurements and simulation results, which are ever growing while powered by decreasing computational costs and increasing storage capacity, thus providing a fertile environment for data-driven techniques. In this thesis, we study the application of different machine learning techniques in the uncertainty quantification pipeline in reservoir simulations. In particular, we demonstrate applications of machine learning for acceleration of subsurface flow simulations (Chapter 2), parametrization of geological models (Chapters 3 and 4), and parametric synthesis of geology (Chapter 5).

1.1 Why machine learning?

In the last two decades, the field of machine learning has seen a resurgence in interest from both academia and industry. This resurgence can be explained by at least two inevitable events in the history of computers: First, the incredible pace at which hardware is developed resulted in an uninterrupted exponential growth in computing power and storage capacity for several decades. This phenomena has been given the name of "Moore's law", which roughly states that the number of transistors that can fit in a chip doubles about every two years. To give an idea of its effects, a modern smartphone has several times more computing power than the NASA supercomputers of 1960 used to send humans to the moon. This in turn led to a second event: the inevitable widespread adoption of personal computers – in particular of mobile phones – as hardware became cheaper, leading to the explosion of data that we see today. For example, as of 2018, we upload about 300 hours of video to YouTube *per minute* [1] – and this is just one type of data in one platform. Since only about half the world population is currently online [2], and only a few of those have access to broadband, it is fair to say that data availability is nowhere near its peak.

The two events prepared the ground for many success stories in machine learning. The increase in computational power prompted the reconsideration of previous techniques that were once deemed impractical. Moreover, the boost in computational power came hand in hand with the boost in data availability, enabling the use of data-hungry techniques. Of all machine learning techniques that benefited from these events, the clear winners have been those based on *neural networks*, which strive in the presence of very large datasets and powerful computers. The success of neural networks comes from their seemingly unlimited performance: Given *enough* data and computational resources, neural networks have repeatedly shown to perform better than other methods in a variety of tasks. Indeed, many prominent achievements in the field are done using neural networks. For example, Apple Siri [3] and Google Duplex [4] both use recurrent neural networks for their voice recognition system. In computer vision, starting from the work of [5], pretty much all of the best performing methods in object recognition use convolutional neural networks, including those used in self-driving cars [6]. Finally, two recent examples are Google AlphaZero and OpenAI Five, where neural networks are trained via self-play (reinforcement learning) to beat humans in the board game Go [7], and in the popular strategy video game Dota 2 [8]. These state-of-the-art examples require an enormous amount of computing power and data, and would not have been possible without the resources of our time.

It quickly became evident that machine learning was a topic that could not be avoided. Industries were advised to embrace it in order to survive, data was branded "the new oil",

and machine learning was equated to "the new electricity" [9]. Research funding soared in both academia and industry, resulting in a rate increase of $\times 9$ in published papers in the field compared to 1996 [10], and even the appearance of extreme application-specific hardware such as TPUs [11] – note that the development of hardware to accommodate one particular software application is rarely seen. The fast evolving literature together with the involvement of the hardware sector to advance the field of machine learning provide a dynamic ecosystem that is being leveraged in more traditional industries.

We should note, however, that the current trend does not come without skepticism. As it often happens with transformative technologies, they are accompanied with wild speculations, and recent concerns highlight similarities of the current trends with the years predating the dotcom bubble. Nevertheless, the Internet *did* eventually change industries as well as societies in very fundamental ways; similarly, machine learning is likely to have a great impact in how industries operate in the future.

1.1.1 Machine learning and physics

The aforementioned application examples are instances of so-called AI-hard problems. These are problems that cannot be described nor solved using a set of rules or laws, or at least not so in a convenient manner. For example, there are currently no known set of rules that fully describe vision or speech. In Chess and Go, the rules are known but it is impossible to hard-code all the possible scenarios of a game. For these cases, data-driven methods provide an alternative to tackle the absence of mathematical models. This raises the question about the place of machine learning in the traditional engineering setting where the laws of physics are available. Another concern regards the validity and rigorousness of using data-driven approaches to modeling in place of mathematical models derived from the laws of physics.

Here we would like to provide a philosophical argument: Regardless of whether the laws of physics are invented or discovered, they originate from data. Hooke observed that the extension of a spring is proportional to its load. Grimaldi and Riccioli observed that the distance traveled by a free-falling object is proportional to the square of the time. Darcy derived his law describing porous media flow based on experiments with water and sand. Kepler derived his laws after many years of recording the motion of planets. Finally, Newton established the laws that generalize these and other previous observations. From Newton's laws, several physical models were axiomatically derived to describe different phenomena, and as the models stood the test of time, the laws became validated. However, it is worth acknowledging that underlying these laws is a data-driven *learning* process. As with any data-driven model, the models will be valid

within the data distribution that they were "trained". For example, Hooke's law only applies in the elastic range, Darcy's law only applies in the laminar regime, and Newton's laws break down outside the classical realm. In the latter case, it took us more than 200 years until we were able to explore the very small (quantum realm) and the very large and fast (relativistic realm) to discover the limits of the model – in particular, classical physics are only an (excellent) approximation of relativistic physics. Notably, a single model that can explain all realms (a theory of everything) remains one of the most important unsolved problems in physics. Nevertheless, each model does an excellent job in explaining the observations within their own scope. The bottom line is that a model should be regarded as long as it does the job, or as George E. P. Box often said, "All models are wrong, but some are useful".

1.1.2 Application examples

We discuss the applicability of machine learning methods in engineering. First, machine learning is hardly justified if the problem can already be solved by current physical models in an efficient manner. For example, it is unnecessary to train a data-driven model to describe the motion of a free-falling object using thousands of videos of free-falling apples, since this can be coded in a simple algebraic equation. In contrast, a much harder problem is to track arbitrary motions of arbitrary objects in a video, e.g. to achieve scene understanding for self-driving cars – hard-coding all possible scenarios becomes much more laborious. In some problems, a physical model needs to be evaluated several times (e.g. in uncertainty quantification), making the procedure very expensive when the physical model has a high computational cost; in these cases, a data-driven surrogate with lower evaluation cost could be constructed to replace the physical model – a large number of approximate solutions is sometimes preferred over a small number of high-fidelity solutions. In other problems, the current mathematical model for a system is inadequate or oversimplified, then a data-driven approach might provide an effective alternative. Finally, when no mathematical models are available, data-driven methods are often the only choice.

To summarize, engineering applications of machine learning are most effective when *data is available* and:

- When a mathematical model would be too complex or impossible to implement in practice.
- When current models are incomplete or inadequate. One recent example is the prediction of aftershocks following an earthquake [12], in which it was found that a neural network achieved higher accuracy than standard models.

- When the predictive model needs to be evaluated several times, and a data-driven model would be computationally cheaper. In effect, this is data-driven surrogate modeling and is widely applied in engineering.

- When laborious human assessment is involved. One example is the inspection of medical scans to detect tumors [13]. Another example is the interpretation of seismic images, done by geologists, for identification of salt deposits [14].

1.2 Basic concepts

Machine learning consists of a set of techniques to build a computer system to accomplish a task, without explicitly programming the necessary steps to achieve such task. This vague definition can take several concrete forms, from simple linear regression to systems trained using reinforcement learning. At a basic level, given a dataset of observations and a task, a machine learning technique consists of a hypothesis space (the model family, e.g. polynomials, neural networks, support vector machines, etc.) and a metric to measure the task performance (the objective function to be optimized, e.g. least square error, cross-entropy, distribution divergences, etc.). The objective is to infer from the dataset the optimal hypothesis for the task, as measured by the performance metric. The techniques are broadly classified into two groups according to the task:

- Supervised learning concerns the estimation of *conditional* distributions. Given a dataset of paired observations $\{(x_1, y_1), \cdots, (x_n, y_n)\}$, the goal is to model the conditional probability $p(y|x)$. In practice, we often settle for simpler objectives such as estimating correlations, conditional means, maximum a posteriori, maximum likelihood, etc. A simple example is the least squares approach to fit a polynomial f to minimize $\sum_i (y_i - f(x_i))^2$ over the dataset, equivalent to a maximum likelihood estimation using a Gaussian noise model. Supervised learning is applied in Chapter 2.

- Unsupervised learning concerns the full distribution of a dataset. Given a dataset of observations $\{x_1, \cdots, x_n\}$, the goal is to model the distribution $p(x)$. In this case, we often settle for the estimation of descriptive statistics (e.g. mean, median, standard deviation, etc.), identification of clusters, principal axes of variation (e.g. principal component analysis), anomaly detection, etc. Recent developments in unsupervised learning focus on *implicit modeling*: In some applications, the real interest does not lie in knowing the distribution p itself, but in efficiently sampling realizations from such distribution. Implicit modeling therefore aims to construct an efficient *procedure* to sample from p without explicit knowledge of p (unlike e.g.

Monte Carlo methods). The emphasis here is on the construction of the sampling procedure rather than modeling of the distribution, with the aim of obtaining efficient samplers for complex and high-dimensional distributions. The evident downside of this approach is the absence of p, which might be useful for sample assessment. Unsupervised learning is applied in Chapters 3 to 5.

Occam's razor

In contrast to the standard optimization setting where the goal is to find the global extremum of an objective function, the goal in machine learning is *generalization*, meaning that the computer system should be able to perform effectively not only in the training dataset, but also under new scenarios (that is, the model needs to be accurate for new inputs not present in the training set).

For a concrete example, we can imagine the initial attempts towards learning the motion of free-falling objects, presumably from data consisting of paired measurements of the distance and time. As we know, a line is not sufficient to fit this data, whereas a quadratic polynomial can fit it perfectly (neglecting measurement noise and other effects). However, note that polynomials of higher degree could also fit the data; in fact, for any finite set of points there exist infinitely many polynomials of sufficiently high degree that fit the points perfectly. Nevertheless, a strong *intuition* tells us that the quadratic polynomial is the correct hypothesis for this data, that polynomials of higher degree are unnecessary, and that a line is insufficient. This intuition is at the heart of the learning problem and is summarized in the principle of Occam's razor: *Among all hypotheses that equally explain a phenomena, the simplest is often the correct one.* This principle can be thought of as analogous to the principle of least effort in physics; indeed, many notable physicists have given their own version of Occam's razor including: "Everything should be made as simple as possible, but not simpler" by Albert Einstein, and "You can recognize truth by its beauty and simplicity" by Richard Feynman.

When a model performs well in the training data but poorly outside of it, we say that the model simply "memorized" or "overfitted" the training data. As an extreme example, consider a "database function" f defined as $f(x_i) = y_i,\ (x_i, y_i) \in$ dataset, else $f(x) = 123$, which would perform perfectly (have zero error) in any finite dataset. True learning happens when the model performs well both inside and outside the training set; in such case, we say that the model *generalizes*.

In simple cases where we only have one or two dependent variables, visualization aids in detecting overfitting and choosing the correct hypothesis, hence the learning problem becomes a trivial matter that can be resolved using our intuition. In more practical

problems, however, we can encounter dozens or even millions of dependent variables and therefore visualization becomes difficult; moreover, our intuition often fails in very high dimensions (see e.g. [15]). For example, a house pricing model depends on the property size, distance to the city center, number of rooms, number of windows, etc., easily resulting in dozens of variables. An object recognition system trained on 1024×768 color images results in $3 \times 1024 \times 768$ variables (in the RGB format). In such cases, it becomes much more difficult to determine the right hypothesis. The essence of machine learning is to apply the principle of Occam's razor in an automated manner – without this requirement, machine learning would be no different than curve-fitting.

1.3 Outline of the thesis

Uncertainty quantification in reservoir simulations contains unique challenges that make it computationally expensive. Due to the physical extent of the simulated domain together with the heterogeneity of the subsurface, subsurface simulations usually require the use of extremely high grid resolutions in order to model the flow accurately. Additionally, the impossibility of obtaining direct measurements of the whole subsurface results in the need to perform the already expensive simulations for a very large number of times (in the order of thousands or even millions) for uncertainty quantification, causing even higher computational costs.

A variety of methods have been proposed to reduce the computational cost of uncertainty quantification in reservoir simulations. We can roughly categorize these methods based on whether they aim to reduce the computational burden of the simulator itself (run faster), or reduce the number of simulations required by refining the solution space (run smarter). Diverse methods have been dedicated to the first group, from methods that reduce the model complexity, to surrogate models, to methods amenable to parallelization. In Chapter 2, we introduce a method belonging to this category, where we embed a surrogate model into a model order reduction method in order to obtain further speedups. The proposed method is therefore a hybrid between model order reduction techniques and surrogate models.

Within the second category, parametrization is particularly useful in the context of subsurface simulations where the large number of uncertain variables are highly correlated and redundantly represented as a consequence of the grid discretization. One useful analogy to parametrization is an index of words or a dictionary (see Figure 1.1): Consider the task of inferring the content of a book based solely on the frequency of letters. A priori, this task would need to consider any possible arrangement of letters however implausible. On the other hand, since most books consist of words, we know that most

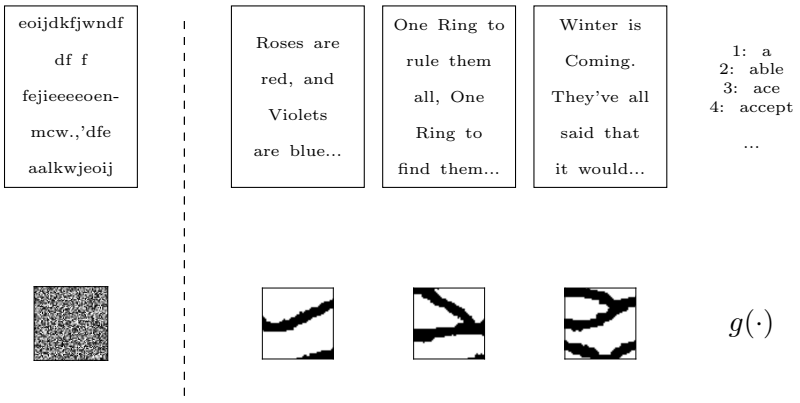


FIGURE 1.1: A dictionary provides all plausible arrangement of letters (top row). Similarly, a geological parametrization provides all plausible realizations of the subsurface (bottom row).

arrangements are unlikely and can be quickly discarded – such information can be conveyed via a dictionary. Likewise, real subsurface images are not completely random but instead contain clear spatial correlations. By using a suitable parametrization of the subsurface, we can narrow our focus on only plausible realizations, thus reducing the number of simulations required in uncertainty quantification and inversion problems. Parametrization is the subject of Chapters 3 to 5, where we parametrize the geology using recent techniques in unsupervised learning.

The following is a summary of each of the remaining chapters. Note that each chapter intends to be self-contained as they were initially written as separate papers.

Chapter 2: A machine learning approach for efficient uncertainty quantification using multiscale methods

The multiscale finite volume method (MsFV) is a popular method that aims to reduce the computational complexity of elliptic problems. It is particularly useful for simulations on extremely high grid resolutions (a common scenario in subsurface simulations). It achieves this by using customized basis functions to solve the problem on a much coarser grid. These custom basis functions are derived from solving small localized elliptic problems over the domain. Thus, unlike finite element methods where the basis functions are piecewise polynomials (e.g. linear, quadratic, cubic, etc.), the basis functions in MsFV are numerically customized with the aim of better capturing the heterogeneity in the underlying properties. In the context of uncertainty quantification, the localized problems would need to be computed several times for each realization of the properties to obtain the corresponding basis functions, after which they are used to solve the global problem and then discarded from memory. We see this as free data that can be further leveraged to build a surrogate model to obtain the basis functions more efficiently,

replacing the need to repeatedly solve the localized problems in the subsequent series of runs and further accelerating the uncertainty quantification task. We use a neural network surrogate, motivated by the high expressive power as well as recent advances in machine learning that greatly improve the effectiveness of neural networks. In our experiments, the surrogate achieves a speedup of two orders of magnitude in obtaining the basis functions without compromising the simulation results.

Chapter 3: Parametrization of stochastic inputs using generative adversarial networks with application in geology

In this and subsequent chapters, we direct our attention to the parametrization of geological images. A historical challenge in the parametrization of geological images is the preservation of visual patterns of the images, as well as high-order spatial statistics. This is in part due to the simplifying assumptions in the mathematical modeling that are often necessary to make the implementation feasible. For example, principal component analysis assumes a simple linear combination of basis functions, which are in turn devised to preserve mere covariances. Here we adopt a neural network parametrization, whose expressive power makes it one of the most flexible forms of parametrization. Moreover, instead of explicitly defining the spatial statistics to be preserved, we let this be learned by a second neural network. This approach is possible due to a very recent technique in machine learning called generative adversarial networks. The idea is to simultaneously train two competing neural networks: the generator (parametrization) network is trained to generate plausible images, while a discriminator network is trained to correctly classify between "fake" (generated) images and "real" images (i.e. from a dataset of prior images). In this manner, the generator is iteratively encouraged to generate better images in order to fool the discriminator. The appeal in this approach is that no spatial statistics are hand-crafted, instead they are implicitly learned by the discriminator. As such, the expressive power of neural networks are leveraged twice: it is leveraged in the generator to reproduce complex images, and in the discriminator to learn complex high-order statistics of the images. Our results show that the neural parametrization is very effective in reproducing the visual patterns, and more importantly, the high-order statistics of the flow responses in an uncertainty propagation study.

Chapter 4: Parametric generation of conditional geological realizations using generative neural networks

We extend the work in the previous chapter and introduce a method for post-hoc conditioning of a pre-trained generator to new observations, i.e. given a generator

trained on unconditional realizations, we wish to generate realizations conditioned on new spatial observations. We begin by using a Bayesian framework to formulate the posterior distribution of the latent vector given observations. Next, we train a neural network to sample from this posterior by minimizing the Kullback-Leibler divergence between the network distribution and the posterior. Finally, we couple this neural network with the original unconditional generator to obtain the conditional generator, thus maintaining the parametrization of the sampling process. In our experiments, the method shows very good results for several test cases considered, honoring the conditioning while producing diverse realizations.

Chapter 5: Exemplar-based parametric synthesis of geology using kernel discrepancies and generative neural networks

In this chapter, we look at a very different approach to obtain a parametrization of the geology. So far we have considered parametrization methods that require the availability of a large dataset of realizations that inform the spatial patterns and variability of the subsurface, serving as the training set for the generative neural network. Here we introduce a different approach that only requires a single exemplar image to train the generator. We first introduce an energy function that measures the plausibility of a geological image with respect to the exemplar. This function evaluates the discrepancy in the patch distributions of the image and the exemplar. The assumption is that geological images A and B are equivalent if we cannot distinguish a bag of patches extracted from A from a bag of patches extracted from B. Having defined the energy function, it is already possible to synthesize new realizations by minimization using e.g. gradient-descent; however, this approach is slow and non-parametric, therefore we introduce a method for parametric synthesis by training a generative neural network to sample solutions of the minimization problem. Our experiments show that the method obtains very good synthesis results, reproducing the visual patterns of the exemplar and showing good agreement in the spatial statistics.

We conclude our thesis in Chapter 6.

Chapter 2

A machine learning approach for efficient uncertainty quantification using multiscale methods

Several multiscale methods account for sub-grid scale features using coarse scale basis functions. For example, in the Multiscale Finite Volume method the coarse scale basis functions are obtained by solving a set of local problems over dual-grid cells. We introduce a data-driven approach for the estimation of these coarse scale basis functions. Specifically, we employ a neural network predictor fitted using a set of solution samples from which it learns to generate subsequent basis functions at a lower computational cost than solving the local problems. The computational advantage of this approach is realized for uncertainty quantification tasks where a large number of realizations has to be evaluated. We attribute the ability to learn these basis functions to the modularity of the local problems and the redundancy of the permeability patches between samples. The proposed method is evaluated on elliptic problems yielding very promising results.

Published in Journal of Computational Physics (2017): https://doi.org/10.1016/j.jcp.2017.10.034

2.1 Introduction

Uncertainty quantification is an important task in practical engineering where some parameters are unknown or highly uncertain. After selecting adequate priors for the uncertain parameters, simulations are performed for a large number of realizations. In the particular case of reservoir simulations, the problem is further aggravated where very fine details of the geological models are needed (large number of cells) for accurate description of the flow. One traditional approach to address this problem is to upscale the geological models. Another more recent approach is to use multiscale methods. In these methods, the global fine scale problem is decomposed into many smaller local problems. The solution of these smaller local problems results in a set of numerically computed basis functions which are then used to build a coarse system of equations. After solving the coarse system, an interpolation is performed with the basis functions to obtain the fine scale solution.

We note that in multiscale methods, a large number of local problems are solved under the same boundary conditions to obtain the required basis functions. This process is repeated for each geological realization in uncertainty quantification tasks. Our aim is to exploit the redundancy that may arise in solving these local problems for several geological realizations by introducing a data-driven approach for estimating the basis functions efficiently. Specifically, we exploit the large number of local problem solutions that become available after a given number of full runs to construct a computationally cheap function that generates approximate solutions to local problems, i.e. approximate basis functions. In effect, what we propose is a type of hybrid surrogate model by embedding a data-driven approach into the multiscale numerical method. In this work, we focus on one multiscale method called the Multiscale Finite Volume method (MsFV) introduced by Jenny et al. [16]. However, the proposed approach can be applied to any multiscale method where the explicit construction of basis functions is performed such as in the more recent multiscale method based on restriction-smoothed basis functions (MsRSB) [17].

Aarnes and Efendiev [18] introduced a multiscale mixed finite element (MsMFE) method for porous media flows with stochastic permeability field where a set of precomputed basis functions is constructed based on selected set of realizations of the permeability field. These basis functions are then used to build a low-dimensional approximation space for the velocity field. We note that the cost of solving the upscaled problem (i.e. coarse scale) in [18] increases with the number of selected set of realizations. In contrast, in this manuscript we directly address the generation of basis functions via a "black box" surrogate modeling approach using machine learning. The generated basis functions are then directly employed in the multiscale formulation without any

further modification. Another major difference is that our method directly benefits from increasing the number of realizations used to build the surrogate model without any increase in the computational cost of solving the coarse scale problems for new realizations.

This paper presents the first attempt to combine/embed machine learning techniques within multiscale numerical methods with very promising results. The motivation for our work comes from the observation that computational power and data storage capacity are ever increasing. This trend is likely to continue for some time and results in an increased ability to store and data-mine large volumes of simulation data. Another source of motivation comes from the renewed interest in machine learning among the research community, specially in the branch of deep learning to tackle AI-complete tasks such as computer vision and natural language processing. Neural network models are regarded as *universal function approximators* [19–21] with capacity to learn highly non-linear maps. Therefore, they seem to be suitable for our current application.

The rest of this chapter is organized as follows: In Section 2.2, we give a brief description of the multiscale finite volume method (MsFV) and neural networks (NN). In Section 2.3, we present the methodology for the proposed approach for machine learning the basis. In Section 2.4, we examine the effectiveness of the presented method for uncertainty quantification in two test cases. Finally, in Section 2.5 we report the conclusions of this work along with a brief discussion of future directions.

2.2 Background

In this section we briefly describe the two main components of the proposed method: multiscale finite volume (MsFV) methods and neural networks (NN) for surrogate modeling. A number of variants of the MsFV method have been proposed since its introduction in [16]. In this work, we employ the MsFV method as described in [22, 23].

2.2.1 Multiscale finite volume method

We consider an elliptic equation describing pressure

$$-\nabla \cdot (\mathbf{K}\nabla p) = q \tag{2.1}$$

where p denotes the fluid pressure, q denotes fluid sources, and $\mathbf{K}$ denotes the permeability tensor. Discretizing Eq. (2.1) by the finite volume method results in a system of

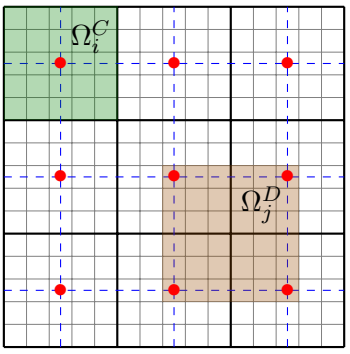


FIGURE 2.1: A square domain with a fine discretization of size 15×15 (grey thin lines). A coarse *primal grid* of size 3×3 is defined on top of the fine grid (black bold lines). A *primal cell* Ω_i^C is highlighted in green. The centers of each primal cell are marked with red dots. From these centers, the *dual grid* is defined (blue dashed lines). A *dual cell* Ω_j^D is highlighted in light red.

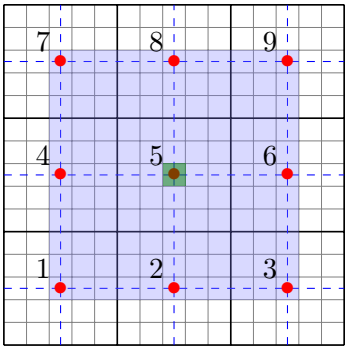
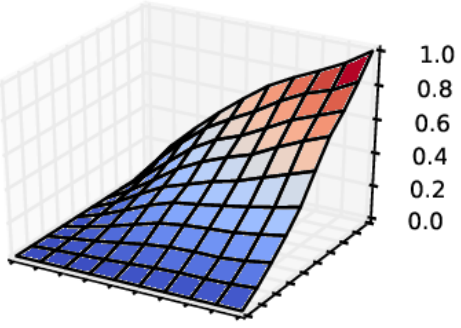
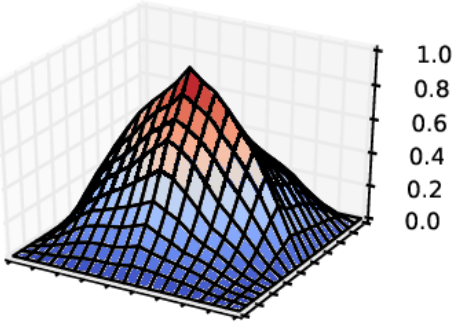


FIGURE 2.2: Shown in blue is the support region of basis function ϕ^5 corresponding to coarse node 5 (green cell). Basis function ϕ^5 is obtained by solving the local problems in Eq. (2.2) for $i = 5$, then $\phi^5 = \sum_{j=1}^{N_D} \phi_j^5$. In this example, only ϕ^5 is an *interior* basis function (see Section 2.3).



(A) A partial basis function.



(B) A complete basis function.

FIGURE 2.3: Illustration of a local solution (partial basis function) and a basis function within its support region.

equations of the form $\mathbf{Ap} = \mathbf{q}$, which for some applications (such as reservoir simulation) tends to be extremely large. The MsFV method tackles this problem by constructing and solving a much coarser system of equations $\mathbf{A}_C\mathbf{p}_C = \mathbf{r}_C$, the solution of which is then used to obtain an approximation of $\mathbf{p}$ by interpolation. For this, it relies on a set of basis functions which are obtained by solving local problems. In this sense, the method slightly resembles the finite element method, except that the basis functions employed are not piecewise polynomials but numerically computed functions.

The method begins with the definition of a pair of overlapping coarse grids, namely the *primal grid* and the *dual grid*, as shown in Figure 2.1. In principle, the primal grid can be any coarse partition defined over the fine grid. Next, we define the *coarse nodes* as the fine cells at the centers of each primal cell. Lastly, the dual grid is defined by the lines connecting these coarse nodes. We denote the primal cells with $\Omega_i^C,\ i \in \{1, \cdots, N_C\}$, and the dual cells with $\Omega_j^D,\ j \in \{1, \cdots, N_D\}$.

A set of (partial) basis functions are obtained by solving the local problems

$$\begin{aligned} \nabla \cdot (\mathbf{K} \cdot \nabla \phi_j^i) &= 0 \quad \text{in } \Omega_j^D \\ \nabla \cdot {}_{\parallel}(\mathbf{K} \cdot \nabla \phi_j^i)_{\parallel} &= 0 \quad \text{on } \partial\Omega_j^D \\ \phi_j^i(\mathbf{x}_k) &= \delta_{ik} \quad k \in \{1, \cdots, N_C\}, \end{aligned} \tag{2.2}$$

where ϕ_j^i denotes the (partial) basis function on dual cell Ω_j^D (see Figure 2.3a) associated with coarse node i, $\mathbf{x}_k$ denotes the coordinate of coarse node k, and ${}_{\parallel} \cdot {}_{\parallel}$ denotes the tangential component over the dual cell boundary $\partial\Omega_j^D$. In the 2D case, this means solving 1D problems over $\partial\Omega_j^D$, the solutions of which become the boundary conditions for the 2D problem on Ω_j^D.

Another component of the MsFV formulation employed are the correction functions, obtained by solving the local problems

$$\begin{aligned} \nabla \cdot (\mathbf{K} \cdot \nabla \hat{\phi}_j) &= q \quad \text{in } \Omega_j^D \\ \nabla \cdot {}_{\parallel}(\mathbf{K} \cdot \nabla \hat{\phi}_j)_{\parallel} &= q \quad \text{on } \partial\Omega_j^D \\ \hat{\phi}_j(\mathbf{x}_k) &= 0 \quad k \in \{1, \cdots, N_C\}, \end{aligned} \tag{2.3}$$

where $\hat{\phi}_j$ denotes the correction function on dual cell Ω_j^D.

Once the basis and correction functions are obtained, we approximate the fine scale pressure p as an interpolation of coarse scale pressure values $\mathbf{p}_C$ plus a correction term

$$p \approx \sum_{j=1}^{N_D} (\sum_{i=1}^{N_C} \phi_j^i p_C^i + \hat{\phi}_j) \tag{2.4}$$

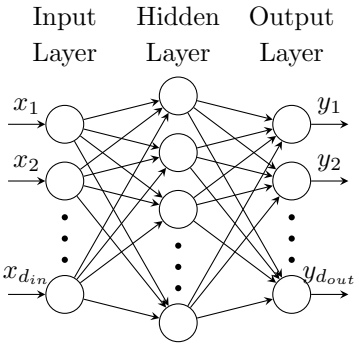


FIGURE 2.4: Representation of a 1-hidden layer neural network as a graph. The first column of nodes (from left to right) is the input layer, taking inputs $\mathbf{x} = (x_1, \cdots, x_{d_{in}})$ of dimension d_{in}, and the last column is the output layer with output $\mathbf{y} = (y_1, \cdots, y_{d_{out}})$ of dimension d_{out}. The intermediate column is the hidden layer. Each line connecting two nodes represents a multiplication by a scalar weight.

Substituting Eq. (2.4) in Eq. (2.1), and applying finite volume discretization over the primal grid, we get the coarse system of equations $\mathbf{A}_C \mathbf{p}_C = \mathbf{r}_C$.

In the current work, it is more convenient to express the basis and correction functions in a global point of view (see Figure 2.3b) by writing $\phi^i = \sum_{j=1}^{N_D} \phi_j^i$ and $\hat{\phi} = \sum_{j=1}^{N_D} \hat{\phi}_j$ (for the basis functions, we actually only need to sum over supporting dual cells, i.e. dual cells that are associated with the corresponding node). Then, Eq. (2.4) has a simpler interpolation expression of the form:

$$p \approx \sum_{i=1}^{N_C} \phi^i p_C^i + \hat{\phi} \tag{2.5}$$

In the case that the pressure solution will be utilised to drive a transport problem at the fine scale, a flux reconstruction step consisting of solving additional local Neumann problems is necessary [22].

2.2.2 Feedforward neural networks for surrogate modeling

A *feedforward neural network* f is a function consisting of a compositional chain of simpler functions, i.e. $f(\mathbf{x}) = f^{(n)}(f^{(n-1)}(\cdots(f^{(1)}(\mathbf{x}))\cdots))$. Here, f is said to have n *layers*, where $f^{(1)}$ is the first layer, $f^{(2)}$ the second layer, etc. The first layer is also called the *input layer*, the last layer the *output layer*, and the layers in between are called *hidden layers*. The size or *number of units* of a layer $f^{(i)} : \mathbb{R}^{d_{in}^i} \to \mathbb{R}^{d_{out}^i}$ refers to its output dimension, i.e. d_{out}^i. Neural networks are typically represented as a graph as shown in Figure 2.4.

The input layer encompasses any pre-processing of the data, while the output layer may take many forms depending on the learning task. For example, in classification tasks, the *softmax* function is normally employed where the output layer size (number of units) equals the number of classes, and the softmax output represents the probability of the input sample belonging to each class. This is a way of embedding prior information of the learning task into the model. For regression tasks, the identity function is normally employed, but other functions can also be chosen to embed any prior information of the output space.

A conventional hidden layer is an affine transformation followed by an element-wise nonlinear function or *activation*, i.e. $f^{(i)}(\mathbf{x}) = \sigma_i(\mathbf{W}_i\mathbf{x} + \mathbf{b}_i)$. Some popular choices for σ_i are the hyperbolic tangent, the sigmoid function, and more recently, the rectified linear unit [24].

2.2.2.1 Neural network optimization

Once the network *architecture* is defined (number of layers n, layer sizes d^i_{out}, activation functions σ_i, etc.), the optimization problem is to find $\boldsymbol{\theta} = [\mathbf{W}_1; \mathbf{b}_1, \cdots, \mathbf{W}_n; \mathbf{b}_n]$ such that f best describes the observations. Let $\{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_N, \mathbf{y}_N)\}$ be the set of observations used to train the model (the *training set*), then the minimization problem is formulated as:

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i; \boldsymbol{\theta})||_2^2 \tag{2.6}$$

where $J(\boldsymbol{\theta}) \coloneqq \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i; \boldsymbol{\theta})||_2^2$ is the *cost function*. This problem can be solved using gradient-based algorithms such as gradient descent:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.7}$$

where ϵ is the *learning rate*. The derivative $\nabla_{\boldsymbol{\theta}} J$ can be obtained with numerical differentiation schemes. In neural networks, this is typically the *backpropagation* algorithm [25].

More recent gradient-based algorithms improve over gradient descent by offering adaptive learning rates such as AdaGrad [26], RMSProp [27] and Adam [28]. The basic idea in these methods is to use a separate learning rate for each scalar parameter, and adapt these rates throughout the training process based on the historical values of the partial derivatives with respect to each parameter. The initial global learning rate ϵ_0 is a tunable hyperparameter.

A recent important development regarding neural network optimization is *batch normalization* [29], which has shown to significantly speed up the optimization process. The

method consists of adaptive reparametrization of inputs to each activation function. In essence, the values after the affine transformation in a layer are normalized by the mean and standard deviation before being fed into the layer activation function.

2.2.2.2 Regularization

The optimization of $\boldsymbol{\theta}$ from Eq. (2.6) alone yields a model that is prone to overfitting, i.e. it does not necessarily perform well for samples not seen in the training set. Hence, validation assessment is necessary where a separate set of samples that are not used in optimizing Eq. (2.6), called the *validation set*, is employed to assess the accuracy of the model. One simple regularization technique is *early stopping*, where the model is assessed after each update (or number of updates) of $\boldsymbol{\theta}$ in Eq. (2.7); when the cost function on the validation set begins to increase, the optimization is early stopped. This is the stopping criteria in neural network optimization, which differs from conventional optimization where the criteria is generally based on the gradient norm. Another set of regularization techniques are *parameter norm penalties*, where an additional term is added to the cost function:

$$J_{reg}(\boldsymbol{\theta}) \coloneqq \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{y}_i - f(\mathbf{x}_i;\boldsymbol{\theta})||_2^2 + \alpha\Omega(\boldsymbol{\theta}) \tag{2.8}$$

For example, for L^2 parameter norm, $\Omega(\boldsymbol{\theta}) = \frac{1}{2}||\boldsymbol{\theta}||_2^2$. The additional parameter α is a regularization *hyperparameter* that is chosen using the validation set.

More recently, *Dropout* [30] has shown to be a very effective regularization technique that approximates model averaging in neural networks. The technique consists of randomly dropping out units of the network during the optimization iteration, by which the optimizer "sees" a number of different "models" in the process. Since traditional model averaging in neural networks is usually extremely expensive, this approach serves as a proxy for averaging an exponential number of models. In practice, a *dropout rate* is chosen which indicates the probability of a unit being dropped out. This hyperparameter is tuned using a validation set. The authors in [30] suggested using a *max-norm* constraint along with dropout, which consists of constraining the norm of some of the weights by a fixed constant c, tuned with a validation set. This allows for more aggressive optimization search without the possibility of weights blowing up.

2.2.2.3 Architecture design and hyperparameter tuning

The design of the network architecture, i.e. defining the number of layers, the number of units for each layer, activation functions, regularizers, etc. is not a straightforward

task. In principle, one can consider these parameters as additional hyperparameters and tune them using a validation set. However, hyperparameter optimization is an expensive task given that the objective function (performance on the validation set) is non-linear and non-differentiable with respect to the hyperparameters. In practice, heuristics and expertise are heavily employed in the design process to reduce the number of hyperparameters. Nevertheless, general guidelines do exist for the design of neural network models. The following is a compilation of guidelines extracted from [31, 32]:

- Begin with a few number of layers and units, and well-tested optimizers and regularizers.
- Start with as few hyperparameters as possible to enable quick manual search to obtain some insight of the learning task.
- Overfit, then regularize: increase the number of layers/units to overfit the training set, then apply regularization techniques to improve generalization. That is, we first want the network to be complex enough to approximate the target function, and then regularize it to perform well outside the training set.
- Regarding the choice of activation functions, the current default recommendation is to use rectified linear units (ReLUs). These have beneficial properties for the optimization such as non-vanishing gradients.
- Whether to use few large layers, or many small layers is an open debate. It is generally believed that many small layers generalize better, although the ultimate decision will largely depend on implementation and trial and error.
- Early stopping should almost always be employed.
- The learning rate ϵ_0 is very influential in the model performance and should be fine-tuned.
- If there is an architecture that performs well for a similar task, use it as the base architecture.

Once the general architecture is selected, key hyperparameters should be optimized. Traditional techniques such as grid search and random search are normally prohibitive. In the current manuscript we employed the *Tree-Parzen Estimator* [33], a sequential model-based hyperparameter optimization approach where a model is sequentially constructed to approximate the performance of hyperparameters based on historical measurements, and then subsequently choose new hyperparameters to test based on this model. Other hyperparameter optimization techniques include Bayesian optimization for neural networks [34] and Hyperband [35].

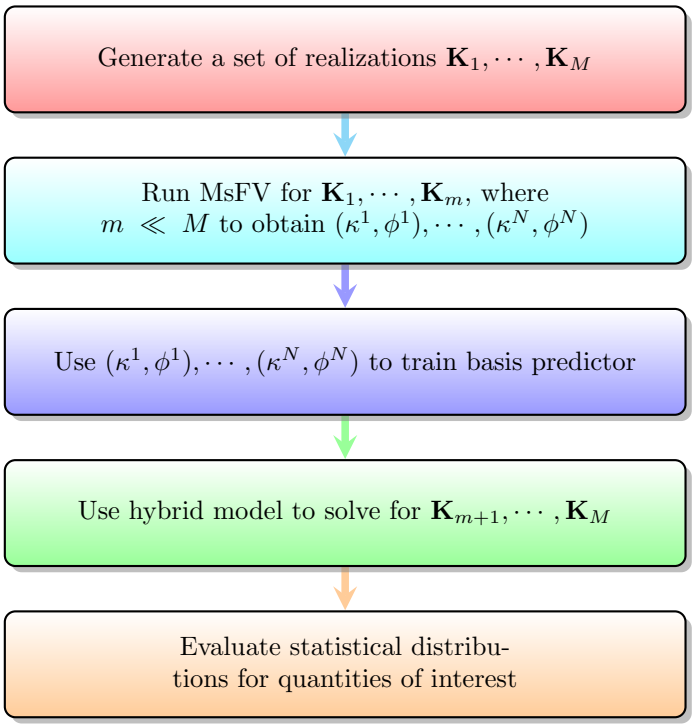


FIGURE 2.5: Workflow of the proposed method.

2.3 Methodology

We first introduce some terminology to simplify the presentation. A basis function is *interior* if its support region does *not* touch the domain boundary (see Figure 2.2). For practical purposes, we limit the learning process to interior basis functions, i.e. we build a predictor to generate interior basis functions while computing the remaining basis functions as usual from the local problems defined by Eq. (2.2). In practice, this should not be a major concern given that the number of interior basis functions is normally much larger than the number of non-interior basis functions (basis functions on the *edges* and *vertices* of the domain). In any case, it is pretty straightforward to train additional predictors for the remaining types of basis functions.

We define a *permeability patch* κ^i as the cropped region of the permeability field $\mathbf{K}$ that corresponds to the support region of a basis function ϕ^i. In the learning framework, the permeability patches $\kappa^1, \kappa^2, \cdots, \kappa^N$ are our inputs, and the corresponding basis functions $\phi^1, \phi^2, \cdots, \phi^N$ are our outputs. In practice, it is more convenient to work with the log-permeabilities, i.e. $\log \kappa^1, \cdots \log \kappa^N$.

For clarification, suppose we have an 81×81 Cartesian grid, over which we defined a 9×9 primal grid. Then there are $7 \times 7 = 49$ interior basis functions, each with array size (number of fine cells in the support region) of 19×19. The array size of each input

permeability patch is $19 \times 19 \times d$, where $d = 1$ for isotropic fields, and $d = 2$ in the anisotropic case.

The method we propose aims to speedup uncertainty quantification studies where multiscale methods are employed in the propagation task. In particular, we focus on the use of the MsFV method to solve Eq. (2.1) for large number of realizations of the permeability field $\mathbf{K}$. Our method however could be applied to any multiscale method where the explicit generation of the basis functions is performed. Consider an uncertainty propagation task of solving Eq. (2.1) for $\mathbf{K}_1, \cdots, \mathbf{K}_M$. Let $m \ll M$ be the number of full MsFV runs that we can afford. For each $\mathbf{K}_i$ where $i = 1, \cdots, m$, the MsFV simulation delivers a set of basis functions with their corresponding permeability patches. The union of these sets provides the learning dataset $\{(\kappa^1, \phi^1), \cdots, (\kappa^N, \phi^N)\}$. This dataset is used to train a predictor model that maps from permeability patches κ to basis functions ϕ that has an evaluation cost that is much cheaper than solving the local problems. The model we employed here is a neural network. Once the model is trained, it is used to predict the basis functions in the subsequent runs for $\mathbf{K}_{m+1}, \mathbf{K}_{m+2}, \cdots, \mathbf{K}_M$. The end result is a hybrid approach where the MsFV formulation is modified to obtain the basis functions through a data-driven model instead of being computed from the local problems. Figure 2.5 summarises the workflow of the proposed method.

To ensure the partition of unity property of the basis functions (which is not necessarily fulfilled in the learning model), we perform a post-processing step on the generated basis functions as follows:

$$\phi^i_{new}(x) = \frac{\phi^i(x)}{\sum_k \phi^k(x)}$$

We note that the presented method benefits from the use of structured grids with coarse cells of same size. In the case of structured grids with cells of different sizes, the patches could be scaled to a unique input size. This is a standard preprocessing step in computer vision to handle images of different sizes. We also note that handling unstructured grids is not a straightforward task and is beyond the scope of the current manuscript.

2.3.1 Implementation and computational aspects

The computational gain of the proposed method is achieved by replacing the solution of local problems by a constant number of matrix-vector multiplications followed by element-wise function evaluations. For a network of input and output sizes n and hidden layers of size m, this means an evaluation complexity of $\mathcal{O}(mn)$. The constants associated with the evaluation cost will depend on the number of layers of the network.

Efficient algorithms for solving systems of n linear equations exist where complexities of $\mathcal{O}(n \log n)$ (FFT) or even $\mathcal{O}(n)$ (multigrid methods) are achieved, however the constants associated with these algorithms are large, usually requiring a large value of n to be economical. This is in contrast to the MsFV approach where small local problems are preferred. Likewise, there exist efficient algorithms to perform matrix-vector multiplications that are only justified in practice when the matrices and vectors are very large.

Regarding the training of the neural network, this is performed in an *offline phase* as with other surrogate modeling techniques, and the justification of the cost will depend on the particular uncertainty quantification task at hand. The larger the uncertainty quantification task, the larger the time budget that can be assigned to the surrogate modeling process. As a practical note, it is worth mentioning that due to the surge of interest in neural networks and AI in general, efficient implementations have been intensively developed in recent years, both from the software and hardware sectors. Indeed, dedicated hardware devices are currently being released for various neural network implementations.

2.3.2 Other machine learning techniques

The proposed algorithm is not limited to neural networks and other traditional machine learning techniques are indeed applicable such as Gaussian processes and support vector machines to model the basis function predictor. A number of reasons led us to choose the neural network model. First, neural networks are universal approximators [19], meaning that they can fit any measurable function with arbitrary accuracy. This practically covers any function encountered in engineering applications. Secondly, neural networks scale very well with the size of the dataset, in contrast to Gaussian processes and support vector machines. This is desirable where the trends of the cost of numerical simulations and data storage are ever decreasing. Lastly, research in neural networks is characterized by a remarkably large and evolving body of literature from which we can benefit in the near future. Advances in the field is specially strong in problems related to computer vision, which shares a lot of features with our work.

2.4 Numerical experiments

We consider the task of solving Eq. (2.1) over a unit square $[0, 1]^2 \subset \mathbb{R}^2$. The domain is discretized into 81×81 fine grid, with a primal coarse grid of 9×9. For estimating the statistical distributions, we utilize $M = 1000$ realizations of isotropic log-permeability

TABLE 2.1: R^2-scores on different permeability types

Correlation length	R^2-score
$L = 0.1$	0.927
$L = 0.2$	0.953
$L = 0.4$	0.964

fields generated assuming a zero mean gaussian random field with an exponential covariance of the form

$$\mathrm{Cov}(\mathbf{x}_1, \mathbf{x}_2) = \sigma^2 \exp\left(-\frac{\|\mathbf{x}_1 - \mathbf{x}_2\|}{L}\right)$$

where $\|\cdot\|$ denotes the Euclidean norm. We choose $\sigma = 1.0$, and we investigate three values for the correlation length: $L = 0.1$, 0.2, and 0.4.

2.4.1 Learning process

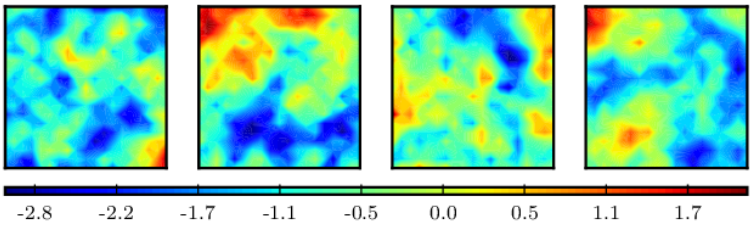
We assume a budget of $m = 20$ full MsFV runs, obtaining a dataset of 980 samples (since each run yields 49 samples). The array sizes of the inputs (permeability patches) and outputs (basis functions) are 19×19. We set aside 20% of the dataset for validation (this should be done at the level of the realizations, i.e. samples generated from 4 full MsFV runs).

The architecture employed is a fully connected network with 1-hidden layer of size 1024 and ReLU activation function. Naturally, the input and output layers are of size $19 \times 19 = 361$, matching the size of the permeability patch and basis function. Additionally, we employ the hard sigmoid function as the activation of the output layer. This is to embed the prior knowledge that basis functions take values between 0 and 1. The hard sigmoid is the function $x \in \mathbb{R} \mapsto \max(0, \min(1, 0.2x + 0.5))$. This choice of output activation, despite usually being problematic for gradient-based optimizations, gave good results when coupled with dropout and batch normalization.

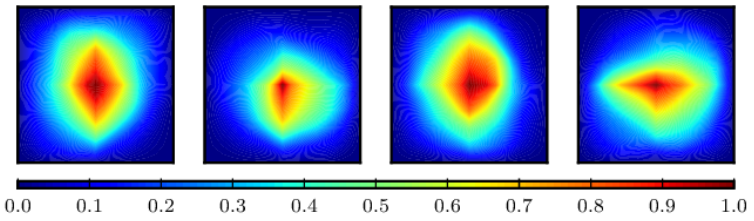
To train the network, the gradient-based optimizer Adam [28] seemed more robust during our trials. The initial learning rate was set to $\epsilon_0 = 10^{-3}$. For regularization, a dropout rate of 5% after the hidden layer, and a max-norm constraint of 4 have proven useful. Additionally, early stopping is employed. All these hyperparameter values were chosen based on default recommendations along with some manual explorations.

A convenient metric employed to report the performance of a trained model is the coefficient of determination (or R^2-score):

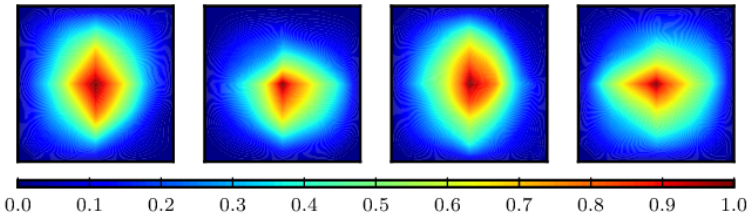
$$R^2(f) = 1 - \frac{\sum \|\phi^i - \hat{\phi}^i\|^2}{\sum \|\phi^i - \bar{\phi}\|^2}$$



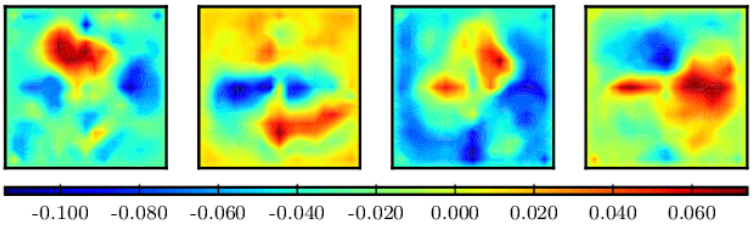
(A) Input $\log \kappa$



(B) Target ϕ



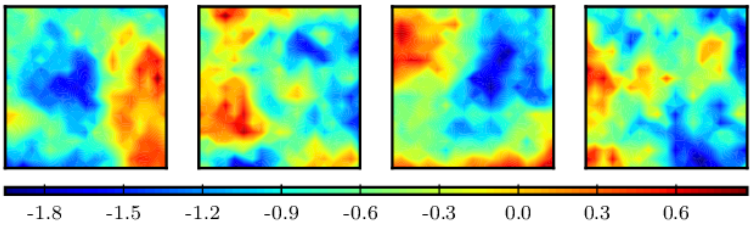
(C) Predicted $\hat{\phi}$



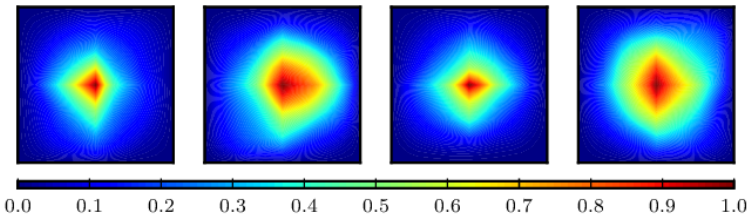
(D) Difference $\phi - \hat{\phi}$

FIGURE 2.6: Performance of basis function predictor, case $L = 0.1$

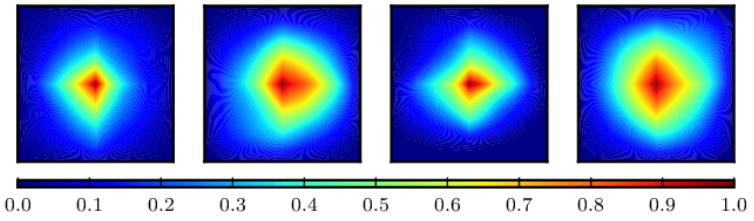
where $\|\cdot\|$ denotes the L^2 norm, f is the trained model, $\hat{\phi}^i = f(\log \kappa^i),\ i = 1, 2, \cdots, N_{val}$ are the *predicted* basis functions, $\phi^1, \phi^2, \cdots, \phi^{N_{val}}$ are the *true* basis functions, and $\bar{\phi} = \frac{1}{N_{val}} \sum \phi^i$ is the sample mean of the true basis functions. A score of 1.0 corresponds to perfect prediction, while a score below 0 means that the predictor performance is worse than a model that always predicts the sample mean. Table 2.1 shows the validation scores obtained on the three permeability types considered, i.e. correlation lengths $L = 0.1$, 0.2 and 0.4. Figures 2.6, 2.7 and 2.8 show some of the predicted basis functions for cases $L = 0.1$, 0.2 and 0.4, respectively. We see that the prediction is more challenging for the case of shortest correlation length. This is likely due to the permeability field being more heterogeneous for shorter correlation lengths.



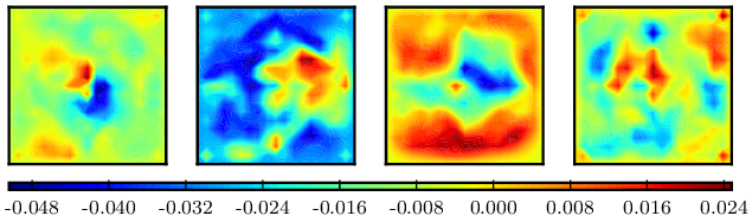
(A) Input $\log \kappa$



(B) Target ϕ



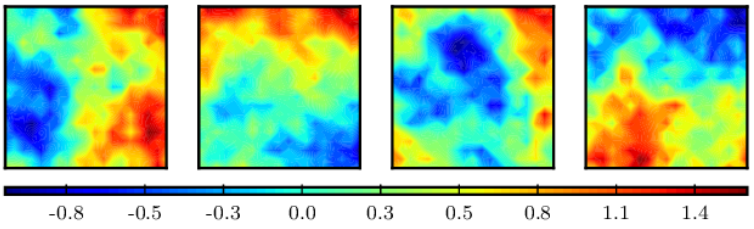
(C) Predicted $\hat{\phi}$



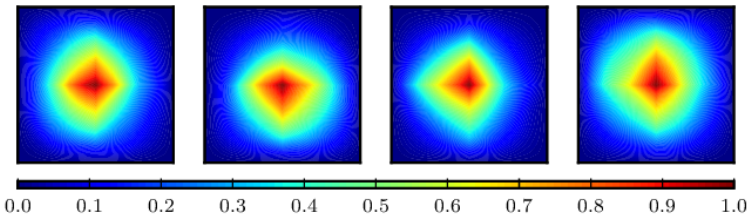
(D) Difference $\phi - \hat{\phi}$

FIGURE 2.7: Performance of basis function predictor, case $L = 0.2$

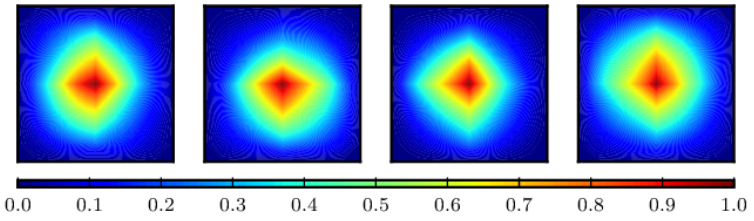
Predictor uncertainty Error estimations of the predicted basis functions might be of interest to fully quantify the uncertainties in the results. Such estimations are readily available in machine learning models such as Gaussian processes. For neural networks, a number of methods such as Bootstrap aggregating (bagging) and others as discussed in [36] could be employed. Another possibility is to employ the dropout technique as a Bayesian approximation method [37]. In our work, we consider the uncertainties in the predicted basis functions to be of second order and the presented numerical results support this assumption.



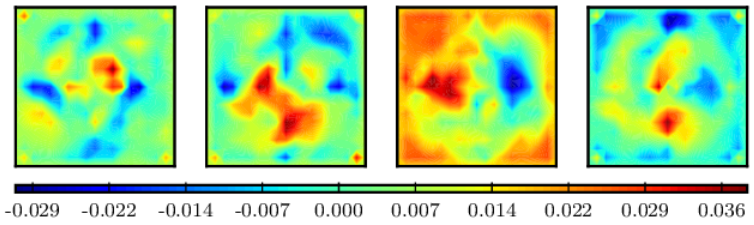
(A) Input $\log \kappa$



(B) Target ϕ



(C) Predicted $\hat{\phi}$



(D) Difference $\phi - \hat{\phi}$

FIGURE 2.8: Performance of basis function predictor, case $L = 0.4$

2.4.2 Hybrid model

Once the neural network model is trained, we can use it to compute the basis functions in the MsFV formulation. To assess the effectiveness of this hybrid approach (MsFV-NN), we consider two test cases:

Quarter-five spot problem: In this problem, injection and production points are located at $(0, 0)$ and $(1, 1)$ of the unit square, respectively. No-flow boundary conditions are imposed. We assume unit injection/production rates, i.e. $q(0, 0) = 1$ and $q(1, 1) = -1$.

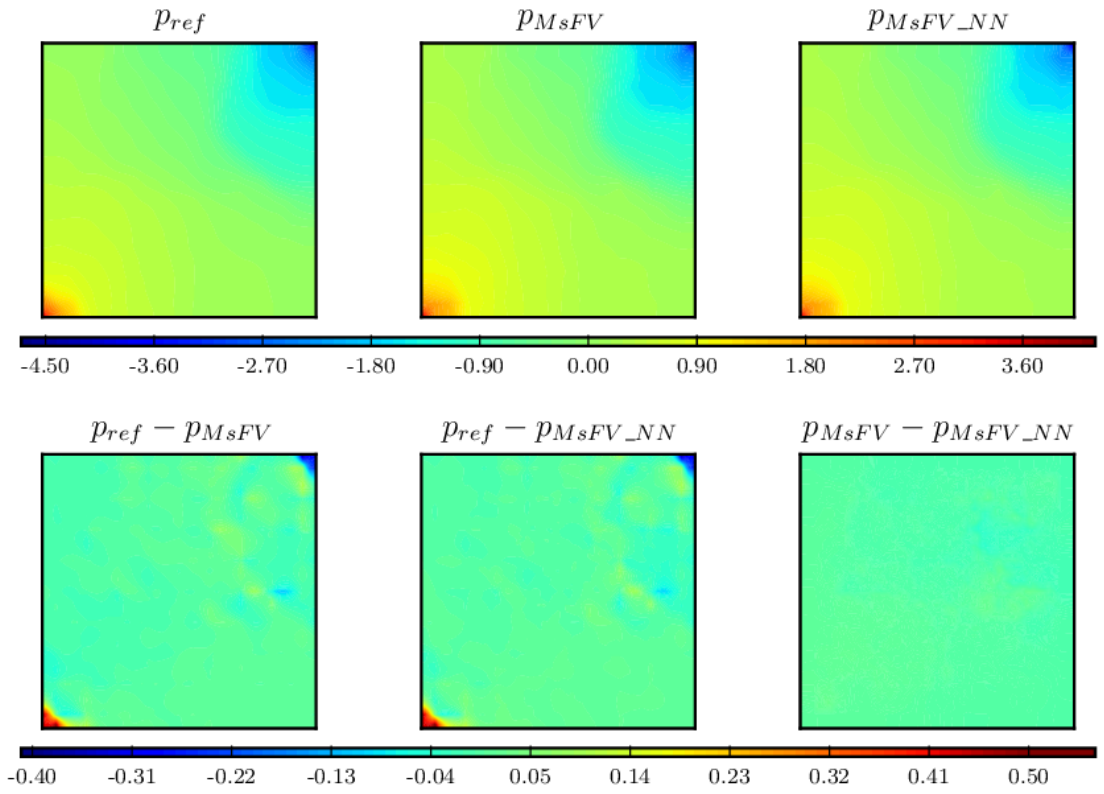
Uniform flow problem: Here, uniformly distributed inflow and outflow conditions are imposed on the left and right sides of the unit square, respectively. No-flow boundary conditions are imposed on the remaining top and bottom sides. A total inflow/outflow rate of 1 is assumed. For the unit square, this means $v \cdot \hat{n} = -1$ and $v \cdot \hat{n} = 1$ on the left and right sides, respectively, where $\hat{n}$ denotes the outward-pointing unit normal to the boundary.

In both cases, a pressure value of 0 is imposed at the center of the square to close the problem. The pressure Eq. (2.1) is solved using three methods: a standard cell-centered finite volume method at the fine-scale level which is taken as the reference "true" solution, the standard MsFV method, and the proposed hybrid method (MsFV-NN). Additionally, we also compute and compare the total velocities, which can be derived from the corresponding pressure solutions. In the reference solution, the total velocity can be derived using Darcy's law ($v = -\lambda \mathbf{K} \nabla p$ where λ is the total mobility, here assumed as $\lambda = 1$), whereas in the MsFV and MsFV-NN methods, the total velocities are derived via a flux reconstruction step, as mentioned before. We take a further step and use the total velocity to solve a tracer flow problem. In this case, we solve the following advection equation:

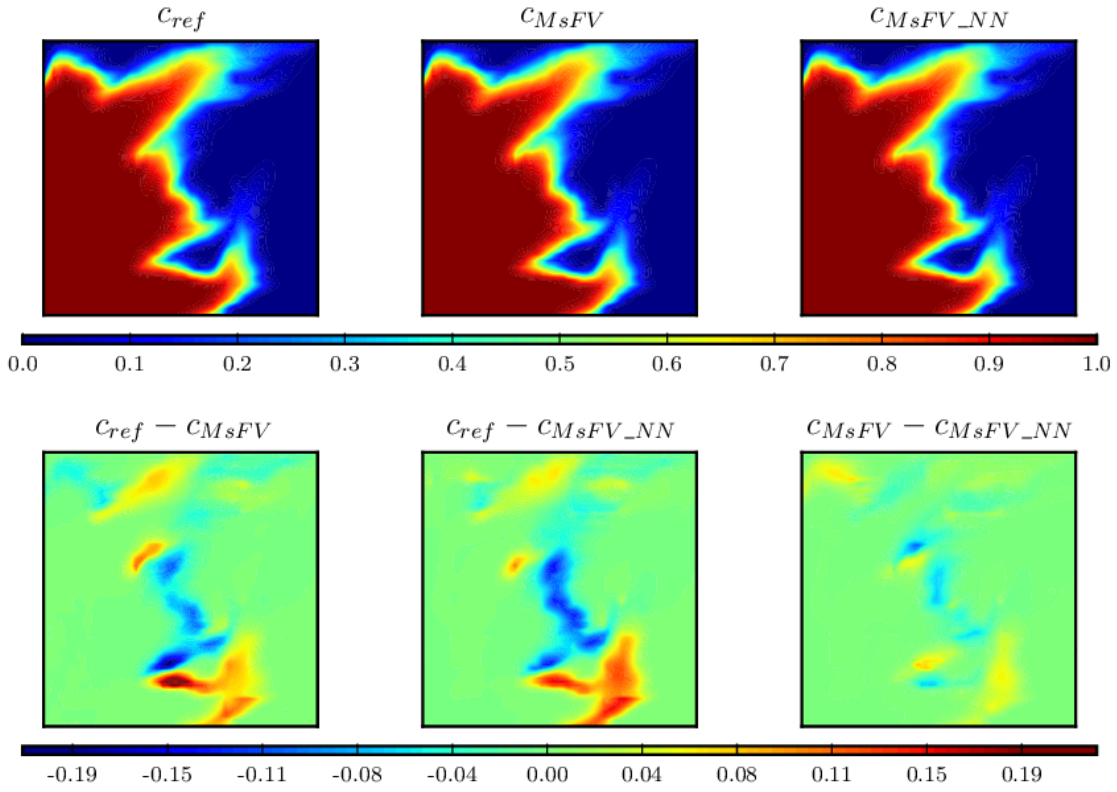
$$\varphi \frac{\partial c}{\partial t} + \nabla \cdot (cv) = \frac{q_w}{\rho_w} \tag{2.9}$$

where c denotes the concentration of the injected fluid (in this case water), φ denotes the domain porosity, q_w denotes sources/sinks of the injected fluid, and ρ_w denotes the density of the injected fluid. In all cases, we assume water with dimensionless density of $\rho_w = 1$ that is injected into a reservoir with constant porosity $\varphi = 0.2$ initially containing only oil, i.e. $c(\mathbf{x}, t = 0) = 0$, which we assume to have the same viscosity as the injected fluid. Under these conditions the total velocity v does not change in time. The simulation time for both test cases is from $t = 0$ until $t = 0.4$. In reservoir engineering, it is more convenient to work with pore volume injected (PVI) as the time unit, which expresses the ratio of the total volume of fluid injected until time t and the reservoir pore volume (for constant injection, $t_{PVI} = q_{in} t / V_\varphi$ where V_φ is the pore volume).

Figures 2.9 and 2.10 show sample solutions for one realization of correlation length $L = 0.1$, for the two test cases considered. We also show the contour plot of the difference between the reference and MsFV, the reference and MsFV-NN, and MsFV and MsFV-NN.

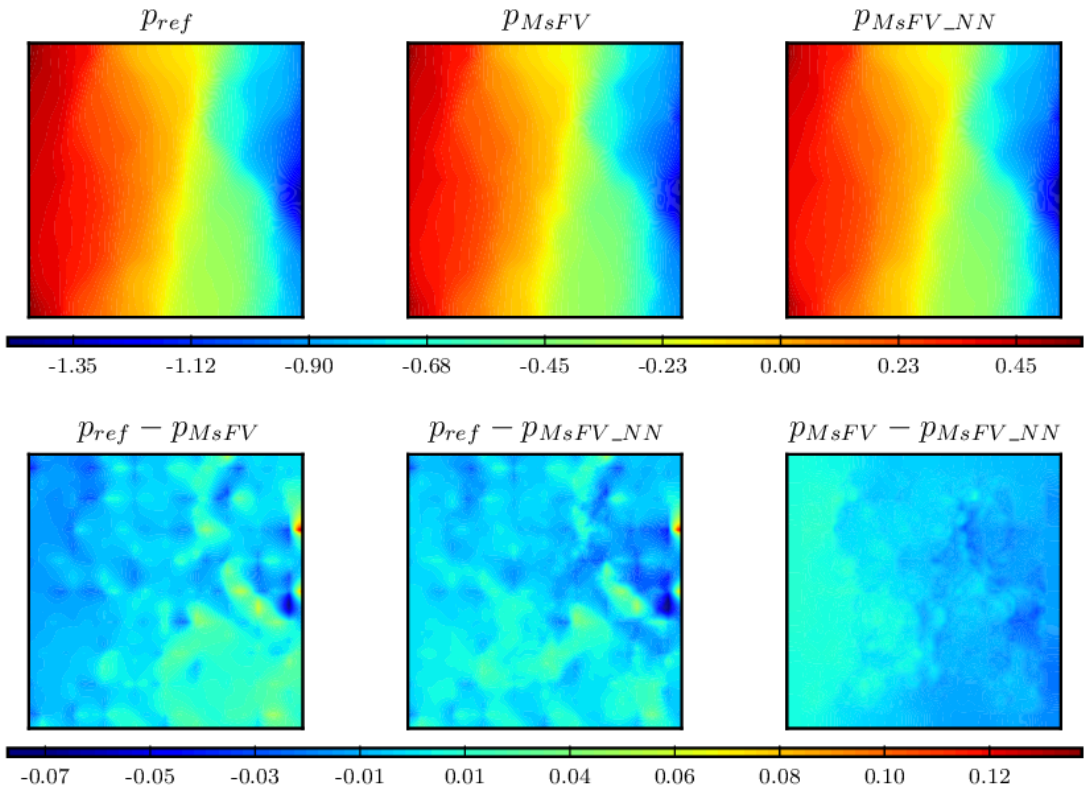


(A) Pressure solution for one realization.

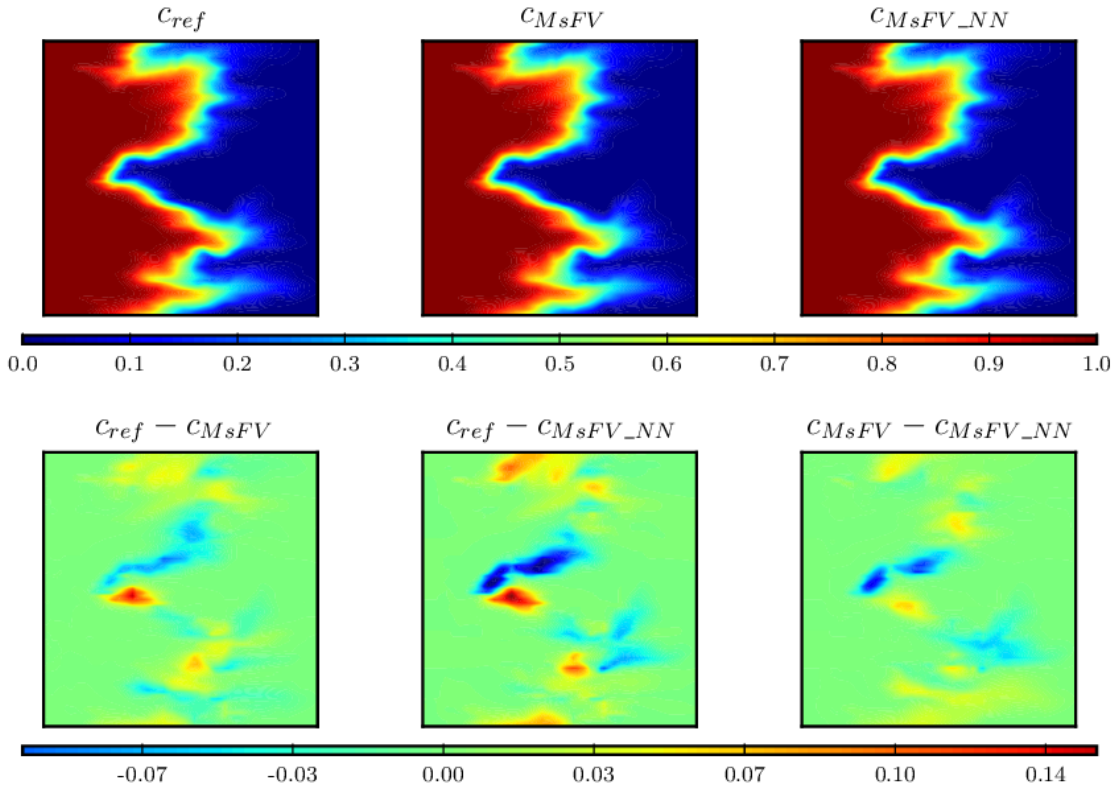


(B) Concentration solution at $t = 0.5$ PVI for one realization.

FIGURE 2.9: Quarter-five spot problem: Sample solution for one realization based on the reference (standard FVM), MsFV and MsFV-NN.

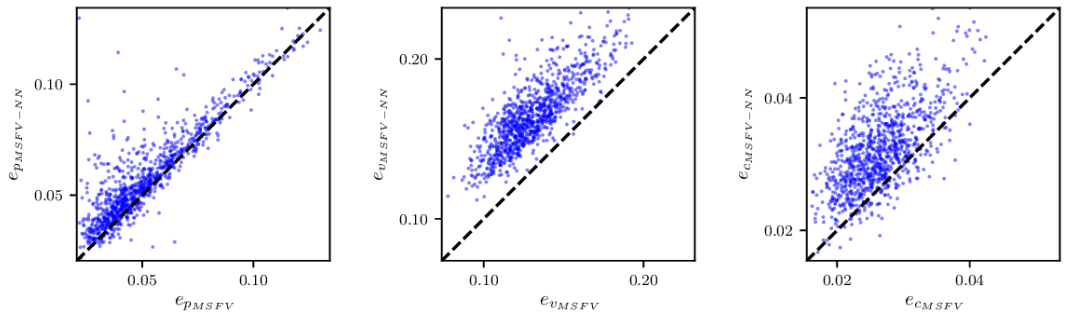


(A) Pressure solution for one realization.

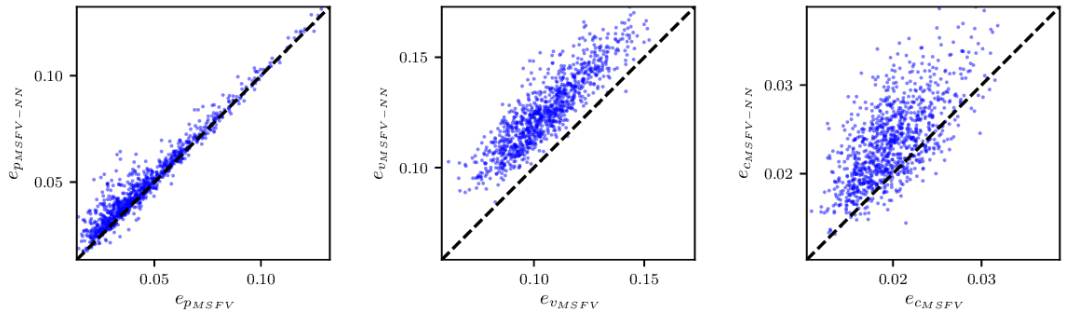


(B) Concentration solution at $t = 0.5$ PVI for one realization.

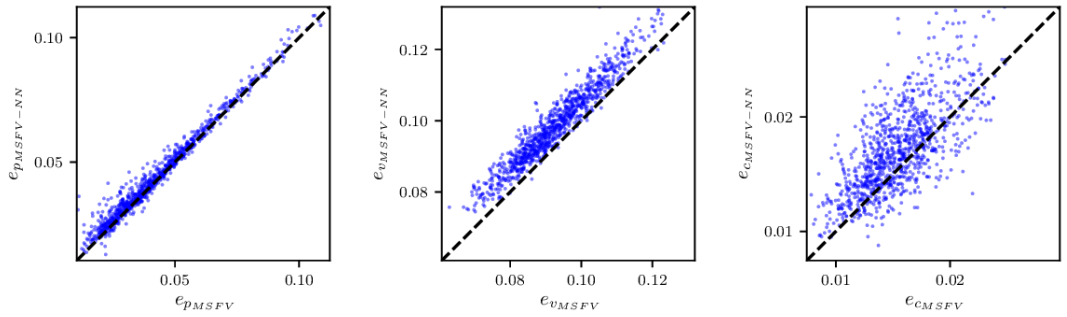
FIGURE 2.10: Uniform flow problem: Sample solution for one realization based on the reference (standard FVM), MsFV and MsFV-NN.



(A) Case $L = 0.1$



(B) Case $L = 0.2$

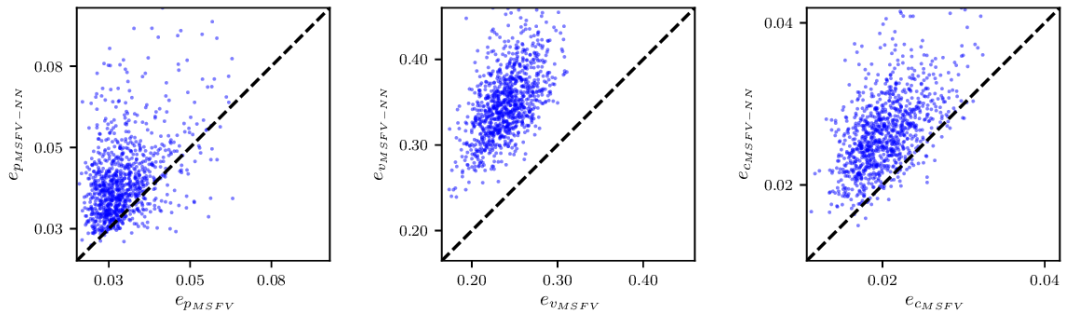


(C) Case $L = 0.4$

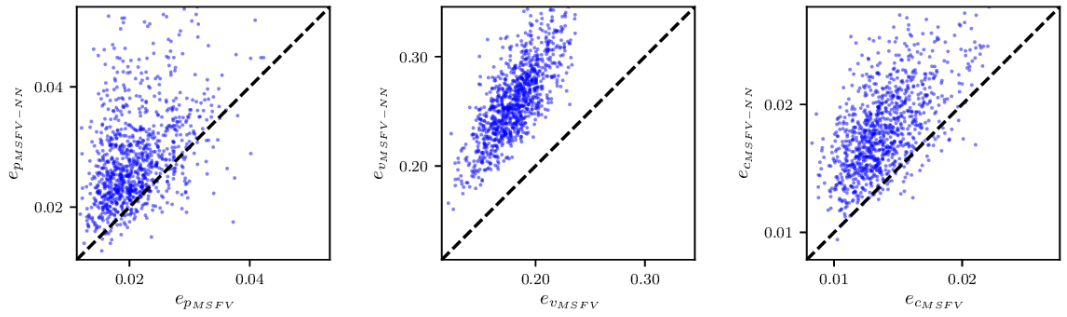
FIGURE 2.11: Quarter five spot problem: Comparison of errors in MsFV and MsFV-NN.

2.4.2.1 Comparison of errors

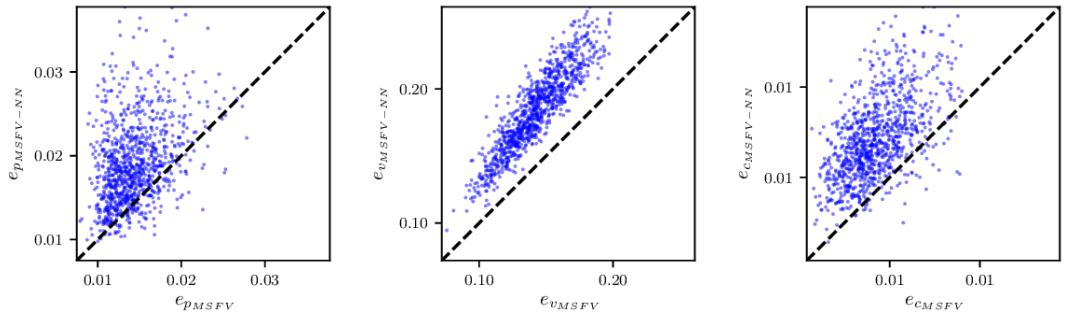
The errors of the solutions (pressure, velocity, concentration) of MsFV and MsFV-NN are measured with respect to the reference solution using an area weighted norm. Let $\mathbf{u} = (u_1, \cdots, u_n)$ be a vector of values corresponding to cells $\Omega_1, \cdots, \Omega_n$ and let $|\Omega_i|$ be the area of cell i, we define the area weighted norm as $\|\mathbf{u}\| = (\sum_i |u_i|^2 |\Omega_i|)^{1/2}$. Using this notation, the pressure error (e_p), the velocity error (e_v), and the concentration error



(A) Case $L = 0.1$



(B) Case $L = 0.2$



(C) Case $L = 0.4$

FIGURE 2.12: Uniform flow problem: Comparison of errors in MsFV and MsFV-NN.

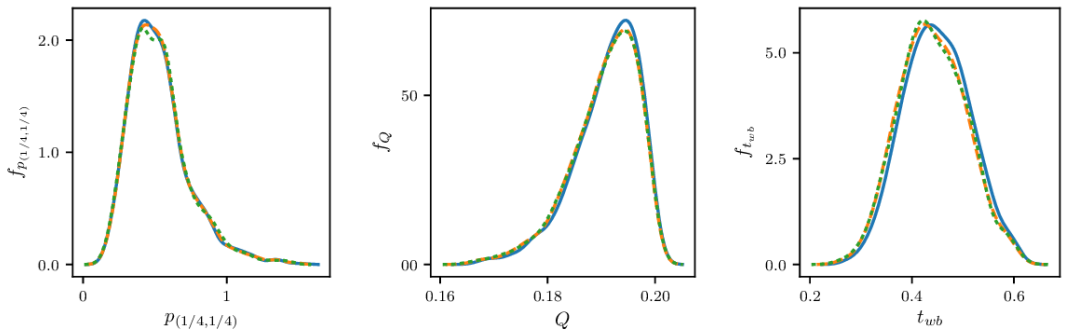
(e_c) are:

$$e_p = \frac{\|\mathbf{p}^{ref} - \mathbf{p}\|}{\|\mathbf{p}^{ref}\|} \tag{2.10}$$

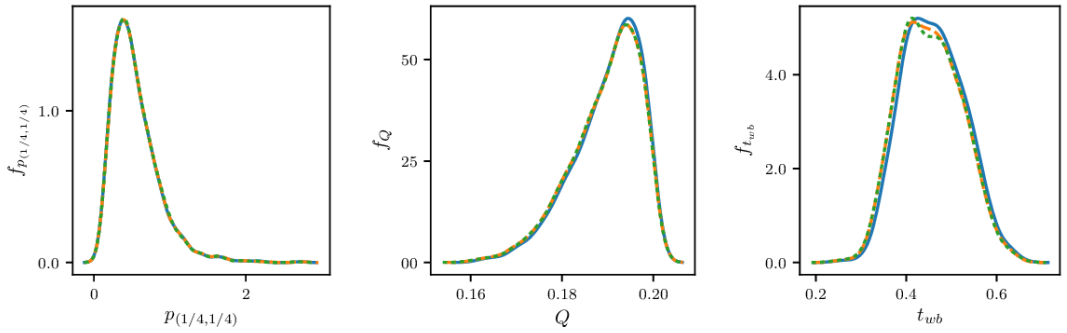
$$e_v = \frac{\|\mathbf{v}_x^{ref} - \mathbf{v}_x\|}{\|\mathbf{v}_x^{ref}\|} + \frac{\|\mathbf{v}_y^{ref} - \mathbf{v}_y\|}{\|\mathbf{v}_y^{ref}\|} \tag{2.11}$$

$$e_c = \frac{1}{T}\int_0^T \frac{\|\mathbf{c}^{ref}(\cdot,t) - \mathbf{c}(\cdot,t)\|}{\|\mathbf{c}^{ref}(\cdot,t)\|} dt \tag{2.12}$$

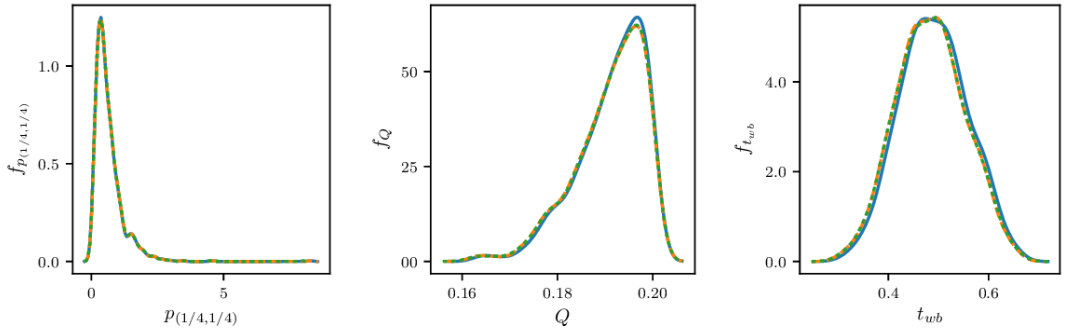
Figures 2.11 and 2.12 show scatter plots of the errors obtained by the MsFV and MsFV-NN. As expected, a better predictor performance (in terms of the R^2-score) corresponded to a better correlation between both errors.



(A) Case $L = 0.1$



(B) Case $L = 0.2$



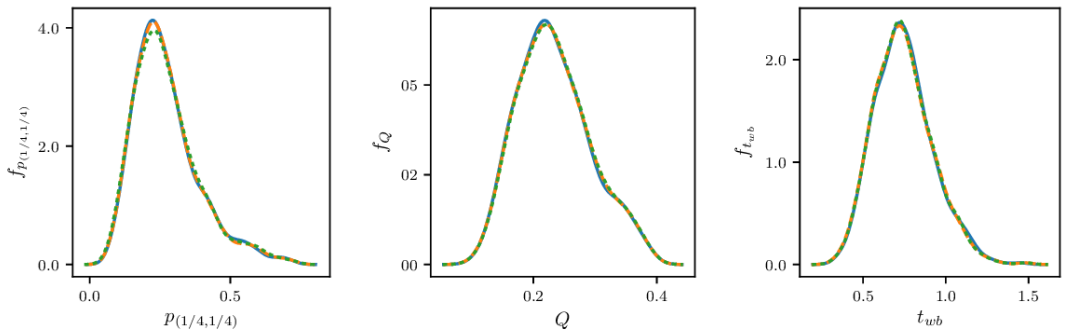
(C) Case $L = 0.4$

FIGURE 2.13: Quarter five spot problem: Estimated distributions by MsFV (orange dashed line), MsFV-NN (green dotted line), and reference (blue line).

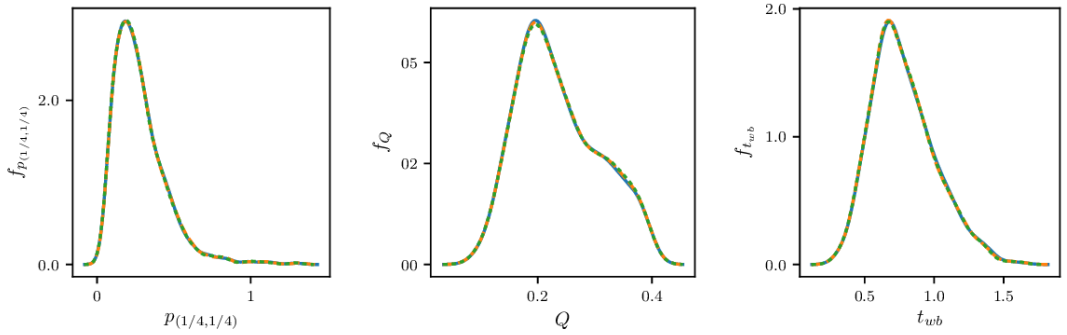
2.4.2.2 Estimated distributions

Finally, we compare all three methods in an uncertainty quantification task where we estimate the pressure p at $(1/4, 1/4)$, the total production Q, and the water breakthrough time t_{wb} (time when water fraction reaches 1% at the production well).

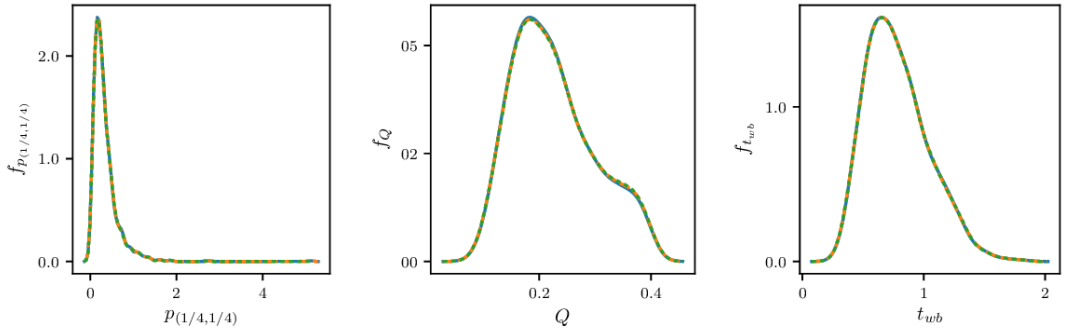
Figures 2.13 and 2.14 show the estimated distributions according to each method. We can see that the distributions given by MsFV and MsFV-NN are almost indistinguishable even for the less accurate predictor ($L = 0.1$). From these results, it is clear that the effectiveness of the hybrid model is attached to the effectiveness of the target model



(A) Case $L = 0.1$



(B) Case $L = 0.2$



(C) Case $L = 0.4$

FIGURE 2.14: Uniform flow problem: Estimated distributions by MsFV (orange dashed line), MsFV-NN (green dotted line), and reference (blue line).

TABLE 2.2: Results of hyperparameter tuning.

Dropout rate	5.4%
Learning rate	3.1×10^{-3}
R^2-score	0.97

(MsFV). The hybrid model is expected to perform well as long as the target model itself serves as a good proxy to the "true" solution.

2.4.3 Hyperparameter tuning

In this section, we show how to further improve the learning performance by fine-tuning the model with a hyperparameter optimization algorithm. Specifically, we employ the Tree-Parzen Estimator algorithm. We shall consider the case of $L = 0.1$ where the trained predictor performed with a score of 0.927.

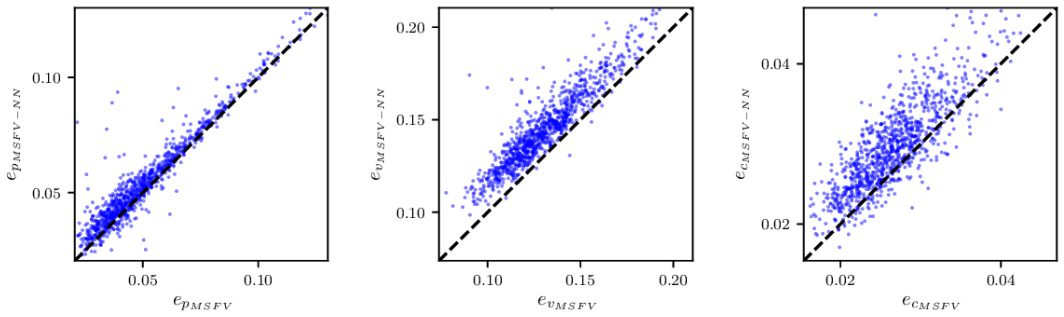
Previously, a dropout rate of 5% and a default learning rate of 10^{-3} have been fixed. Here we let the hyperparameter optimization algorithm tune the values for the dropout rate and the learning rate. Moreover, we also employ batch normalization to enhance the optimization process.

Table 2.2 summarizes the hyperparameter optimization results under a budget of 20 iterations where we observe significant improvement in the R^2-score. Figure 2.15 shows the error scatter plot for the resulting hybrid model. An improvement in the correlation is observed (please compare with Figures 2.11 and 2.12). Figure 2.16 compares the estimated distributions for the quantities of interest where a strong agreement between the data-driven approach and the MsFV is observed. Finally, Table 2.3 presents summary statistics of the results obtained. Overall, we see that there are improvements in both the errors and the estimated distributions when the learning performance increases, as is expected. Of course, even more improvements can be achieved by further tuning the model, for example by increasing the number of iterations of the hyperparameter optimization, or by employing additional tools such as L^1 and L^2 regularizers.

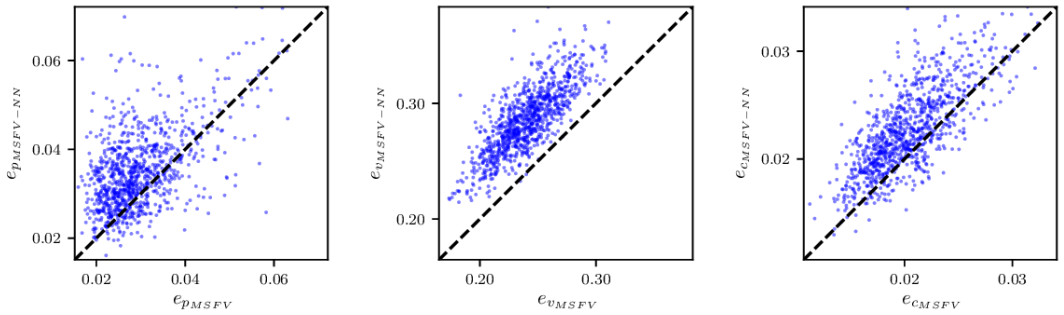
2.4.4 Computational gains

For an estimate of the computational speedup provided by the proposed method, we compared the time taken to generate 1000 basis functions using the predictor vs. the standard approach of solving local problems. Since the MsFV method obtains the basis functions by solving local problems which involve many intermediary steps, and this could lead to data-derived overheads which are implementation-dependent, we decided to measure the time of solving the four local *2D problems only*, i.e. without accounting for the overheads of getting the local boundary conditions (which are obtained by solving the 1D problems) and assembling the local matrices. We employed two solvers for the local problems: GMRES iterative solver and UMFPACK direct solver, both highly optimized C-compiled packages provided in `numpy/scipy`.

Table 2.4 summarises the run times obtained. These results were obtained using one thread (except for the last row which is run on GPU). Here, "batch eval" refers to the prediction of the N basis functions "at once": for a given input vector $\kappa_i, i =$

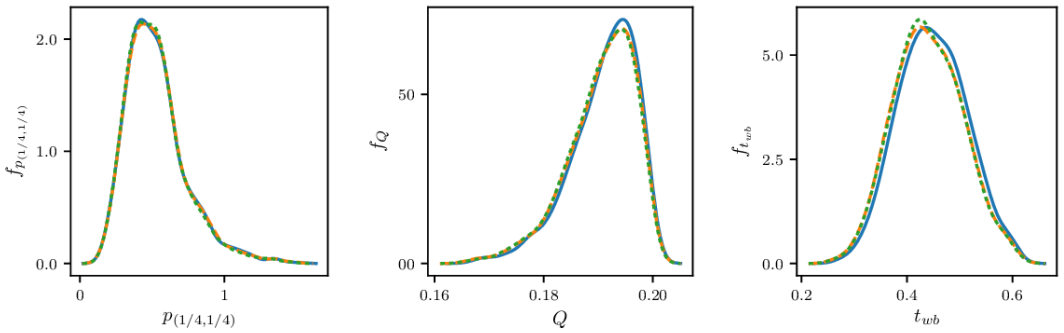


(A) Quarter five spot problem ($L = 0.1$)

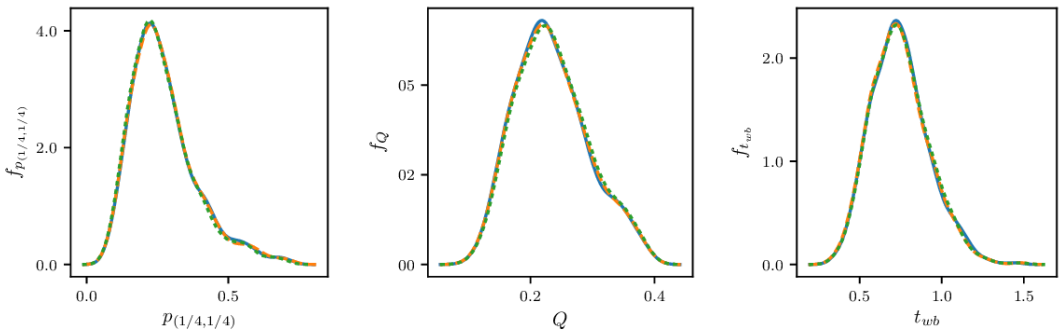


(B) Uniform flow problem ($L = 0.1$)

FIGURE 2.15: Comparison of errors in MsFV and MsFV-NN (tuned model, $L = 0.1$). Improvements can be seen with respect to Figures 2.11(a) and 2.12(a).



(A) Quarter five spot problem ($L = 0.1$)



(B) Uniform flow problem ($L = 0.1$)

FIGURE 2.16: Estimated distributions (tuned model, $L = 0.1$) by MsFV (orange dashed line), MsFV-NN (green dotted line), and reference (blue line). Compare with Figures 2.13(a) and 2.14(a).

TABLE 2.3: Summary statistics and point estimates ($L = 0.1$).

(A) Quarter five spot problem

	Reference	MsFV	MsFV-NN	Untuned
$\bar{e}_p$	-	0.0525	0.0560	0.0586
s_{e_p}	-	0.0228	0.0226	0.0232
$\bar{e}_v$	-	0.1312	0.1463	0.1654
s_{e_v}	-	0.0216	0.0243	0.0231
$\bar{e}_c$	-	0.0268	0.0298	0.0326
s_{e_c}	-	0.0053	0.0068	0.0058
$\bar{p}_{(1/4,1/4)}$	0.5283	0.5297	0.5303	0.5240
$s_{p_{(1/4,1/4)}}$	0.2075	0.2081	0.2092	0.2036
$\bar{Q}$	0.1910	0.1906	0.1906	0.1904
s_Q	0.0060	0.0062	0.0063	0.0062
$\bar{t}_{wb}$	0.4488	0.4404	0.4406	0.4402
$s_{t_{wb}}$	0.0639	0.0646	0.0650	0.0639

(B) Uniform flow problem

	Reference	MsFV	MsFV-NN	Untuned
$\bar{e}_p$	-	0.0301	0.0357	0.0408
s_{e_p}	-	0.0093	0.0105	0.0141
$\bar{e}_v$	-	0.2412	0.2876	0.3504
s_{e_v}	-	0.0267	0.0338	0.0420
$\bar{e}_c$	-	0.0205	0.0228	0.0267
s_{e_c}	-	0.0039	0.0043	0.0050
$\bar{p}_{(1/4,1/4)}$	0.2724	0.2722	0.2732	0.2677
$s_{p_{(1/4,1/4)}}$	0.1160	0.1162	0.1181	0.1135
$\bar{Q}$	0.2339	0.2345	0.2352	0.2365
s_Q	0.0589	0.059	0.0590	0.0591
$\bar{t}_{wb}$	0.7460	0.7423	0.7436	0.7479
$s_{t_{wb}}$	0.1746	0.1743	0.1738	0.1737

TABLE 2.4: Time to generate 1000 basis functions using different methods.

Method	Time [sec]
Sparse direct solver (UMFPACK)	2.14
GMRES (`tol=1e-5`)	7.21
GMRES (`tol=1e-16`)	23.77
NN prediction	0.389
NN (batch eval)	0.083
NN (batch eval) (GPU)	0.017

$1, 2, ..., N$, the predictor performs a matrix-vector multiplication on κ_i. But this can be implemented as a matrix-matrix multiplication simply by building the matrix $\mathcal{K}$ whose columns are the vectors κ_i, allowing for additional numerical optimizations.

In this case, we see that the direct solver outperformed the iterative solver for the local problems since the local matrices are small, which is the common scenario in multiscale methods. We also see that the data-driven approach clearly outperforms the solver component, and if we add the overheads of solving the 1D problems plus the local matrix assembly, the computational advantage will be amplified. We note however that these times can vary depending on the implementation. In particular, different neural network architectures and different solvers for the system of equations may yield different times. Nevertheless, it is unlikely that solving the local problems will outperform a forward pass of a neural network, i.e. direct matrix-vector computations.

2.5 Conclusions and remarks

We have seen that for the presented subsurface flow problems, shallow neural networks performed very well as a simple surrogate for the computation of basis functions in the multiscale finite volume method. Further, we draw the following remarks:

- Results obtained for uncertainty propagation using MsFV and the proposed MsFV-NN method were practically indistinguishable.
- The proposed method is applicable to any multiscale method where the sub-grid scales are captured numerically by solving local problems.
- The proposed method is scalable with large coarse partitions (since more data samples are obtained per simulation run).

In addition, we note that if the *data distribution* remains unchanged (or is similar to that of the training data), then the same trained predictor can be used for different problem conditions (for example, to perform well location optimization), and further computational gains can be achieved since we avoid training a new predictor. This is the situation in cases such as steady state flow or tracer flow.

We have presented the first application of machine learning to capture sub-grid scale heterogeneities within a multiscale method. As our next step, we aim to study the application of the presented method for multiphase flow in porous media. Other possible research directions include extensions to more general permeability fields with anisotropy and channelized structures.

Chapter 3

Parametrization of stochastic inputs using generative adversarial networks with application in geology

We investigate generative adversarial networks as a tool for sample-based parametrization of stochastic inputs in numerical simulations. We address parametrization from the point of view of emulating the data generating process, instead of explicitly constructing a parametric form to preserve statistics of the data. By emulating the data generating process, we replicate the statistics of the data. This is done by training a neural network to generate samples that follow the data distribution using a recent technique called *generative adversarial networks*. The method is assessed in subsurface flow problems, where the effective parametrization of underground properties is important due to the high dimensionality and presence of high spatial correlations. We experiment with unconditional and conditional realizations of binary channelized geological models, and perform uncertainty quantification and parameter estimation. Results show that the parametrization using generative adversarial networks is very effective in preserving visual realism as well as high order statistics of the flow responses, while achieving a dimensionality reduction of two orders of magnitude.

In preparation for submission to journal.

3.1 Introduction

Many problem scenarios such as uncertainty quantification and parameter estimation involve the solution of partial differential equations with a stochastic input. Input in this context is understood as any system property that affects the system response, e.g. the conductivity tensor in the heat equation. This is because in many real applications, some properties of the system are uncertain or simply unknown. The general approach is to set a probabilistic framework where we represent such uncertainties as random variables with a distribution predefined using domain knowledge. In some cases where both the distribution and the forward map are trivial, a closed-form solution could be obtained; however this is very rarely the case. Often in practice, we can only resort to a brute-force approach where we draw several realizations of the random variables and fully solve the partial differential equations for each realization in an effort to estimate distributions or bounds of the system's response. This approach suffers from slow convergence and the need to perform a large number of forward simulations, which led to the development of several methods to reduce the computational burden of this task.

A straightforward solution is to reduce the computational cost of the forward map itself – a large number of methods have been developed in this direction. Another different direction is focused on reducing or refining the search space or distribution of the random variables, for example by regularization or parametrization, thus reducing the number of simulations required. Parametrization is specially useful in problems where the number of random variables is huge but the variables are highly redundant and correlated. This is generally the case in subsurface flow problems: Complete prior knowledge of subsurface properties (e.g. porosity or permeability) is impossible, yet is very influential in the flow responses. At the same time, accurate flow modeling often requires the use of extremely large simulation grids. When the subsurface property is discretized, the number of free variables is naively associated with the number of grid cells. The random variables thus obtained are hardly independent, whose assumption during the modeling leads to unnecessary computations over unrealistic realizations. The goal of parametrization is to discover statistical relationships between the random variables in order to obtain a reduced and more effective representation.

The importance of parametrization in subsurface simulations resulted in a variety of methods in the literature including zonation [38, 39] and zonation-based methods [40–45], PCA-based methods [46–51], SVD-based methods [52–55], discrete wavelet transform [56–58], discrete cosine transform [59–61], level set methods [62–64], and dictionary learning [65, 66]. Many current methods begin by proposing parametric forms for the random vector to be modeled which are then explicitly fitted to preserve certain statistics

of the random vector. Many methods inevitably adopt some oversimplifying assumptions during the modeling process, either on the parametric form to be employed or the statistics to be reproduced, which are often necessary for the method to be actually feasible. In this work, we consider the use of neural networks for *both* parametrization of the random vector *and* definition of its relevant statistics. This is motivated by recent advances in the field of machine learning, as well as the high expressive power of neural networks that makes them one of the least constraining forms of parametrization and very suitable to model complex data.

The idea is to view parametrization as emulating the data generating process itself – by emulating the data generating process, we replicate the statistics of the random vector. We seek to construct a deterministic function called the generator – in this case, a neural network – that takes a low-dimensional vector as input (the reduced representation), and aims to output a realization of the target random vector. The low-dimensional vector is assumed to come from an easy-to-sample distribution, e.g. a multivariate normal or an uniform distribution, and is what provides the element of stochasticity. Generating a new realization then only requires sampling the low-dimensional vector and a forward pass of the generator network. The neural network is trained using a dataset of prior realizations that inform the patterns and variability of the random vector (e.g. geological realizations from a database or from multipoint geostatistical simulations [67, 68]).

The component missing in the description above is the definition of an objective function to actually train such generator; in particular, how do we quantify the discrepancy between generated samples and actual samples? This is resolved using a recent technique in machine learning called *generative adversarial networks* [69] (GAN). The idea in GANs is to let a second classifier neural network, called the discriminator, define the objective function. The discriminator takes the role of an adversary against the generator where both are trained alternately following a minmax game: the discriminator is trained to maximize a classification performance where it needs to discern between "fake" (from the generator) and "real" (from the dataset) samples, while the generator is trained to minimize it. Hence, the generator is iteratively encouraged to generate good realizations in order to fool the discriminator, while the discriminator is in turn iteratively encouraged to improve its ability to classify correctly. Equilibrium of this adversarial game occurs when the generator effectively learns the data distribution, and the discriminator is $\frac{1}{2}$ (coin toss scenario).

The benefit of this approach is that we do not need to manually specify which statistics need to be preserved, instead we let the discriminator network implicitly learn the relevant statistics from data. We can see that the high expressive power of neural networks is leveraged twice: on one hand, the expressive power of neural networks is used in the

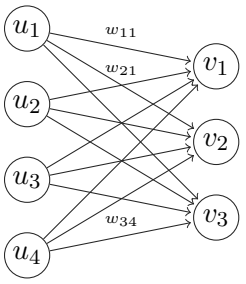
parametrization (generator) to be able to generate complex realizations; on the other hand, it is used in the discriminator to learn the complex high-order statistics of the data.

In this work, we parametrize binary channelized permeability models based on the classical Strebelle training image [67], a benchmark problem often employed due to the difficulty in obtaining a parametrization that preserves the visual realism and spatial and flow statistics. To assess the method, we consider uncertainty propagation in subsurface flow problems for a large number of realizations of the permeability and compare the statistics in their flow responses. We also perform parameter estimation using natural evolution strategies [70, 71], a general black-box optimization method that is suitable for the obtained reduced representation. We further discuss training difficulties of GANs encountered during our implementation such as working with small datasets and inherent issues of the standard formulation of GAN [69].

In the field of geology, the motivation to adopt a sample-based parametrization approach comes from the trend of increasing data availability, in particular realistic geological models, supported by decreasing computational costs, improvements in sensing methods, and the increasing interest in multipoint geostatistical simulations [68, 72, 73]. The interest in the latter stems from the increasing desire to obtain more realistic geological models that captures complex features of the geology, which is often lacking in more traditional methods based on two-point statistics. Multipoint geostatistical simulations can also serve as an unlimited source of realistic geological models to train our neural networks.

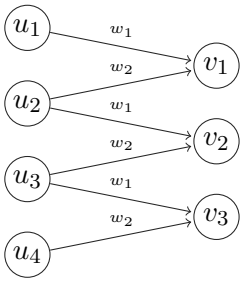
This work is an extension of our preliminary work in [74]. There are a number of recent works in geology-related fields where GANs have been studied. In [75, 76], the authors train a GAN to generate images of porous media for image reconstruction. In [77], the authors train a GAN to generate geological models and apply it for history matching. In [78, 79], the authors study ways to generate conditional realizations using a generator trained on unconditional realizations. In this work, we focus on the capabilities of GANs as a parametrization tool to preserve high order statistics of the flow responses as well as visual realism.

The rest of this chapter is organized as follows: In Section 3.2, we briefly describe convolutional neural networks – an architecture that is widely employed for modern neural networks – and the method of generative adversarial networks. In Section 3.3, we present our numerical results for uncertainty quantification and parameter estimation experiments. In Section 3.4, we provide further discussions for practical implementation. Finally, we draw our conclusions in Section 3.5.



$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{pmatrix}$$

(A) A fully connected layer.



$$\mathbf{W} = \begin{pmatrix} w_1 & w_2 & 0 & 0 \\ 0 & w_1 & w_2 & 0 \\ 0 & 0 & w_1 & w_2 \end{pmatrix}$$

(B) A convolutional layer.

FIGURE 3.1: Transformation matrix of a fully connected layer (a), and of a convolutional layer (b). In this example, the convolutional layer has only 2 free weights, whereas the fully connected layer has 12 free weights.

3.2 Background

3.2.1 Convolutional neural networks

A (feedforward) neural network is a composition of functions $f(\mathbf{x}) = f^L(f^{L-1}(\cdots(f^1(\mathbf{x}))))$ where each function $f^l(\mathbf{x})$, called a layer, is of the form $\sigma_l(\mathbf{W}_l\mathbf{x}+\mathbf{b}_l)$, i.e. a linear transform followed by a component-wise non-linearity. The choice of the number of layers L, the non-linear functions σ_l, and the sizes of $\mathbf{W}_l, \mathbf{b}_l$ are part of the *architecture design* process, which is largely problem-dependent and led by heavy use of heuristics and domain knowledge. Modern architectures use non-linearities such as rectifier linear units (ReLU, $\sigma(x) = x^+ = \max(0, x)$), leaky rectifier linear units (leaky ReLU, $\sigma(x) = x^+ + 0.01x^-$), tanh, sigmoid, and others; and can have as much as 100 layers [80, 81]. After an architecture is assumed, the weights of $\mathbf{W}_l, \mathbf{b}_l$ are optimized following an objective function.

A major architectural choice that led to huge advances in computer vision is the use of *convolutional layers* [82]. An example of a convolution is the following: Let $\mathbf{u} = (u_1, \cdots, u_m)$ be an input vector and $\mathbf{k} = (w_1, w_2)$ be a *filter*. The output of convolving the filter $\mathbf{k}$ on $\mathbf{u}$ is $\mathbf{v} = (v_1, \cdots, v_p)$ where $v_i = w_1u_i + w_2u_{i+1}$ (using stride 1). A benefit of using convolutional layers is that the associated matrix is sparse and with repeated weights, resulting in a huge reduction of the number of free weights for optimization. A second benefit comes from the type of regularization that this operator inherently

imposes that is often useful in applications where there is a spatial or temporal extent and the assumption of data locality is valid, e.g. natural images and speech. Informally, closer points have a higher influence than farther points. A convolution is illustrated in Figure 3.1: In a traditional fully connected layer, the associated matrix is dense and all its weights are to be determined in optimization. In a convolutional layer, the connections are constrained in such a way that each output component only depends on a neighborhood of the input using a same set of weights, and as a result the associated matrix is a sparse diagonal-constant matrix.

The convolution as described above has a contracting effect, i.e. the output size is always smaller or equal to the input size, which can be controlled by the filter stride. Modern classifier networks consist of a series of multiple convolutional layers that successively contract an image to a single number (binary classification) or a vector of numbers (multiclass classification). However, in some cases such as in decoder and generative networks, we wish to achieve the opposite effect to get an output that is larger than the input (e.g. to reconstruct an image given a compressed code). This can be achieved by simply *transposing* the convolutions: Following the example in Figure 3.1b, to convert from $\mathbf{v}$ to $\mathbf{u}$, we can consider weight matrices of the form $\mathbf{W}^\top$, i.e. the transpose of the convolution matrix from $\mathbf{u}$ to $\mathbf{v}$. Modern decoders and generators consist of a series of multiple transposed convolutions that successively upscale a small vector to a large output such as an image or audio.

The operations and properties discussed so far extend naturally to 2D and 3D tensors. For a 2D or 3D input tensor, the filter is also a 2D or 3D tensor, respectively. Note however that in the 3D case, the filter tensor is such that the depth (orthogonal to the spatial extent) is always equal to the depth of the 3D input tensor, therefore the output is always a 2D tensor, and the striding is done in the spatial extent (width and height). On the other hand, we allow the application of multiple filters to the same input, thereby producing a 3D output tensor if required, consisting of the stack of multiple convolution outputs. This way of operating with convolutions is inherited from image processing: Color images are 3D tensors consisting of three 2D tensors indicating the red, green, and blue intensities (RGB format). Image filtering normally operates on all three values, e.g. the greyscale filter is $v_{ij} = 0.299 \cdot u_{ij,\text{red}} + 0.587 \cdot u_{ij,\text{green}} + 0.114 \cdot u_{ij,\text{blue}}$. The output of a convolution filter is also called a *feature map*.

Finally, we show in Figure 3.2 a popular pyramid architecture used in generator networks [83] for image synthesis. The blocks shown represent the state shapes (stack of feature maps) as the input vector is passed through the network. The input vector z is first treated as a "1-pixel image" (with many "feature maps"). The blocks are subsequently expanded in the spatial extent (width and height) while thinned in depth: A

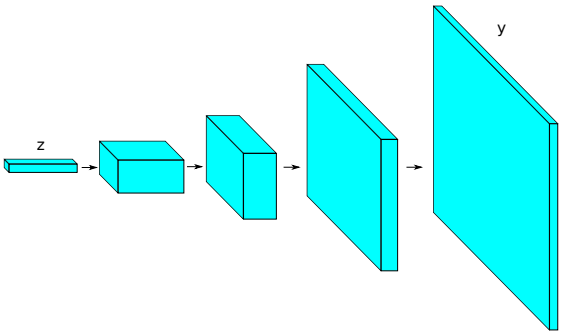


FIGURE 3.2: Illustration of a typical pyramid architecture used in generator networks.

series of transposed convolutions is used to upsample the spatial extent until reaching the desired size; at the same time, the number of convolution filters is initially large, but it is subsequently reduced in the following layers. For classifier networks, usually the inverted architecture is used where the transposed convolutions are replaced with normal convolutions. Further notes on modern convolutional neural networks can be found in [84].

3.2.2 Generative adversarial networks

Let $\mathbf{z} \sim p_z$, $\mathbf{y} \sim \mathbb{P}_y$, where p_z is a known, easy-to-sample distribution (e.g. multivariate normal or uniform distribution), and $\mathbb{P}_y$ is the unknown distribution of interest (e.g. the distribution of all possible geomodels in a particular zone). The distribution $\mathbb{P}_y$ is only known through realizations $\{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$ (e.g. realizations provided by multipoint geostatistical simulations). Let $G_\theta \colon \mathcal{Z} \to \mathcal{Y}$ be a neural network – called the generator – parametrized by weights θ to be determined. Given p_z fixed, this neural network induces a distribution $G_\theta(\mathbf{z}) \sim \mathbb{P}_\theta$ that depends on θ, and whose explicit form is complicated or intractable (since neural networks contain multiple non-linearities). On the other hand, sampling from this distribution is easy since it only involves sampling $\mathbf{z}$ and a forward evaluation of G_θ. The goal is to find θ such that $\mathbb{P}_\theta = \mathbb{P}_y$.

Generative adversarial networks (GAN) [69] approach this problem by considering a second classifier neural network –called the discriminator– to classify between "fake" samples (generated by the generator) and "real" samples (coming from the dataset of realizations). Let $D_\psi \colon \mathcal{Y} \to [0, 1]$ be the discriminator network parametrized by weights ψ to be determined. The training of the generator and discriminator uses the following loss function:

$$\mathcal{L}(\psi, \theta) \coloneqq \mathop{\mathbb{E}}_{\mathbf{y} \sim \mathbb{P}_y} \log D_\psi(\mathbf{y}) + \mathop{\mathbb{E}}_{\tilde{\mathbf{y}} \sim \mathbb{P}_\theta} \log(1 - D_\psi(\tilde{\mathbf{y}})) \tag{3.1}$$

where $\tilde{\mathbf{y}} = G_\theta(\mathbf{z}) \sim \mathbb{P}_\theta$. In effect, this loss is the classification score of the discriminator, therefore we train D_ψ to maximize this function, and G_θ to minimize it:

$$\min_\theta \max_\psi \mathcal{L}(\psi, \theta) \tag{3.2}$$

In practice, optimization of this minmax game is done alternately using some variant of stochastic gradient descent, where the gradient can be obtained using automatic differentiation algorithms. It is shown in [69] that in the infinite capacity setting, optimization of this minmax game amounts to minimizing the Jensen-Shannon divergence between $\mathbb{P}_y$ and $\mathbb{P}_\theta$. Equilibrium of the game occurs when $\mathbb{P}_y = \mathbb{P}_\theta$ and $D_\psi = \frac{1}{2}$ in the support of $\mathbb{P}_y$ (coin toss scenario).

3.2.2.1 Wasserstein GAN

In practice, optimization of the minmax game (3.2) is known to be very unstable, prompting numerous works to understand and address this issue [83, 85–91]. Of the many works, we find that the Wasserstein formulation of GAN (WGAN) [90, 91] is well-suited for our application. This formulation proposes the objective function

$$\mathcal{L}(\psi, \theta) \coloneqq \mathop{\mathbb{E}}_{\mathbf{y} \sim \mathbb{P}_y} D_\psi(\mathbf{y}) - \mathop{\mathbb{E}}_{\tilde{\mathbf{y}} \sim \mathbb{P}_\theta} D_\psi(\tilde{\mathbf{y}}) \tag{3.3}$$

and a constraint in the search space of D_ψ,

$$\min_\theta \max_{\psi : D_\psi \in \mathcal{D}} \mathcal{L}(\psi, \theta) \tag{3.4}$$

where now $D_\psi \colon \mathcal{Y} \to \mathbb{R}$ and $\mathcal{D}$ is the set of 1-Lipschitz functions. This constraint can be loosely enforced by constraining the weights ψ to a compact space, e.g. by clipping the values of the weights in an interval $[-c, c]$. In practice, $\mathcal{D}$ is a set of k-Lipschitz functions for a constant k that is irrelevant for optimization. Although the modifications in Equations (3.3) and (3.4) over Equations (3.1) and (3.2) seem trivial, the derivation of this formulation is rather involved and can be found in [90]. In essence, this formulation aims to minimize the *Wasserstein distance* between two distributions, instead of the Jensen-Shannon divergence. Here we only highlight important consequences of this formulation:

- Access to a meaningful loss metric. This is because

$$W(\mathbb{P}_y, \mathbb{P}_\theta) \approx \max_{\psi : D_\psi \in \mathcal{D}} \mathcal{L}(\psi, \theta) \tag{3.5}$$

where W denotes the Wasserstein distance.

- Better stability. In particular, mode collapse is drastically reduced (see Section 3.4.1).
- Robustness to architectural choices and optimization parameters.

We experimentally verify these points in Section 3.4.1 and discuss their implications for our current application.

A pseudo-code of the training process is shown in Algorithm 1. Note that D is trained multiple times (n_D) per each iteration of G. This is to keep D near optimality so that the Wasserstein estimate in Equation (3.5) is accurate before every update of G. We also note that even though we show a simple gradient ascent/descent in the update steps (lines 6 and 11), it is more common to use update schemes such as RMSProp [27] and Adam [28] that are better suited for neural network optimization.

Algorithm 1 The WGAN algorithm

Require: n_D iterations of D per iteration of G, initial guesses $\theta_{\text{init}}, \psi_{\text{init}}$, step size η, batch size m, clipping interval c.

1: **while** θ has not converged **do**

▷ Train D

2: **for** $t = 1, ..., n_D$ **do**

3: Sample $\{\mathbf{z}_1, \cdots, \mathbf{z}_m\} \sim \mathbb{P}_z$ to get $\{\tilde{\mathbf{y}}_1, \cdots, \tilde{\mathbf{y}}_m\}$, $\tilde{\mathbf{y}}_i = G_\theta(\mathbf{z}_i)$

4: Sample $\{\mathbf{y}_1, \cdots, \mathbf{y}_m\} \sim \mathbb{P}_y$ (draw a subset of the dataset)

5: $\nabla_\psi \mathcal{L}(\psi, \theta) \leftarrow \nabla_\psi \left[\frac{1}{m}\sum_{i=1}^m D_\psi(\mathbf{y}_i) - \frac{1}{m}\sum_{i=1}^m D_\psi(\tilde{\mathbf{y}}_i)\right]$

6: $\psi \leftarrow \psi + \eta \nabla_\psi \mathcal{L}(\psi, \theta)$

7: $\psi \leftarrow \text{clip}(\psi, -c, c)$

8: **end for**

▷ Train G

9: Sample $\{\mathbf{z}_1, \cdots, \mathbf{z}_m\} \sim \mathbb{P}_z$ to get $\{\tilde{\mathbf{y}}_1, \cdots, \tilde{\mathbf{y}}_m\}$, $\tilde{\mathbf{y}}_i = G_\theta(\mathbf{z}_i)$

10: $\nabla_\theta \mathcal{L}(\psi, \theta) \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m D_\psi(\tilde{\mathbf{y}}_i)$

11: $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\psi, \theta)$

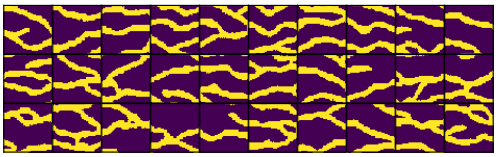
12: **end while**

3.3 Numerical experiments

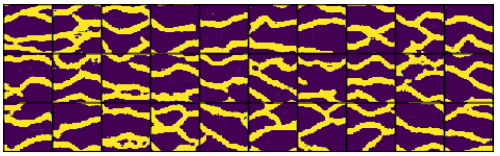
We perform parametrization of unconditional and conditional realizations (conditioned on points over the domain) of a binary channelized permeability using 1000 prior realizations of each. The training image is the classical 250×250 image by Strebelle [67] containing meandering left-to-right channels. The channels have a log-permeability of 1 and the background has a log-permeability of 0. The conditioning is done at 16 points, summarized in Table 3.1, containing 13 points of high permeability (channel material)

	$j = 12$	$j = 25$	$j = 38$	$j = 51$
$i = 12$	1	1	1	1
$i = 25$	1	1	0	0
$i = 38$	1	0	1	1
$i = 51$	1	1	1	1

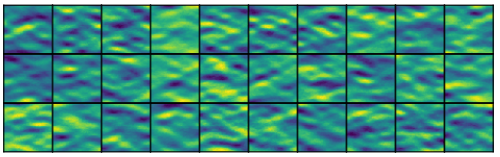
TABLE 3.1: Point conditioning at 16 locations, indicated by cell indices (i, j), regularly distributed across the domain.



Reference

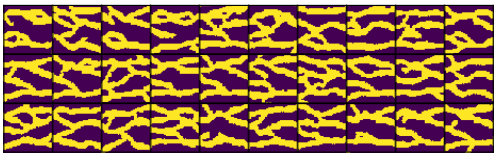


WGAN

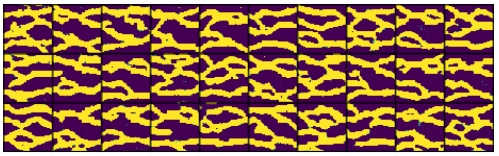


PCA

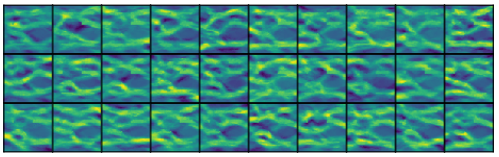
FIGURE 3.3: Unconditional realizations



Reference



WGAN



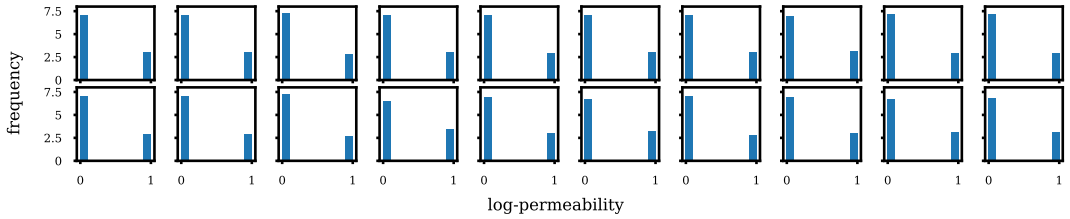
PCA

FIGURE 3.4: Conditional realizations

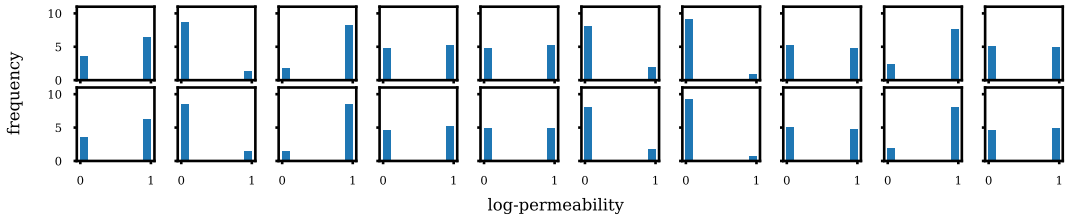
and 3 points of low permeability (background material). The prior realizations are generated using the *snesim* algorithm [67]. The size of the realizations is 64×64.

3.3.1 Implementation

We train separate WGAN models for unconditional and conditional realizations. The same network architecture was used in both cases but trained on their respective prior realizations. The architectures follow the pyramid structure described in Figure 3.2: In the generator, the input tensor is initially upscaled to 4×4 in the spatial extent. The initial number of feature maps is 512. The block is successively upscaled in the spatial extent and reduced in the number of feature maps by a factor of 2, until the spatial extent reaches 32×32. A final transposed convolution upscales the block to 64×64. The non-linearities are ReLUs for all layers except the last layer where we use $\tanh(\cdot)$ (so that output is bounded in $[-1, 1]$). In the discriminator the architecture is inverted, where the initial number of feature maps is 8. The block is successively downscaled in the spatial extent and increased in the number of feature maps by a factor of 2, until



(A) Unconditional case



(B) Conditional case

FIGURE 3.5: Histogram of permeability at 10 random locations based on snesim (first row) and WGAN (second row) realizations.

the spatial extent reaches 4×4. A final convolutional filter reduces this block to a single real value. Note that the size of the discriminator (in terms of total number of weights) is 1/8 times smaller than the generator, which is justified below in Section 3.4.2. All layers except the last use leaky ReLUs. The last layer does not use a non-linearity. We use $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ of dimension 30. This was chosen using principal component analysis as a rule of thumb: to retain 75% of the energy, 54 and 94 eigencomponents are required in the unconditional and conditional cases, respectively. We chose a smaller number to futher investigate the limits of the parametrization. The result is a dimensionality reduction of two orders of magnitude, from $4096 = 64 \times 64$ to 30.

The network is trained using a popular gradient update scheme called Adam, with $\beta_1 = 0.5$, $\beta_2 = 0.999$ (see [28]). We use a step size of 10^{-4}, batch size of 32, and clipping interval $[-0.01, 0.01]$. We perform 5 discriminator iterations per generator iteration. In our experiments, convergence was achieved in around 20,000 generator iterations. The total training time was around 30 minutes using an Nvidia GeForce GTX Titan X GPU. During deployment, the model can generate realizations of 64×64 size at the rate of about 5500 realizations per second.

In Figures 3.3 and 3.4 we show unconditional and conditional realizations generated by our trained models, and realizations generated by snesim (reference). We also show realizations generated with principal component analysis (PCA), retaining 75% of the energy. We see that our model clearly reproduces the visual patterns present in the prior realizations. In Figure 3.5 we show histograms of the permeability at 10 randomly selected locations, based on sets of 5000 fresh realizations generated by snesim (i.e. not

from the prior set) and by WGAN. We find that our model generates values that are very close to either 0 or 1, and almost no value in between (no thresholding has been performed at this stage, only shifting and scaling to move the tanh interval $[-1, 1]$ to $[0, 1]$, i.e. $(x + 1)/2$). The histograms are remarkably close.

3.3.2 Assessment in uncertainty quantification

Our ultimate goal is to achieve a parametrization that preserves not only the visual patterns and spatial statistics but also the flow responses of the prior realizations in flow simulations. In this section, we perform uncertainty quantification and compute flow statistics of interest in practice. We borrow test cases from [50], where the authors parametrize the same type of permeability using Kernel PCA. Thus, we refer the interested reader to such work for results using Kernel PCA. Note that here we use a larger grid (64×64 vs 45×45) and provide an additional flow test case.

We propagate 5000 realizations of the permeability field in 2D single-phase subsurface flow. We consider injection of water for the purpose of displacing oil inside a reservoir (water and oil in this case have the same fluid properties since we consider single-phase flow). The system of equations for this problem is

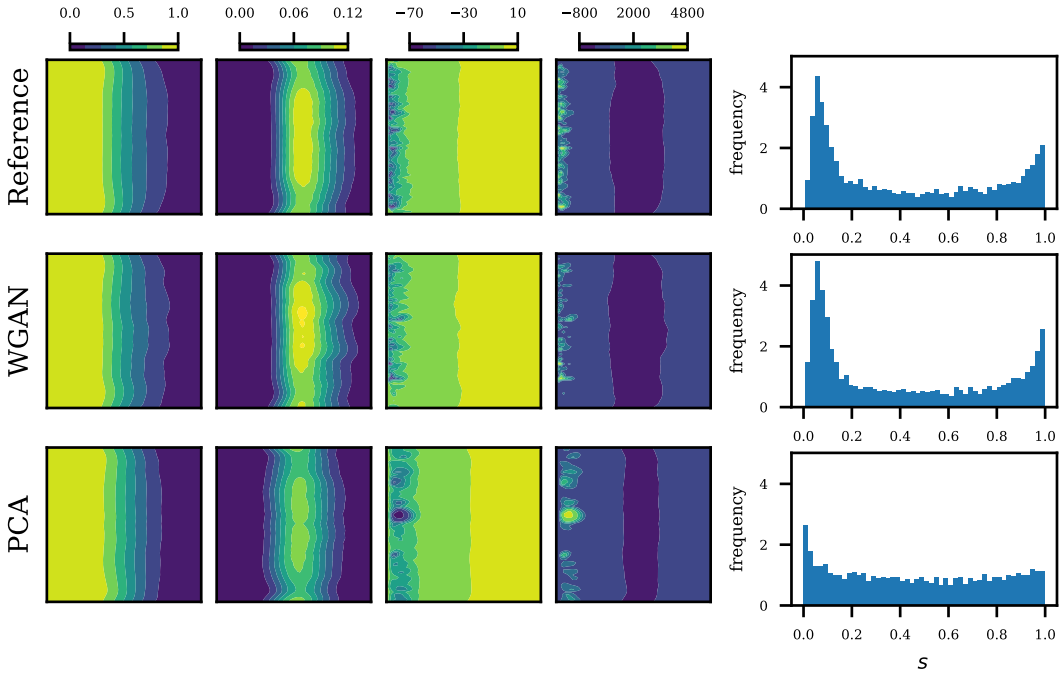
$$-\nabla \cdot (a \nabla p) = q \tag{3.6}$$

$$\varphi \frac{\partial s}{\partial t} + \nabla \cdot (sv) = q_w \tag{3.7}$$

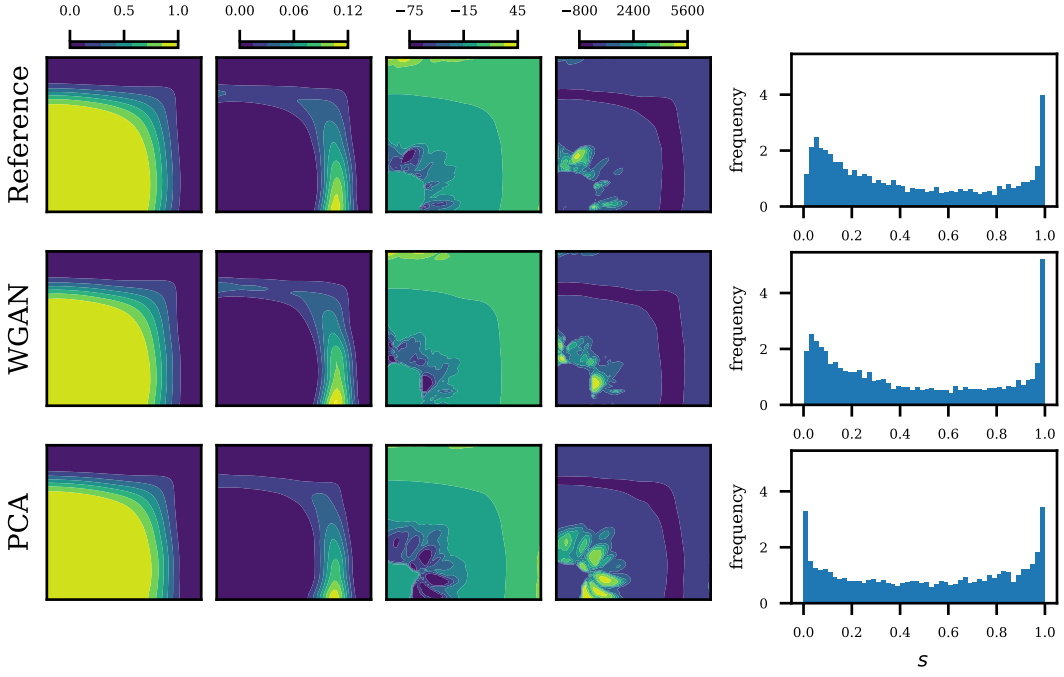
where p is the fluid pressure, $q = q_w + q_o$ denotes (total) fluid sources and sinks, q_w and q_o are the water and oil sources and sinks, respectively, a is the permeability, φ is the porosity, s is the saturation of water, and v is the Darcy velocity.

Our simulation domain is the unit square with 64×64 discretization grid. The reservoir initially contains only oil, i.e. $s(\mathbf{x}, t = 0) = 0$, and we simulate from $t = 0$ until $t = 0.4$. We assume an uniform porosity of $\varphi = 0.2$. We consider two boundary and injection/production conditions:

Uniform flow: We impose uniformly distributed inflow and outflow conditions on the left and right sides of the unit square, respectively, and no-flow boundary conditions on the remaining top and bottom sides. The total injection/production rate is 1. For the unit square, this means $v \cdot \hat{n} = -1$ and $v \cdot \hat{n} = 1$ on the left and right sides, respectively, where $\hat{n}$ denotes the outward-pointing unit normal to the boundary.

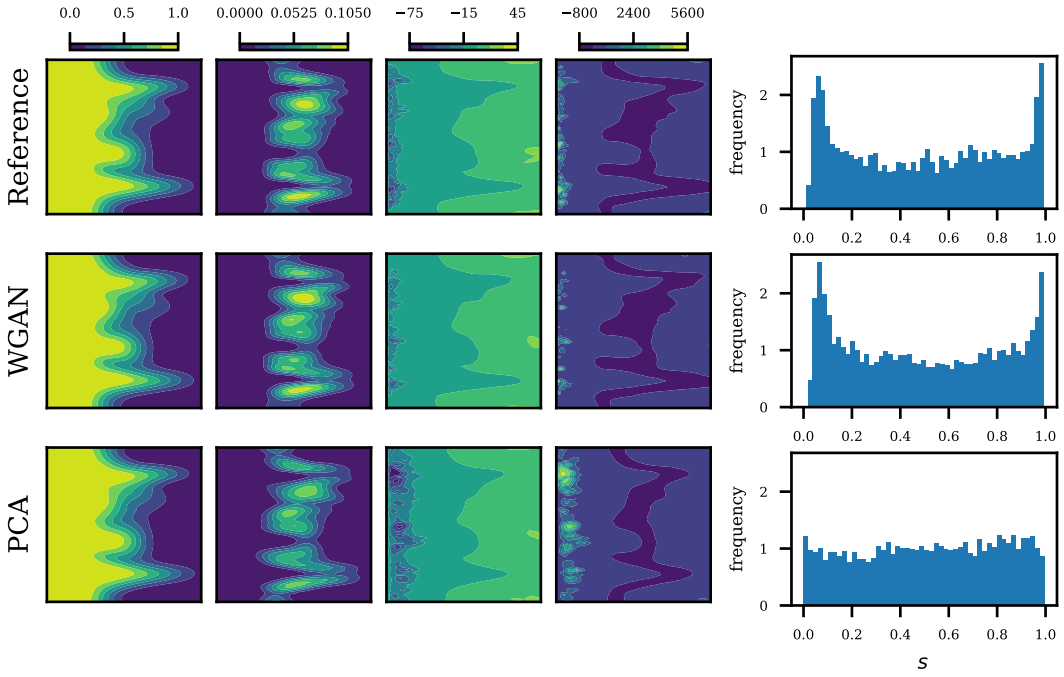


(A) Uniform flow, unconditional realizations

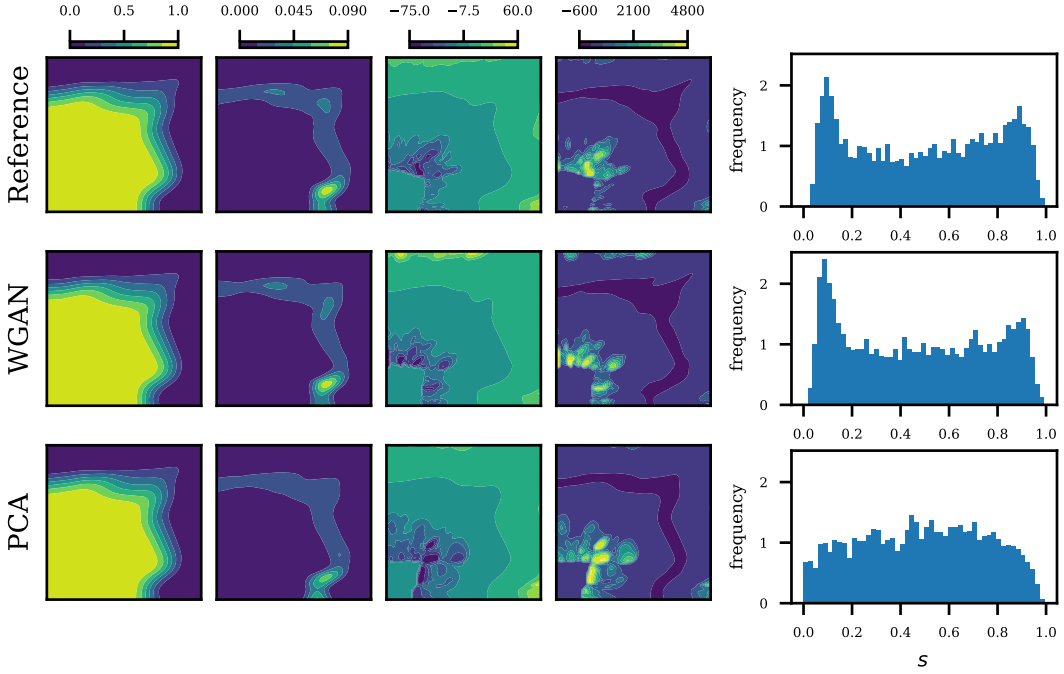


(B) Quarter five, unconditional realizations

FIGURE 3.6: Saturation statistics at $t = 0.5$ PVI for *unconditional realizations*. From left to right: mean, variance, skewness and kurtosis of the saturation map, and lastly the saturation histogram at a given point. The point corresponds to the maximum variance in the reference.

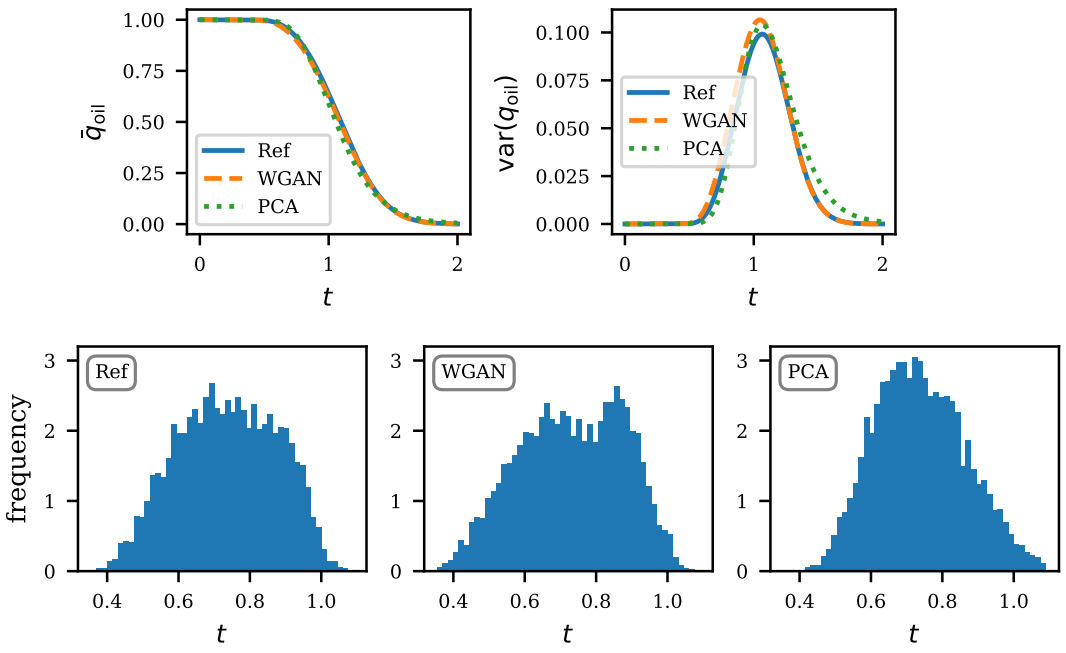


(A) Uniform flow, conditional realizations

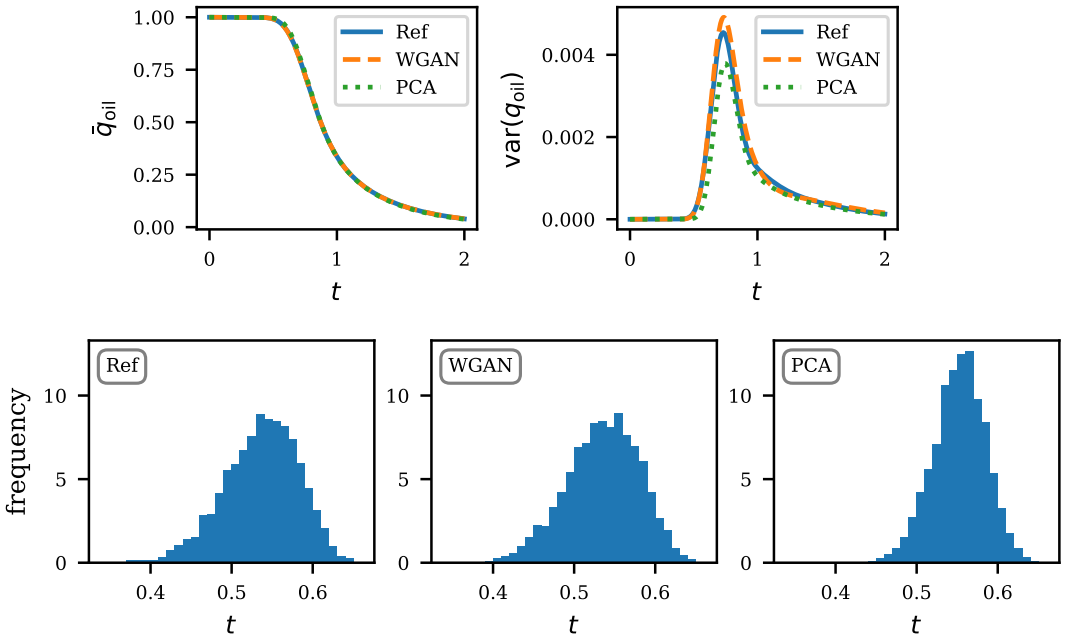


(B) Quarter five, conditional realizations

FIGURE 3.7: Saturation statistics at $t = 0.5$ PVI for *conditional realizations*. From left to right: mean, variance, skewness and kurtosis of the saturation map, and lastly the saturation histogram at a given point. The point corresponds to the maximum variance in the reference.

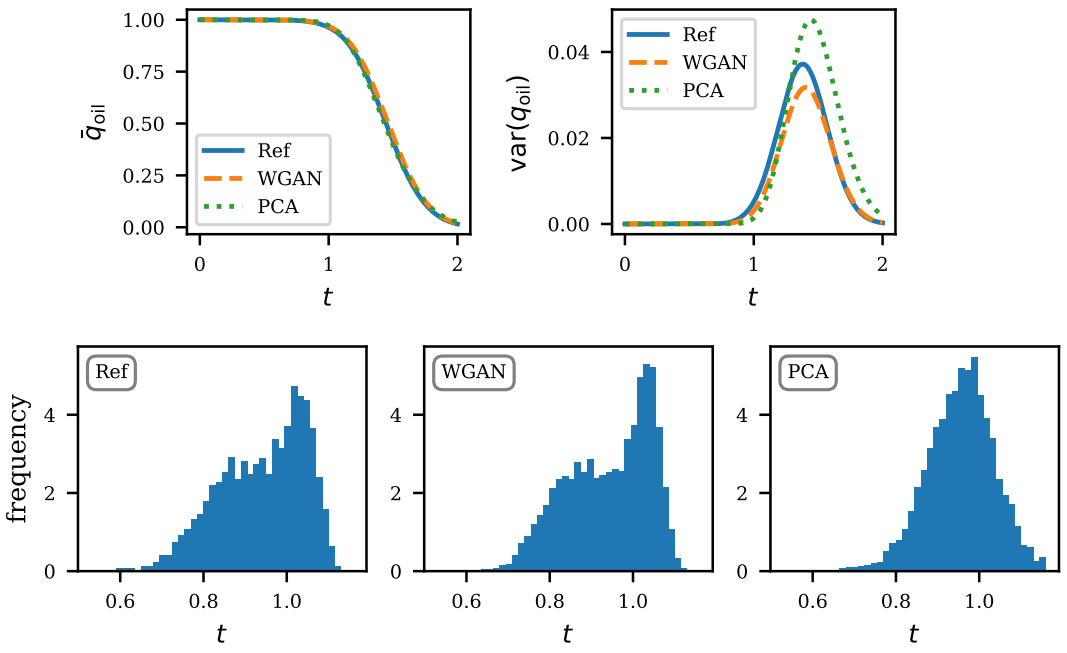


(A) Uniform flow, unconditional realizations

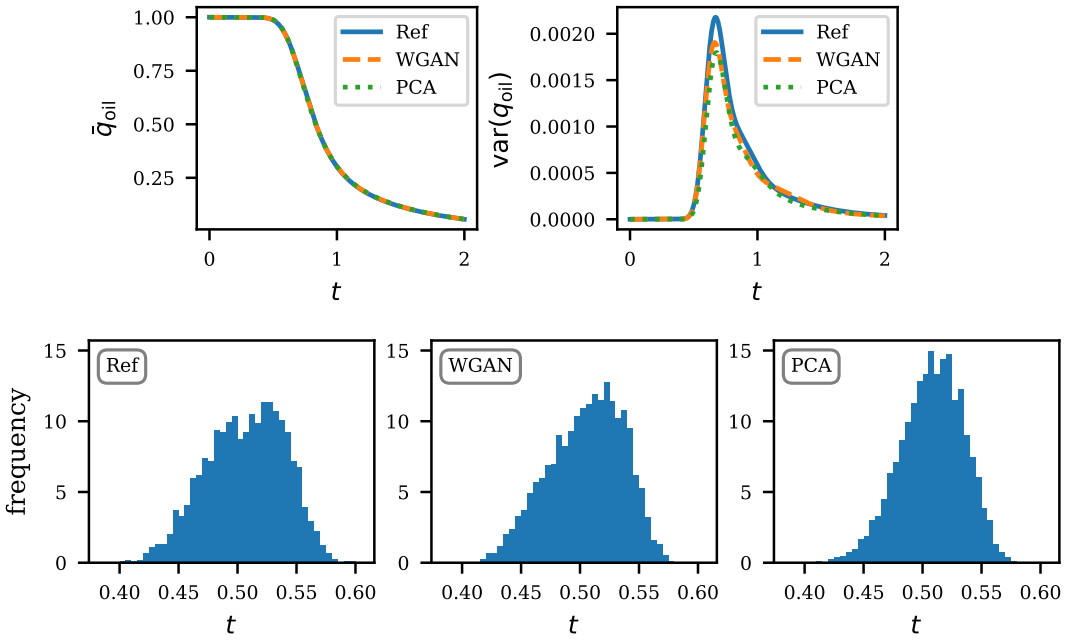


(B) Quarter five, unconditional realizations

FIGURE 3.8: Production statistics for *unconditional realizations*. The top of each subfigure shows the mean and variance of the production curve. The bottom shows the histogram of the water breakthrough time. Times are expressed in pore volume injected.



(A) Uniform flow, conditional realizations



(B) Quarter five, conditional realizations

FIGURE 3.9: Production statistics for *conditional realizations*. The top half of each subfigure shows the mean and variance of the production curve. The bottom show the histogram of the water breakthrough time. Times are expressed in pore volume injected.

Quarter-five spot: We impose injection and production points at $(0, 0)$ and $(1, 1)$ of the unit square, respectively. No-flow boundary conditions are imposed on all sides of the square. The absolute injection/production rate is 1, i.e. $q(0, 0) = 1$ and $q(1, 1) = -1$.

The propagation is done on sets of realizations generated by WGAN and by snesim for comparison. Note that these are fresh realizations not used to train the WGAN models. We also add results using PCA for additional comparison.

Statistics of the saturation map based on 5000 realizations are summarized in Figures 3.6 and 3.7. We plot the saturation at time $t = 0.1$, which corresponds to 0.5 pore volume injected (PVI). From left to right, we plot the mean, variance, skewness and kurtosis of the saturation map. We see that the statistics from realizations generated by WGAN correspond very well with the statistics from realizations generated by snesim (reference). We also see that the PCA parametrization performs very well in the mean and variance, however the discrepancies increase as we move to higher order moments. The discrepancy becomes clearer by plotting the histogram of the 5000 saturations at a fixed point in the domain, shown on the far right of Figures 3.6 and 3.7. We choose the point where reference saturation had the most variance. We see that the histograms by WGAN match the reference remarkably well even for multimodal distributions. The reader may compare our results with [50]. The results suggest that the generator effectively learned to replicate the data generating process.

Statistics of the production curve are summarized in Figures 3.8 and 3.9. On the top half of each subfigure, we show the mean and variance of the production curve based on 5000 realizations. These can in general be approximated well enough by using only the PCA parametrization. We find that the performance of our models are also comparable for this task. To further contrast the ability to preserve higher order statistics, we plot the histogram of the 5000 water breakthrough time results, for which an accurate quantification is of importance in practice. Here we define the water breakthrough time as the time that water level reaches 1% of production. Results are shown on the bottom half of each subfigure in Figures 3.8 and 3.9. In all cases, we find a very good approximation to the reference distribution by WGAN, performing better than a PCA parametrization even for Gaussian-like distributions. Unlike PCA, the responses predicted by WGAN do not have a tendency to be normally distributed (see e.g. Figure 3.9a).

3.3.3 Assessment in parameter estimation

We now assess our models for parameter estimation where we reconstruct the subsurface permeability based on historical data of the oil production stage, also known as history

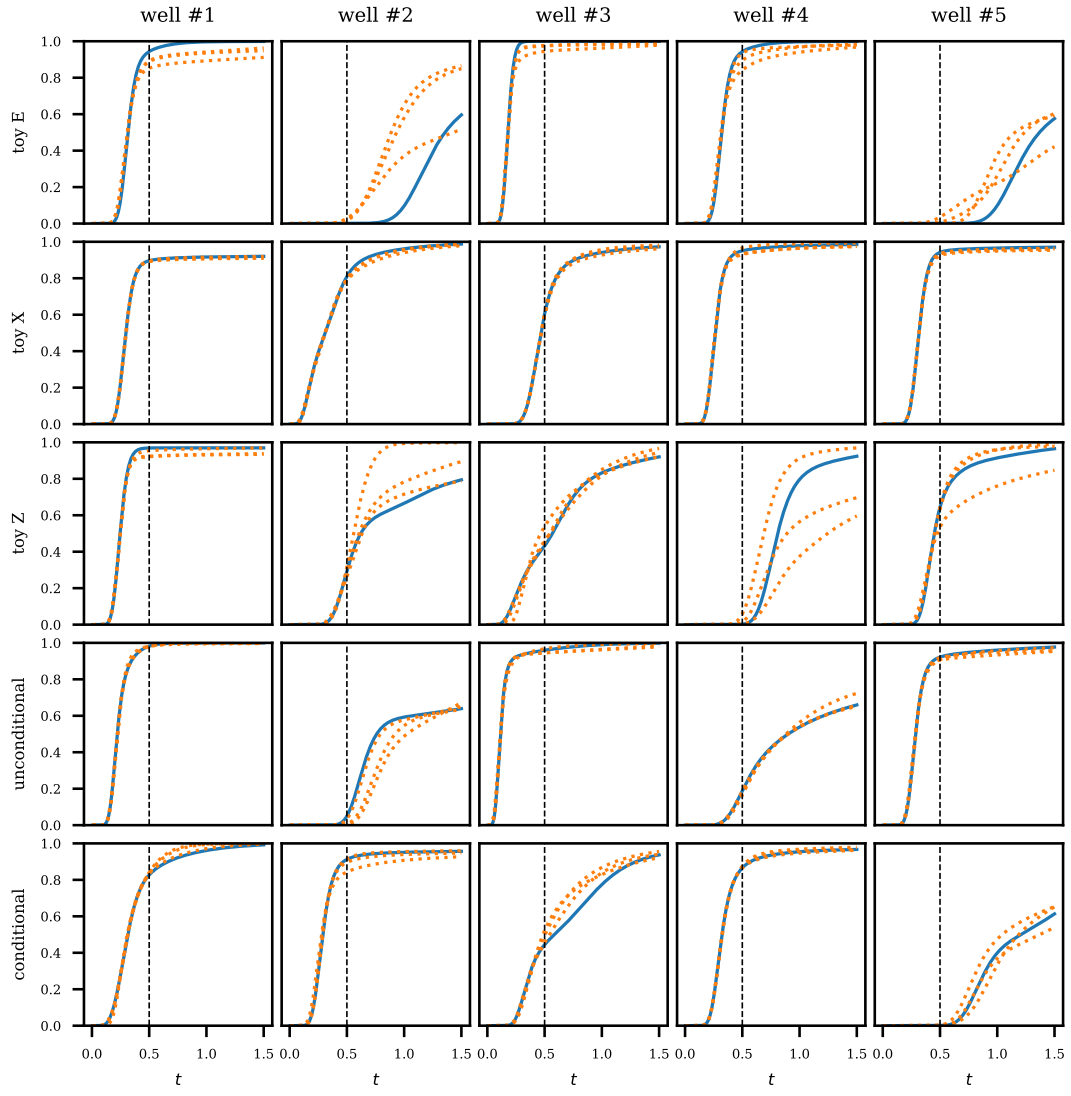


FIGURE 3.10: History matching results. Water level curves from the production wells in different test cases. Blue solid lines denote the target responses. Orange dotted lines are three matching solutions found in the inversion. The black vertical dashed line in each plot marks the end of the observed period. Times are expressed in pore volume injected.

matching. Following the general problem setting from before, we aim to find realizations of the permeability that match the production curves observed at the production wells.

Inversion using natural evolution strategies

Let $\mathbf{d} = \mathcal{M}(\mathbf{a})$ where $\mathcal{M}$ is the forward map, mapping from permeability $\mathbf{a}$ to the output $\mathbf{d}$ being monitored (in our case, the water level curve at the production wells). Given observations $\mathbf{d}_{\text{obs}}$ and assuming i.i.d. Gaussian measurement noise (we use $\sigma = 0.01$),

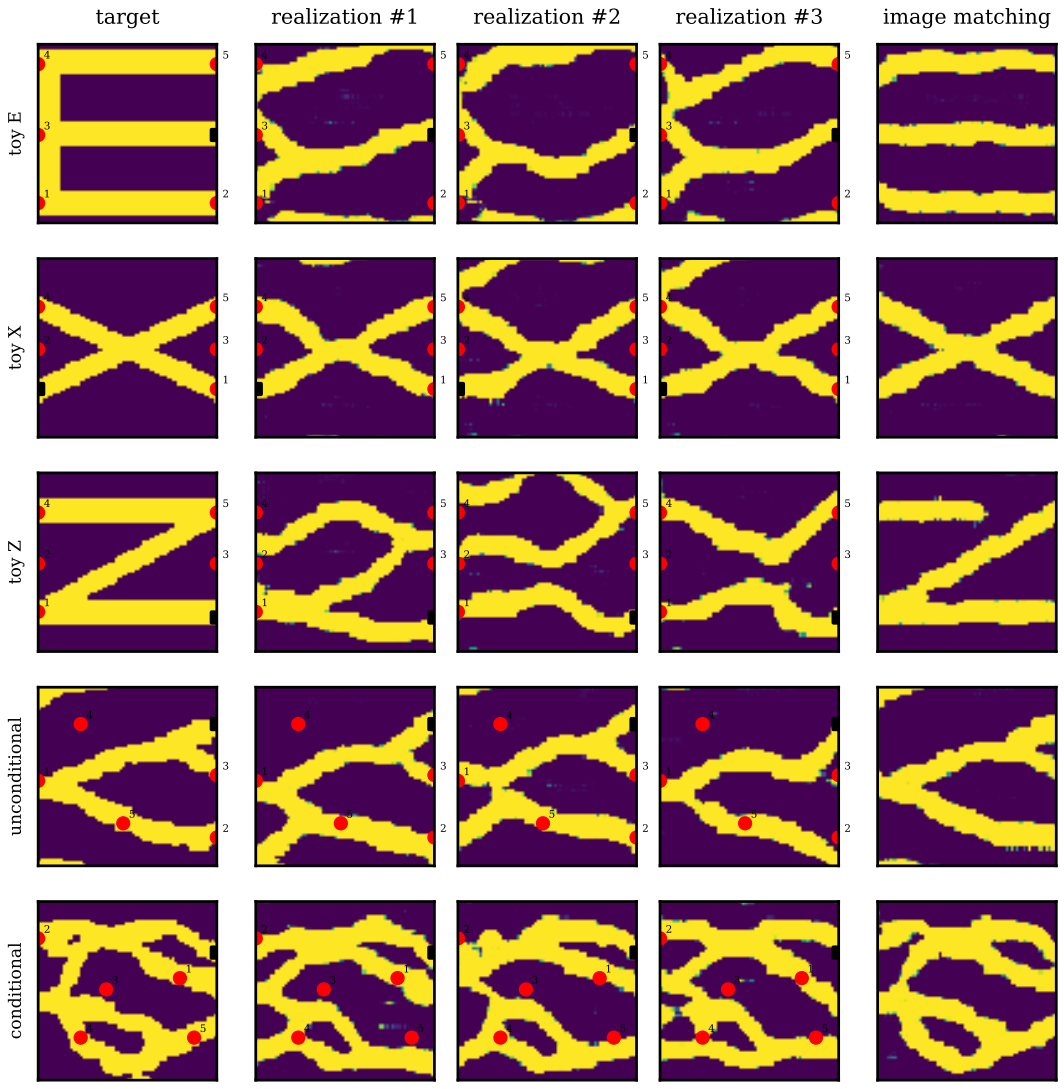


FIGURE 3.11: History matching results. We experiment with three toy images as well as unconditional and conditional snesim realizations. Each case contains one injection well (black square) and five production wells (red circles). We show three solutions that match the observed production period (see Figure 3.10). The last column contains image matching solutions.

and prior $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the objective function to be maximized is

$$f(\mathbf{z}) = -\frac{1}{\sigma^2}(\mathbf{d} - \mathbf{d}_{\text{obs}})^T(\mathbf{d} - \mathbf{d}_{\text{obs}}) - \mathbf{z}^T\mathbf{z} \tag{3.8}$$

$$= -\frac{1}{\sigma^2}(\mathcal{M}(G(\mathbf{z})) - \mathbf{d}_{\text{obs}})^T(\mathcal{M}(G(\mathbf{z})) - \mathbf{d}_{\text{obs}}) - \mathbf{z}^T\mathbf{z} \tag{3.9}$$

To maximize this function, we use natural evolution strategies (NES) [70, 71], a black-box optimization method suitable for the low-dimensional parametrization achieved. Another reasonable alternative is to use gradient-based methods exploiting the differentiability of our generator. This would require adjoint procedures to get the gradient of the

forward map $\mathcal{M}$. We adopted NES due to its generality and easy implementation that does not involve the gradient of f (nor $\mathcal{M}$). NES maximizes f by maximizing an average of f instead, $J(\phi) \coloneqq \mathbb{E}_{\pi(\mathbf{z}|\phi)} f(\mathbf{z})$, where $\pi(\mathbf{z}|\phi)$ is some distribution parametrized by ϕ (e.g. we used the family of Gaussian distributions, in which case ϕ involves the mean and covariance matrix). This is based on the observation that $\max_\phi \mathbb{E}_{\pi(\mathbf{z}|\phi)} f(\mathbf{z}) \leq \max_{\mathbf{z}} f(\mathbf{z})$.

Optimizing the expectation $\mathbb{E}_{\pi(\mathbf{z}|\phi)} f(\mathbf{z})$ (instead of optimizing f directly) has the benefit of not requiring the gradient of f (and therefore of the simulator) since

$$\nabla_\phi J(\phi) = \mathbb{E}_{\pi(\mathbf{z}|\phi)} f(\mathbf{z}) \nabla_\phi \log \pi(\mathbf{z}|\phi)$$

We can be approximated this as

$$\nabla_\phi J(\phi) \approx \frac{1}{N} \sum_{k=1}^{N} f(\mathbf{z}_k) \nabla_\phi \log \pi(\mathbf{z}_k|\phi)$$

by drawing realizations $\mathbf{z}_1, \cdots, \mathbf{z}_N \sim \pi(\mathbf{z}|\phi)$. Optimization proceeds by simple gradient ascent, $\phi \leftarrow \phi + \eta \nabla_\phi J(\phi)$ where η is a step size. Note that we optimize the parameter of the search distribution ϕ, rather than $\mathbf{z}$. As the optimization converges, the search distribution collapses to an optimal value of $\mathbf{z}$. In our implementation, we actually use an improved version of NES which uses the Fisher matrix and natural coordinates, as detailed in [71].

History matching

We consider five target images of the permeability: one unconditional realization and one conditional realization (both using snesim), and three hand-crafted images (see first column in Figure 3.11). The latter were specifically designed to test the limits of the parametrization. For the conditional realization, we use the generator trained on conditional realizations. For the remaining cases, we use the unconditional generator. Note that this poses a difficulty on the hand-crafted toy problems as these have low probability under the generator's distribution.

In each test case, we set one injection well with fixed flow rate of 1, and five production wells with flow rate of -0.2 (locations marked on each image, see Figure 3.11). Our only observed data are the water level curves at the production wells from $t = 0$ to $t = 0.5$ PVI, induced by the target permeabilities. We do not include knowledge of the permeability at the "drilled" wells (as normally done in real applications) in the parameter estimation. For these experiments, we scaled the log-permeability values of 0 and 1 to 0 and 5, emulating a shale and sand scenario. We have done this in part to

allow for a less underdetermined system (i.e. so that a different flow response can be better corresponded to a different permeability pattern).

Results for history matching are shown in Figures 3.10 and 3.11. For each test case, we find three solutions of the inversion problem using different seeds (initial guess). For the conditional and unconditional realizations, we obtain virtually perfect match of the observed period (Figure 3.10). Beyond the observed period, the responses naturally diverge. As is expected, the matching is more difficult for some of the toy problems, in particular toy problem E and toy problem Z. Toy problem X, however, does particularly well.

In Figure 3.11 we show the reconstructed permeabilities for each test case. We also show, in the last column, image matching solutions (we invert conditioning on the whole image using NES). For the conditional and unconditional cases, we see a good visual correspondence between target and solution realizations in the history matching, as well as good visual match in the image matching solutions. This shows that the target image is in the solution space of the parametrization and therefore the history matching can be further improved by supplying more information (e.g. permeability values at wells). This applies to toy problem X as well, where the target seems to have a high probability under the generator's distribution. The reconstruction is more difficult for toy problems E and Z, where the targets seem to have a low probability as suggested by the image matching solutions (and as one could have visually guessed). For these cases, history matching the production data will only improve up to certain point. Note that this is not a failure of the parametrization method; after all, the parametrization is informed by the provided training dataset. In short, the target in question must be a likely realization of the generator's distribution, or rather, the parametrization must be done using samples deemed representative of the geology under study.

3.3.4 Honoring point conditioning

We assess the ability of the generator trained on conditional realizations to reproduce the point conditioning. We analyze 5000 realizations and report in Table 3.2a the percentage of mismatches at each of the 16 conditioning points. We find that mismatches do occur at frequencies of less than 5% at each conditioning point. Next, we count the overall number of realizations with at least 1 mismatch, at least 2 mismatches, and exactly 3 mismatches (there were no realizations with more than 3 mismatches). The result is reported in Table 3.2b. The first row shows the percentage of realizations that contain mismatches. We see that 82.4% of realizations honor all conditioning points. A sizable

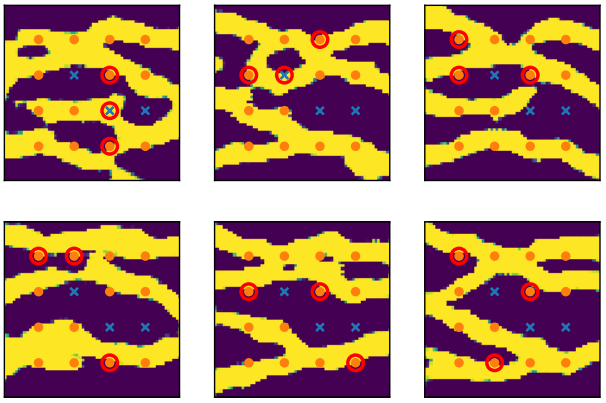
	$j = 12$	$j = 25$	$j = 38$	$j = 51$
$i = 12$	0.38	0.48	1.78	1.82
$i = 25$	0.06	0.34	0.54	0.02
$i = 38$	4.46	1.3	3.0	0.8
$i = 51$	2.06	1.16	1.2	0.26

(A) Percentage of mismatches at each conditioning point.

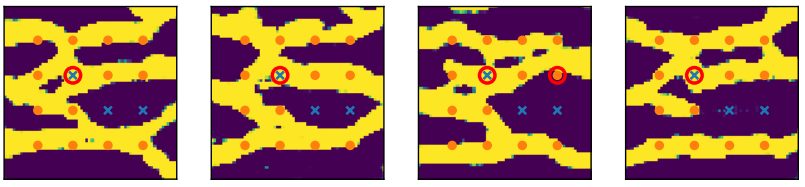
	one or more	two or more	three
exact	17.6	1.82	0.12
1 cell away	1.8	0.02	0.0
2 cells away	0.46	0.0	0.0
3 cells away	0.24	0.0	0.0
4 cells away	0.08	0.0	0.0

(B) Percentage of realizations with mismatches.

TABLE 3.2: Performance in honoring point conditioning.



(A) Realizations containing 3 mismatches.



(B) Realizations with large misplacements (4 cells away).

FIGURE 3.12: Realizations where conditioning failed. Orange dots indicate points conditioned to low permeability (0) and blue crosses indicate points conditioned to high permeability (1). Mismatches are circled in red.

portion (17.6%), however, contains mismatches, although most of them only have 1 mismatch. In particular, we find only 6 (0.12%) realizations containing three mismatches, shown in Figure 3.12a. From the figure, we notice that most mismatches were misplaced by a few cells. A closer look reveals that this is generally the case: In Table 3.2b, we also report the percentage of realizations that contain mismatches with misplacements of 1, 2, 3, and 4 cells (there were no larger misplacements). We find that if we allow a tolerance of 1 cell, the percentage of wrong realizations drops to less than 2%. That is, 98.2% of realizations honor all conditioning points within a 1 cell distance, and 82.4% do so exactly. This could explain the yet good results in flow experiments. Finally, we show in Figure 3.12b the only 4 (0.08%) realizations containing large misplacements of 4 cells.

Note that mismatches do not occur using PCA parametrization (assuming an exact method for the eigendecomposition is used) as it is derived to explicitly preserve the spatial covariances. The presence of mismatches in our method reflects the approach that we take to parametrization: We formulate the parametrization by addressing the data generating process rather than the spatial statistics of the data, resulting in a parametrization that extrapolates to new realizations that, except for a few pixels/cells, are otherwise indistinguishable from data. In view of the good results in our flow experiments, the importance of honoring point conditioning precisely to the cell level could be argued. On the other hand, since conditioning points are normally scarce and obtained from expensive measurements, it is often desirable that these be well honored in the realizations.

3.4 Discussion and practical details

3.4.1 Practical advantages of WGAN

An issue with the standard formulation of GAN is the lack of a convergence curve, or even a loss function that is informative about the sample quality. We illustrate this in Figure 3.13 where we show the convergence curve of our trained WGAN model, and a convergence curve of a GAN using the standard formulation. We also show realizations generated by the models along the training process. The curve of WGAN follows the ideal behavior that is expected in an optimization process, whereas the curve of standard GAN is erratic and shows no correlation with the quality of the generated samples. We can also see another well-known issue of standard GAN which is the tendency to mode collapse, i.e. a lack of sample diversity expressed in the repetition of only one or few image modes. We see that the standard GAN generator jumps from one mode solution

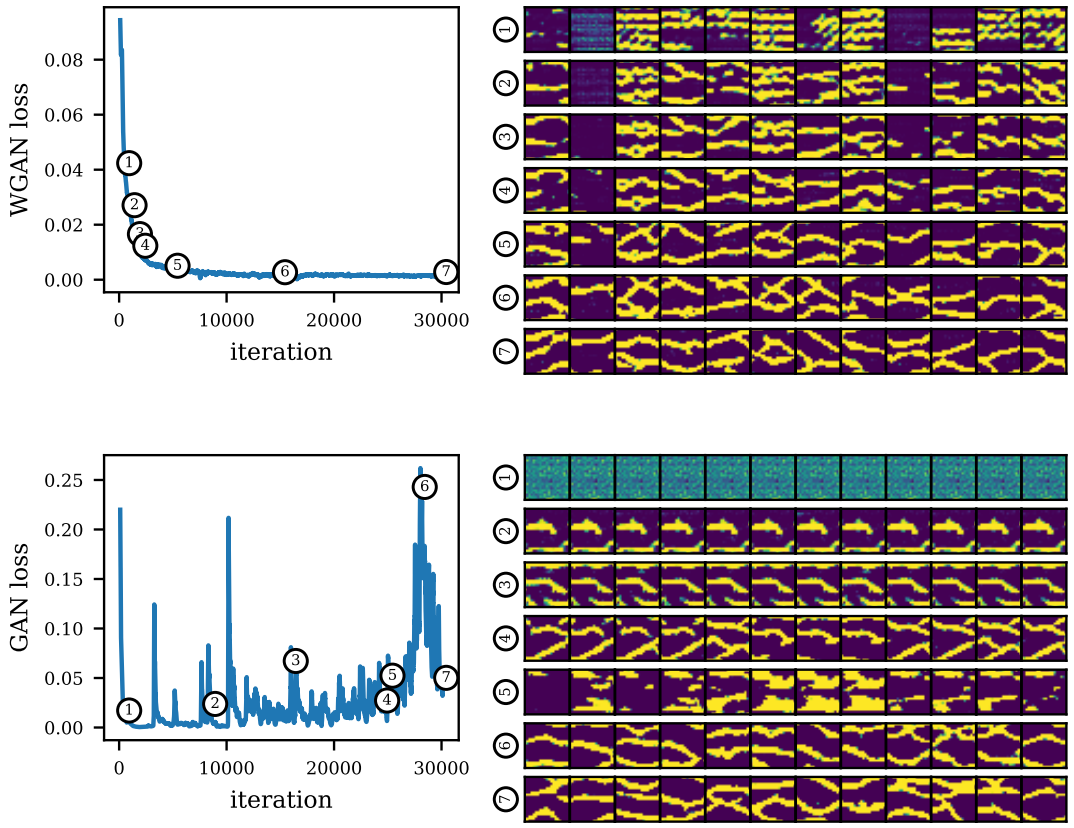


FIGURE 3.13: Convergence curves of a WGAN model (top) and a standard GAN model (bottom). On the right, we show realizations along the training of the corresponding models. We see that GAN loss is uninformative regarding sample quality.

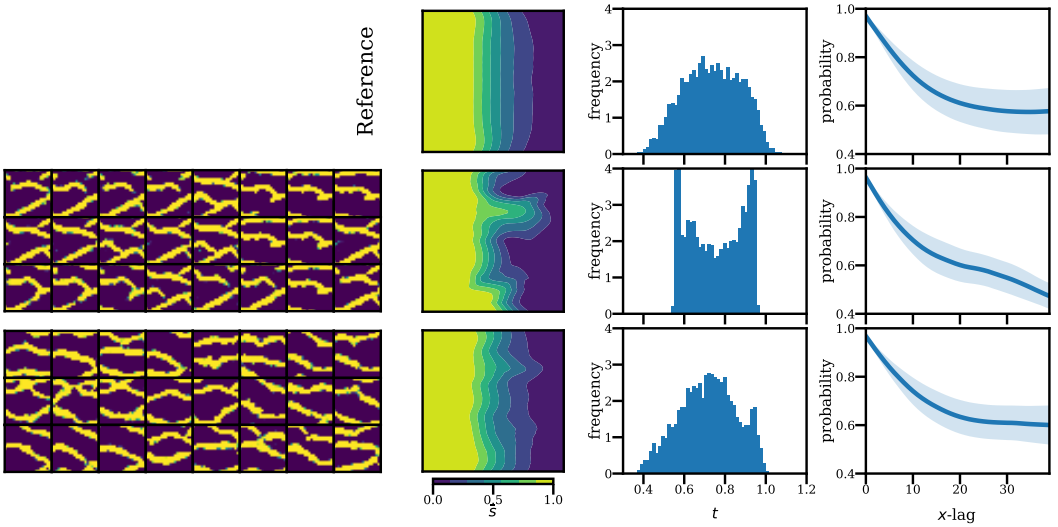


FIGURE 3.14: Examples of missing modes in standard GAN. Second and third rows show realizations generated by collapsing GAN models (left) and their responses (right). First row shows the reference solutions. The standard GAN was trained using the same generator architecture, but a $\times 4$ larger discriminator than the one used in WGAN. We did not manage to find convergence with smaller discriminator sizes.

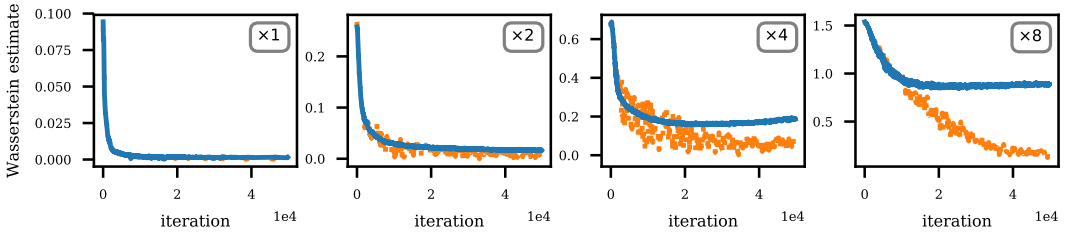
to another. Note that in some cases, however, mode collapse is more subtle and not easily detectable. This is very problematic to our application since it can lead to biases in uncertainty quantification and unsuccessful history matching due to the absence of some modes in the generator.

Given the lack of an informative convergence metric in standard GAN, the training process would involve a human judge serving as the actual loss function to track the visual quality along the training (in practice, weights are saved at several checkpoints and assessed after the fact). On top of this, the human operator would need to look at multiple realizations at once in an attempt to detect mode collapse. Clearly, this subjective process is error prone, not to mention labor intensive. In Figure 3.14 we show two standard GAN models and their flow responses in the unconditional uniform flow test case, based again on 5000 realizations. On the top row, we show again the reference results (mean saturation and water breakthrough time) for comparison. We also compute the two-point probability function [92] of the generated realizations (last column; we show the mean and one standard deviation). We see that in some cases, mode collapse is very evident and the model can be quickly discarded (second row). In other cases (last row), mode collapse is harder to detect and can lead to misleading predictions. We also see that the two-point probability function is not sufficient to detect mode collapse as this function does not measure the overall sample diversity.

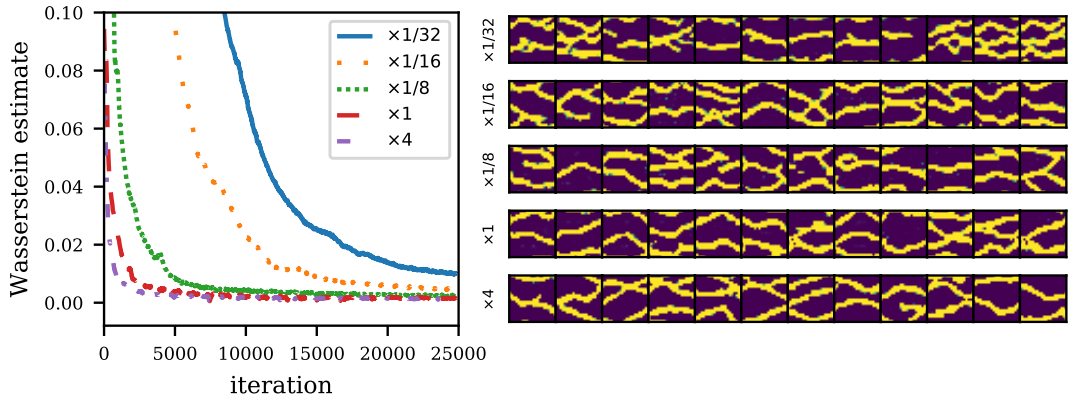
The Wasserstein formulation circumvents these issues by actually allowing the generator to minimize the Wasserstein distance between its distribution and the target distribution (Equation (3.5)), therefore reducing mode collapse. Moreover, the Wasserstein distance is readily available in-training and can be used to assess convergence. Therefore, the Wasserstein formulation is better suited for automated applications, where robustness and a convergence criteria would be necessary.

3.4.2 Network sizes under limited data

As mentioned earlier, architecture design is largely problem-dependent and led by heavy use of heuristics and domain knowledge. The general approach is to start with a baseline architecture from a similar problem domain and tune it to accommodate for the present problem. Current computer vision applications use the pyramid architecture shown in Figure 3.2. These applications benefit from very large datasets of images. In contrast, our application uses a relatively small dataset. Recall that the discriminator D is trained using a limited dataset, therefore D can overfit if it is too large for this dataset. This creates an issue where the Wasserstein estimate in Equation (3.5) is no longer accurate, making the gradients to the generator unreliable. We show the effect



(A) Convergence curve for different sizes of D (and fixed G). Solid blue lines indicate the training loss, and orange dotted lines indicate the validation loss. Note that the losses cannot be compared since the Lipschitz constants are different.

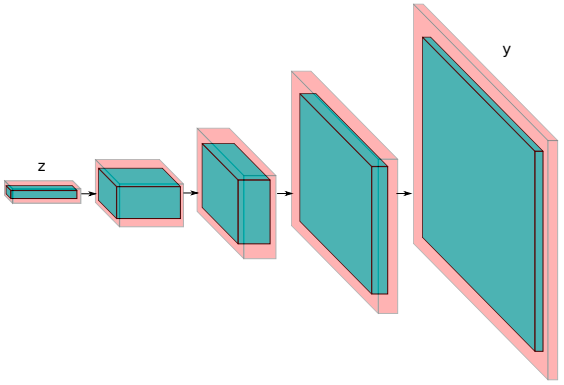


(B) *Left:* Convergence curve for different sizes of G (and fixed D). *Right:* Realizations by generators of different sizes (at 15,000 iterations).

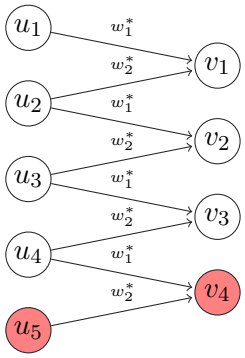
FIGURE 3.15: Performance of models with varying network sizes.

of overfitting in Figure 3.15a by training models with different discriminator sizes, and fixed generator architecture. We train discriminators of 2, 4, and 8 times the size of the discriminator used in our previous experiments. The way we increase the model sizes is by increasing the number of filters in each layer of the discriminator, while keeping everything else constant. Another possibility is to add extra layers to the architecture. To detect overfitting, we evaluate the Wasserstein estimate using a separate validation set of 200 snesim realizations. We see that for an adequate size of the discriminator, the Wasserstein estimate as evaluated on either training or validation set are similar. However for larger models, the Wasserstein estimates on the training and validation sets start to wildly diverge as the optimization progresses, suggesting that the discriminator is overfitting and the estimates are no longer reliable. It is therefore necessary to adjust the size of the discriminator or use regularization techniques when data is very limited.

Regarding generator architectures, network sizes will in general be limited by compute and time resources; on the other hand, we only need just enough network capacity to be able to model complex structures. We illustrate this in Figure 3.15b where we train generators of different sizes (like before, we vary the number of filters in each layer) and



(A) Illustration of artificially expanding the input tensor in the generator network. Blue blocks represent the original state shapes that a normal input tensor follows in the generator. Light red blocks represent the new state shapes of an expanded input tensor.



$$\mathbf{W} = \begin{pmatrix} w_1^* & w_2^* & 0 & 0 & 0 \\ 0 & w_1^* & w_2^* & 0 & 0 \\ 0 & 0 & w_1^* & w_2^* & 0 \\ 0 & 0 & 0 & w_1^* & w_2^* \end{pmatrix}$$

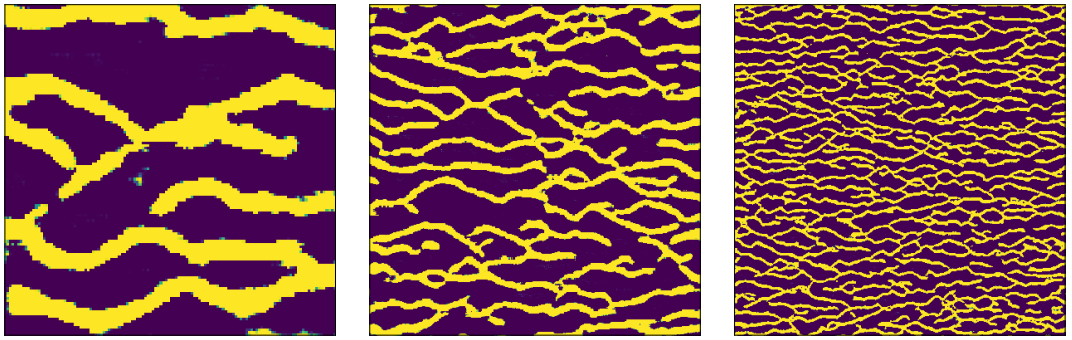
(B) 1D example of the artificial expansion and its associated matrix modification. Weights w_1^*, w_2^* are already trained. The expanded matrix can be obtained by appending an additional row and column.

FIGURE 3.16: Examples of artificially expanding the input tensor to obtain a larger output.

fixed discriminator architecture. We train generators of $\frac{1}{32}$, $\frac{1}{16}$, $\frac{1}{8}$, and 4 times the size of the generator used in previous experiments. We also show realizations generated by each generator model after 15,000 training iterations. We see that for a very small network model ($\frac{1}{32}$), convergence is slow (as measured by iterations). Convergence is faster as the network size increases since it is easier to fit a larger network. Note, however, that iterations of larger networks are more expensive, possibly making convergence actually slower in terms of compute time. Moreover, a larger generator has a higher forward evaluation cost, impacting the performance in deployment. Therefore, it becomes ineffective to keep increasing the network size after certain point.

3.4.3 GAN for multipoint geostatistical simulations

In the domain of geology, a natural question is whether GANs can be applied directly as multipoint geostatistical simulators. This has been studied in a number of recent



(A) 128×128, $n_x = 5$ (B) 368×368, $n_x = 20$ (C) 848×848, $n_x = 50$

FIGURE 3.17: Artificially upscaled realizations by feeding an expanded input tensor. Images (evidently) not at scale.

works [75–77]. The idea here is to use a single large training image and train a GAN model on patches of this image. The result is a generator capable of generating images that resemble the patches of the training image. This simple approach circumvents the usual requirement of large datasets in deep learning techniques. If the generator is solely composed of convolutional layers, we can recover the original size of the training image or generate a larger image by feeding an artificially expanded input tensor to the generator. This is illustrated in Figure 3.17: If a generator has been trained with $\mathbf{z}$ of shape $(n_z, 1, 1)$, we can feed the generator with new tensors of shape (n_z, n_y, n_x) (sampled from the higher dimensional analogue of the same distribution) to obtain larger outputs (Figure 3.16a). This is possible since we can still apply a convolving filter regardless of the input size. This is better illustrated with the 1D example shown in Figure 3.16b. In Figure 3.17, we show some examples using this trick on our trained WGAN model. Whether this trick generalizes to arbitrarily large domains is unclear; in Chapter 5, we introduce a new method to obtain a generative neural network directly from a single exemplar image that avoids this trick.

A possible use case in geomodeling and parametrization is to use a generator that has been trained on unconditional realizations to generate conditional realizations. This has been the focus of recent works in [78, 79] where conditional realizations are obtained from an unconditional generator by performing an optimization in the latent space using an image inpainting technique [93]. In Chapter 4, we propose a method to generate conditional realizations using the pre-trained unconditional generator without sacrificing the parametrization of the generation process. It is worth emphasizing, however, that post-conditioning using an unconditionally trained generator is feasible only if the conditioning is reasonable under the distribution of the training set used to train the generator. As seen in Section 3.3.3, if a target realization has a low probability under the generator's distribution, it is very difficult to reconstruct this realization using

such generator. Likewise, if a conditioning has a low probability under the generator's distribution, it might be very difficult to obtain a realization honoring this conditioning.

3.5 Conclusions

We investigated generative adversarial networks (GAN) as a sample-based parametrization method for geological properties. We parametrized conditional and unconditional permeability, and used the parametrization to perform uncertainty quantification and parameter estimation (history matching). Overall, the method shows very good results in reproducing the spatial statistics and flow responses, as well as preserving visual realism while achieving a dimensionality reduction of two orders of magnitude, from 64×64 to 30. In uncertainty quantification, we found that the method reproduces the high order statistics of the flow responses as evidenced by the estimated distributions of the saturation and the production – in particular, the modality of the distributions are preserved. In parameter estimation, we found successful inversion results in both conditional and unconditional settings, and reasonable inversion results for challenging hand-crafted images that are not plausible. We also compared implementations of the standard formulation of GAN with the Wasserstein formulation, finding the latter to be more suitable for our applications. We discussed issues regarding network size under limited data, finding that the size of the discriminator is important and should be carefully tuned to prevent overfitting. Finally, we discussed current trends in using GANs for multipoint geostatistical simulations. Possible directions to extend this work include improving current GAN methods for limited data, and further assessments in other test cases.

Chapter 4

Parametric generation of conditional geological realizations using generative neural networks

We introduce a method for parametric generation of conditional geological realizations using generative neural networks. We build upon our work in Chapter 3 where we trained a neural network to generate unconditional geological realizations using generative adversarial networks. Here we propose a method for *post-hoc* conditioning of pre-trained generator networks to generate conditional realizations. We frame the problem in the Bayesian setting and model the posterior distribution of the latent vector given observations. We then train an *inference neural network* to sample from the posterior distribution. Once trained, the inference network is coupled with the unconditional generator to obtain the conditional generator, thus also maintaining a parametrization of the conditional generation process.

4.1 Introduction

The large scale nature of geological models makes reservoir simulations an expensive task, prompting numerous works that aim for a reduced representation of the geological properties that can preserve the heterogeneous characteristics required for accurate flow modeling. Very recently, a new method from the machine learning community called

In preparation for submission to journal.

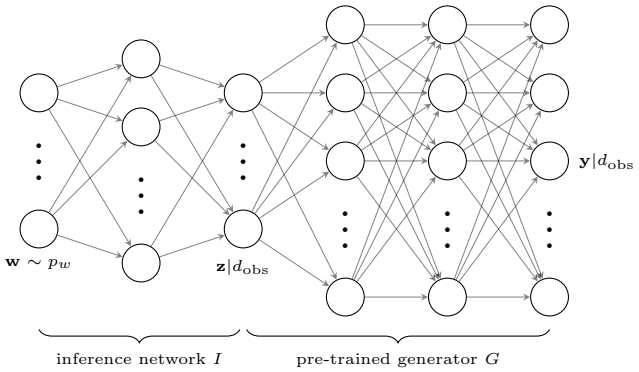


FIGURE 4.1: Overview of methodology, $G \circ I$.

generative adversarial networks [69] has been investigated (see [75–79] as well as Chapter 3) for the purpose of parametrization, reconstruction, and synthesis of geological properties; showing very promising results. This adds to the recent trend in applying machine learning techniques to leverage the increasing availability of data as well as rapid advances in the field [94–100].

Generative adversarial networks is a novel technique for training a neural network to sample from a distribution that is unknown and intractable, by only using samples from this distribution. The result is a *generator network* that is capable of generating realizations from the target distribution –in our case, geological realizations– using a very reduced number of parameters. This is possible thanks to the high expressive power of neural networks. In particular, the method has shown to preserve visual realism as well as flow statistics of the training data in experiments parametrizing geological images.

Recent works [78, 79] focused on the problem of *post-hoc* conditioning of the generator network: given a generator trained on unconditional realizations, the task is to generate realizations conditioned on new spatial observations (hard data). Current approaches are based on a recent inpainting technique introduced in [93] that requires solving an optimization problem for each conditional realization, which can be expensive if several realizations are required, e.g. for history matching or uncertainty quantification. Moreover, the parametrization of the generation process is sacrificed.

In this work, we propose a method to obtain a conditional generator to directly sample conditional realizations. Our emphasis here is on *parametric* sampling, that is, we want to generate conditional realizations without sacrificing the parametrization of the sampling process. To this end, we begin by formulating the posterior distribution of the latent vector conditioned on the new observations using a Bayesian framework. A comparison of this formulation to the recent inpainting technique in [78, 79] is discussed. Thereafter, we train an *inference network* to sample from this posterior distribution

by minimizing the Kullback-Leibler divergence between the inference network's distribution and the posterior. Finally, this inference network is coupled with the original unconditional generator to obtain the conditional generator, as illustrated in Figure 4.1. Sampling new conditional realizations can be done very efficiently and the parametrization of the generation process is maintained. The inference network is usually small since it is a low-to-low dimensional mapping, thus it is relatively easy to train and the increase in complexity of the resulting parametrization is negligible.

The rest of this chapter is organized as follows: In Section 4.2, we briefly describe generative adversarial networks and the Bayesian framework. In Section 4.3, we introduce a method to train an inference neural network to sample from the posterior distribution. In Section 4.4, we show results for geological realizations conditioned on several test cases. Finally, in Section 4.5 we discuss alternatives to the current work and possible directions.

4.2 Background

We briefly describe generative adversarial networks (GAN) and the Bayesian framework for conditioning of geological realizations. Although not central to the method presented here, GAN was used to obtain the unconditional geomodel generator.

4.2.1 Generative adversarial networks

We represent the uncertain subsurface property of interest as a random vector $\mathbf{y} \in \mathbb{R}^{n_y}$ where n_y is very large (e.g. permeability discretized by the simulation grid). This random vector follows a distribution $\mathbf{y} \sim \mathbb{P}_y$ that is unknown and intractable (e.g. distribution of permeability with channels), and instead we are given a set of realizations $\{y_1, \cdots, y_N\}$ of the random vector (e.g. a set of permeability models deemed representative of the area under study). Using this training set, the hope is to find a representation of $\mathbf{y}$ in terms of a reduced number of free parameters. The approach taken here and in recent works is to consider a *latent* random vector $\mathbf{z} \in \mathbb{R}^{n_z}$ with $n_z \ll n_y$ and $\mathbf{z} \sim p_z$ where p_z is manually chosen to be easy to sample from (e.g. a multivariate normal or uniform distribution); and a deterministic neural network $G_\theta \colon \mathbb{R}^{n_z} \to \mathbb{R}^{n_y}$, called a *generator*, parametrized by weights θ to be determined. Given p_z fixed, G_θ induces a distribution $G_\theta(\mathbf{z}) \sim \mathbb{P}_\theta$ which is now unknown and possibly intractable (since G_θ is a neural network with many nonlinearities). On the other hand, sampling from this distribution is easy since it only requires sampling $\mathbf{z} \sim p_z$ and forward-passing through G_θ. The goal is to optimize θ so that $\mathbb{P}_\theta = \mathbb{P}_y$.

A difficulty in this problem is that both $\mathbb{P}_y$ and $\mathbb{P}_\theta$ are unknown and intractable. Nevertheless, sampling from these distributions is easy (for $\mathbb{P}_y$, one draws a batch of realizations from the training set, assuming the set is big enough). Following this observation, the seminal work in [69] introduces the idea of using a classifier function $D_\psi \colon \mathbb{R}^{n_y} \to [0,1]$, called a *discriminator*, to assess whether a generated realization $\tilde{y}_i = G_\theta(z_i)$ "looks real", i.e. is similar to realizations from the training set. The discriminator is also typically a neural network with weight parameters ψ to be determined. The discriminator is trained to solve a binary classification problem, maximizing the following loss

$$\mathcal{L}(\psi,\theta) \coloneqq \mathop{\mathbb{E}}_{\mathbf{y}\sim\mathbb{P}_y} \log D_\psi(\mathbf{y}) + \mathop{\mathbb{E}}_{\tilde{\mathbf{y}}\sim\mathbb{P}_\theta} \log(1 - D_\psi(\tilde{\mathbf{y}})) \tag{4.1}$$

$$\approx \frac{1}{M}\sum_{i=1}^{M} \log D_\psi(y_i) + \frac{1}{M}\sum_{i=1}^{M} \log(1 - D_\psi(G_\theta(z_i)) \tag{4.2}$$

which is in essence a binary classification score. The approximation is done by taking a batch of $M \leq N$ realizations from the training set for the first term, and sampling M realizations $z_1, \cdots, z_M$ from p_z for the second term.

The generator on the other hand is trained to minimize the same loss, thus an adversarial game is created where G and D optimize the loss in opposite directions,

$$\min_\theta \max_\psi \mathcal{L}(\psi,\theta) \tag{4.3}$$

In practice, this optimization is performed alternately using gradient-based methods, where the gradients with respect to θ and ψ are obtained using automatic differentiation algorithms. The equilibrium is reached when G effectively learns to approximate $\mathbb{P}_y$ and D is $\frac{1}{2}$ in the support of $\mathbb{P}_y$ (coin toss scenario). It is shown in [69] that in the limiting case, this process minimizes the Jensen-Shannon divergence between $\mathbb{P}_\theta$ and $\mathbb{P}_y$.

Variations of GAN Stability issues with the original formulation of GAN has led to numerous works to improve stability and generalize the method (e.g. see [83, 85, 89, 101] and references therein). One line of research generalizes GAN in the framework of integral probability metrics [102]. Given two distributions $\mathbb{P}$ and $\mathbb{Q}$, and a set of real valued functions $\mathcal{D}$, an integral probability metric measures the discrepancy between $\mathbb{P}$ and $\mathbb{Q}$ as follows,

$$\mathrm{d}_\mathcal{D}(\mathbb{P},\mathbb{Q}) = \sup_{D\in\mathcal{D}} \{ \mathop{\mathbb{E}}_{\mathbf{y}\sim\mathbb{P}} D(\mathbf{y}) - \mathop{\mathbb{E}}_{\tilde{\mathbf{y}}\sim\mathbb{Q}} D(\tilde{\mathbf{y}}) \} \tag{4.4}$$

Note the slight similarity with Equation (4.1). The choice of set $\mathcal{D}$ is important and leads to several formulations of GAN. When $\mathcal{D}$ is a ball in a Reproducing Kernel Hilbert Space, $\mathrm{d}_\mathcal{D}$ is the Maximum Mean Discrepancy (MMD GAN) [103, 104]. When $\mathcal{D}$ is a

set of 1-Lipschitz functions, $\mathrm{d}_{\mathcal{D}}$ is the Wasserstein distance (WGAN) [90, 91]. When $\mathcal{D}$ is a Lebesgue ball, we obtain Fisher GAN [105], and when $\mathcal{D}$ is a Sobolev ball, we obtain Sobolev GAN [106]. See [106, 107] for an in-depth discussion. Our unconditional geomodel generator was trained using the Wasserstein formulation (see Chapter 3).

4.2.2 Conditioning on observations

Given a pre-trained generator G, one possible use case is to obtain realizations conditioned on new spatial observations (hard data), that is, we need to find z such that $G(z)$ honors the observations. Let d_{obs} denote the observations and $d(z) = G(z)_{\text{obs}}$ the values at the observed locations given $G(z)$. Under the probabilistic framework, the problem is to find z^* that maximizes its posterior probability given observations,

$$z^* = \arg\max_z p(z|d_{\text{obs}}) \tag{4.5}$$

From Bayes' rule and applying logarithms,

$$p(z|d_{\text{obs}}) \propto p(d_{\text{obs}}|z)p(z) \tag{4.6}$$

$$-\log p(z|d_{\text{obs}}) = -\log p(d_{\text{obs}}|z) - \log p(z) + \text{const.} \tag{4.7}$$

For the prior $p(z)$, a natural choice is p_z for which the generator has been trained. In most applications (and in ours), this is the multivariate standard normal distribution. For the likelihood $p(d_{\text{obs}}|z)$, we take the general assumption of i.i.d. Gaussian measurement noise, $p(d_{\text{obs}}|z) \propto \exp(-\frac{1}{2\sigma^2}\|d(z) - d_{\text{obs}}\|^2)$ where σ is the measurement standard deviation. Then the optimization in Equation (4.5) can be written as

$$z^* = \arg\min_z \mathcal{L}(z) \tag{4.8}$$

$$\mathcal{L}(z) \coloneqq -\log p(z|d_{\text{obs}}) \tag{4.9}$$

$$\overset{(\times 2\lambda)}{=} \|d(z) - d_{\text{obs}}\|^2 + \lambda\|z\|^2 \tag{4.10}$$

$$= \|G(z)_{\text{obs}} - d_{\text{obs}}\|^2 + \lambda\|z\|^2 \tag{4.11}$$

where we multiplied everything by $\lambda = \sigma^2$ and discarded the irrelevant constant. One way to draw different conditional realizations is to optimize Equation (4.8) using a local optimizer and different initial guesses for z.

Comparison to GAN-based inpainting techniques In image processing, image inpaiting is used to fill incomplete images or replace a subregion of an image (e.g. a face with eyes covered). The recent GAN-based inpainting technique by Yeh et al. [93] and

employed in [78, 79] uses an optimization procedure with the following loss

$$\mathcal{L}(z) = \|G(z)_{\text{obs}} - d_{\text{obs}}\|^2 + \lambda \log(1 - D(G(z))) \tag{4.12}$$

The second term in this equation is referred as the *perceptual loss* and is the same second term in the GAN loss in Equation (4.1), which is the classification score on synthetic realizations. We can expect the perceptual loss to act as a regularization that drives z towards a region of high density, or at least towards the support of p_z, assuming that G and D have been trained to convergence, since then D is at an optima for any realization of $G(\mathbf{z})$ for $\mathbf{z} \sim p_z$. We should then expect the perceptual loss to have the same effect as the Bayesian prior p_z. For example, let $\mathbf{z} \sim \mathcal{U}[0, 1]$ and $\mathbf{y} \sim \mathcal{U}[1, 3]$. Then an optimal generator is $G(z) = 2z + 1$ and an optimal discriminator is $D(y) = 1/2$ for $y \in [1, 3]$ and $D(y) = 0$ otherwise. Then $D(G(z)) = 1/2$ for $z \in [0, 1]$, and $D(G(z)) = 0$ otherwise, which is precisely the density function of $\mathbf{z} \sim \mathcal{U}[0, 1]$ scaled by $1/2$. Nevertheless, the perceptual loss can be very useful in practice when G and D are not exactly optimal and there exist realizations $G(z)$ of bad quality. In that case, the perceptual loss can help the optimization to find good quality solutions. In our work, we found the Bayesian prior to be sufficient while removing a layer of complexity in the optimization.

Finally, we also note that both L1 and L2 norms are explored in [93] for the likelihood term, with L1 corresponding to the likelihood $\propto \exp(-\frac{1}{\lambda}\|d(z) - d_{\text{obs}}\|)$.

4.3 Conditional generator for geological realizations

As mentioned in Section 4.2.2, one way to sample multiple realizations conditioned on observations is to solve Equation (4.8) using a local optimizer with different initial guesses. This approach, however, can be expensive and may not capture the full solution space. A better approach could be to use Markov chain Monte Carlo methods, given the latent vector is of moderate size, to better capture the full posterior distribution. Neither approach, however, maintains the parametrization of the sampling process.

We propose constructing a neural network that learns to sample from the posterior distribution. This *inference network* $I_\phi \colon \mathbb{R}^{n_w} \to \mathbb{R}^{n_z}$ is yet another generator network that maps from realizations of a random vector $\mathbf{w} \sim p_w$ with chosen p_w (we naturally chose $p_w = p_z$ and $n_z = n_w$) to realizations of $\mathbf{z}|d_{\text{obs}} \sim p(z|d_{\text{obs}})$. Let $I_\phi(\mathbf{w}) \sim q_\phi(z)$ be the distribution density induced by I_ϕ. This distribution is now unknown and intractable, but is easy to sample from since it only requires sampling $\mathbf{w} \sim p_w$ and forward-passing

through I_ϕ. The Kullback-Leibler divergence from $p(\cdot|d_{\text{obs}})$ to q_ϕ gives us

$$\mathrm{D}_{\mathrm{KL}}(q_\phi \parallel p(\cdot|d_{\text{obs}})) = \mathop{\mathbb{E}}_{\mathbf{z}\sim q_\phi} \log \frac{q_\phi(\mathbf{z})}{p(\mathbf{z}|d_{\text{obs}})} \tag{4.13}$$

$$= \mathop{\mathbb{E}}_{\mathbf{z}\sim q_\phi} -\log p(\mathbf{z}|d_{\text{obs}}) + \mathop{\mathbb{E}}_{\mathbf{z}\sim q_\phi} \log q_\phi(\mathbf{z}) \tag{4.14}$$

$$= \mathop{\mathbb{E}}_{\mathbf{z}\sim q_\phi} \mathcal{L}(\mathbf{z}) + \mathop{\mathbb{E}}_{\mathbf{z}\sim q_\phi} \log q_\phi(\mathbf{z}) \tag{4.15}$$

The first term is the expected loss under the induced distribution q_ϕ, with the loss defined in Equation (4.9). It can be approximated as

$$\mathop{\mathbb{E}}_{\mathbf{z}\sim q_\phi} \mathcal{L}(\mathbf{z}) \approx \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}(I_\phi(w_i)) \tag{4.16}$$

by sampling M realizations $w_1, \cdots, w_M$ from p_w. The second term, however, is more difficult to evaluate since we lack the analytic expression of q_ϕ. The second term is also called the (negative) entropy of q_ϕ, usually denoted $H(q_\phi) \coloneqq -\mathbb{E}_{\mathbf{z}\sim q_\phi} \log q_\phi(\mathbf{z})$. On the other hand, it is easy to obtain realizations $z_1 = I_\phi(w_1), \cdots, z_M = I_\phi(w_M)$. We therefore use a sample entropy estimator such as the Kozachenko-Leonenko estimator [108, 109],

$$\hat{H}(\{z_i, \cdots, z_M\}) = \frac{n_z}{M}\sum_{i=1}^{M} \log \rho_i + \text{const.} \tag{4.17}$$

where ρ_i is the distance between z_i and its k^{th} nearest neighbor. A good rule of thumb is $k \approx \sqrt{M}$ as reported in [109]. Thus, the entropy estimator measures how spread the sample points are.

To train the inference network I_ϕ, we minimize $\mathrm{D}_{\mathrm{KL}}(q_\phi \parallel p(\cdot|d_{\text{obs}}))$, where both the estimator and the expected loss can be differentiated with respect to ϕ using automatic differentiation algorithms. Once the inference network is trained, the conditional generator is the new neural network $G \circ I \colon \mathbb{R}^{n_w} \to \mathbb{R}^{n_y}$, i.e. the composition of the unconditional generator and the inference network, as shown in Figure 4.1. Sampling conditional realizations can then be done very efficiently by directly sampling $\mathbf{w} \sim p_w$ and forward-passing through $G \circ I$, and the parametrization of the generation process is maintained. We summarize the training steps of the inference network in Algorithm 3. Note that we show a simple gradient descent update (line 7), however it is more common to use dedicated update schemes for neural networks such as Adam [28] or RMSProp [27].

Note that since n_z is small in general, the inference network is also small and the network is easy to train relative to the generator. This also means that the relative increase in evaluation cost of the coupling $G \circ I$ is not significant. We find this to be the case in our experiments.

Algorithm 2 Inference network I_ϕ training

Require: Negative log-posterior $\mathcal{L}(z) = -\log p(z|d_{\text{obs}})$. In our case (Equation (4.11)), $\mathcal{L}(z) = \|G(z)_{\text{obs}} - d_{\text{obs}}\|^2 + \lambda\|z\|^2$, batch size M, learning rate η, source distribution p_w (usually equal to p_z).

1: **while** ϕ has not converged **do**
2: Sample $\{w_1, \cdots, w_M\} \sim p_w$
3: Get $\{z_1, \cdots, z_M\}$, $z_i = I_\phi(w_i)$
4: Get $\{\rho_1, \cdots, \rho_M\}$, $\rho_i =$ distance from z_i to its k^{th} nearest neighbor
5: $\nabla_\phi \mathbb{E}\mathcal{L} \leftarrow \frac{1}{M}\sum_{i=1}^{M} \nabla_\phi \mathcal{L}(z_i)$
6: $\nabla_\phi \hat{H} \leftarrow \frac{n_z}{M}\sum_{i=1}^{M} \nabla_\phi \log \rho_i$
7: $\phi \leftarrow \phi - \eta(\nabla_\phi \mathbb{E}\mathcal{L} - \nabla_\phi \hat{H})$
8: **end while**

4.4 Numerical experiments

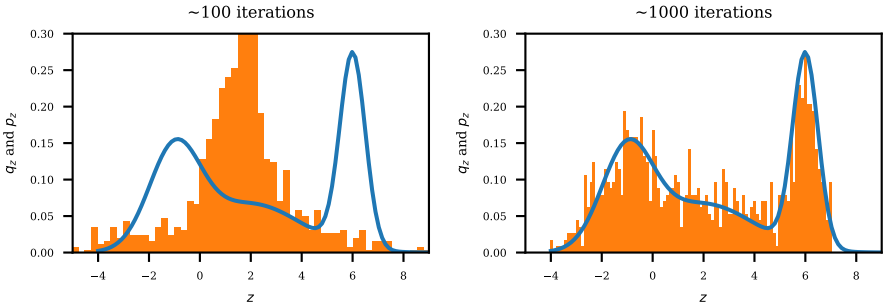
As a sanity check, we first assess the method for two simple test cases where the target distributions are mixture of Gaussians. We later present our main results for conditioning a generator that was pre-trained to generate unconditional realizations of size 64×64. All our numerical experiments are implemented using PyTorch[2] [110], a python package for automatic differentiation. The source code of our implementation is available in our repository[3]. We use the same network architecture for the inference network (except input and output sizes) in all our test cases, consisting of a fully connected network with 3 hidden layers of size 512, and leaky ReLU activation. More details are described in Section 4.6.1.

Mixture of Gaussians

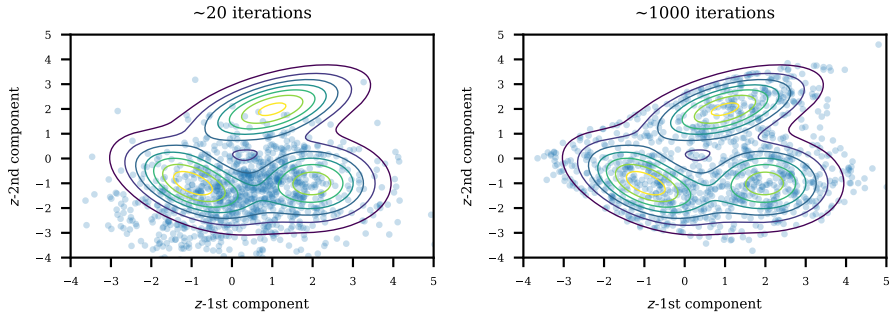
We train a neural network $I_\phi \colon \mathbb{R}^{n_w} \to \mathbb{R}^{n_z}$ to sample simple 1D and 2D mixture of Gaussians. Results are summarized in Figure 4.2, with $n_z = n_w = 1$ in the 1D case, and $n_z = n_w = 2$ in the 2D case. The source distribution p_w is the standard normal in both cases.

The first example, Figure 4.2a, is a mixture of three 1D Gaussians, with centers $\mu_1 = -1$, $\mu_2 = 2$ and $\mu_3 = 6$, and standard deviations $\sigma_1 = 1, \sigma_2 = 2, \sigma_3 = 0.5$, respectively; indicated with blue lines. The orange bars are the normalized histogram of 1000 sample points generated by the neural network. The second example, Figure 4.2b, is a mixture of three 2D Gaussians, with centers $\mu_1 = (-1, -1)$, $\mu_2 = (1, 2)$ and $\mu_3 = (2, -1)$, and covariances $\Sigma_1 = \left(\begin{smallmatrix} 1 & -0.5 \\ -0.5 & 1 \end{smallmatrix}\right)$, $\Sigma_2 = \left(\begin{smallmatrix} 1.5 & 0.6 \\ 0.6 & 0.8 \end{smallmatrix}\right)$, and $\Sigma_3 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, respectively. We plot the contour lines of the mixture of 2D Gaussians, and also scatter plot 4000 sample points

[2] https://pytorch.org/
[3] https://github.com/chanshing/geocondition



(A) Mixture of three 1D Gaussians. The blue line indicates the target distribution, and the normalized histogram corresponds to generated values.



(B) Mixture of three 2D Gaussians. The contour lines indicate the target distribution, and the scattered points correspond to generated values.

FIGURE 4.2: Results of I_ϕ trained to generate mixture of Gaussians.

generated by the inference network. In both test cases, we can see that the neural network effectively learns to transport points from the standard normal distribution to the mixture of Gaussians.

Conditional geological realizations

Our unconditional generator is a neural network $G\colon \mathbb{R}^{30} \to \mathbb{R}^{64\times 64}$ previously trained using the method of generative adversarial networks to generate unconditional realizations of 2D channelized permeability of size 64×64. The input latent vector is of size 30 with standard normal distribution. Details of the implementation is described in Section 4.6.1. Examples of unconditional realizations from the pre-trained generator is shown in Figure 4.3. Note that the conditioning method can be applied to any pre-trained generator network.

We formulate the conditional sampling problem in the Bayesian framework as described in Section 4.2.2, and train an inference network to sample the posterior $p(z|d_{\text{obs}})$. We assume $\lambda = 0.1$ in all our test cases. We use $n_w = n_z = 30$ and $p_w = p_z$ (i.e. $I_\phi\colon \mathbb{R}^{30} \to \mathbb{R}^{30}$, so that if no conditioning were present, I_ϕ should learn the identity function).

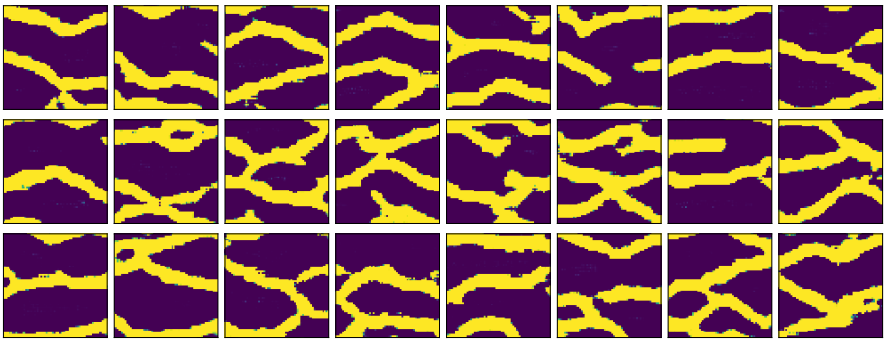


FIGURE 4.3: Unconditional realizations

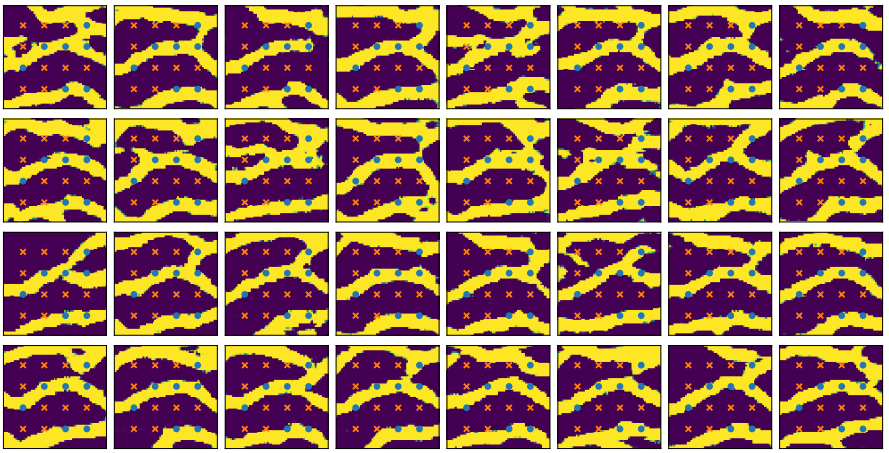
We experiment with several conditioning test cases, conditioning on the presence of channel (high permeability) or background material (low permeability) at locations in the domain. We train an inference network I_ϕ for each test case and then generate conditional realizations using the coupled network $G \circ I$. Here we use the same hyper-parameters to train the inference network in all test cases, although one could fine-tune the optimization for each test case to improve the results.

We show samples of the resulting conditional generator for two conditioning cases in Figure 4.4. We see that the generated realizations honor the conditioning points while maintaining the quality of the original generator. In Figure 4.4b, we deliberately enforce a conditioning setting to obtain a specific channel passing through the domain, and see that the generator is capable of generating multiple realizations reproducing this enforced channel while providing enough variability in the rest of the domain. This could be useful in practice when we know the presence of specific structures in the area. Additional test cases are shown in Section 4.6.2. Although not performed here, a straightforward improvement could be to adopt a safe margin by conditioning a neighborhood of the observed points.

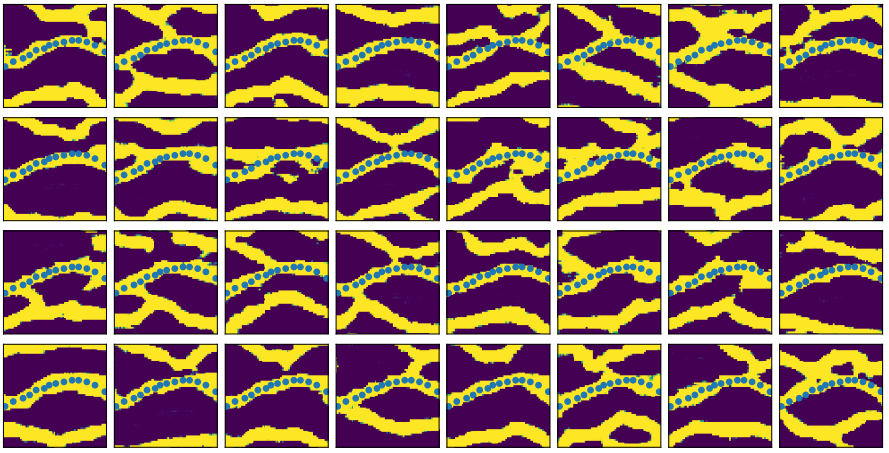
In our experiments, the inference network takes a few seconds to train for each test case using a Nvidia GeForce GTX Titan X GPU. During deployment, $G \circ I$ can generate conditional realizations at the rate of about 5500 realizations per second. We did not find noticeable increase in computational time between G and $G \circ I$. In fact, the bottleneck in the GPU was due to memory operations.

4.5 Conclusion

We presented a method to generate conditional realizations using a pre-trained unconditional generator without sacrificing the parametrization of the generation process, building upon our work in Chapter 3. The method consists of conditioning a pre-trained



(A) Examples A



(B) Examples B

FIGURE 4.4: Conditional realizations of $G \circ I$. We show two conditioning test cases. Blue dot indicates channel material (high permeability) and orange cross indicates background material (low permeability). See Section 4.6.2 for additional test cases.

unconditional generator by stacking an inference network that is trained to sample the posterior distribution of the latent vector given observations. We found the method to be very effective in generating conditional realizations in the several test cases considered, honoring the observations while also producing diverse realizations. The method is based on minimizing a Kullback-Leibler divergence and involves a sample entropy estimation. The sample entropy estimator based on the nearest neighbor ($k = 1$ in Equation (5.5)) was first applied in [111] to improve diversity in the context of neural style. In the same context, [112] used a similar estimator but based on random neighbors. Finally, in the context of generative modeling, [113] used a closed-form expression of the entropy term when using batch normalization [29]. The estimator used in this work is the generalization of the entropy estimator using k^{th} nearest neighbors introduced in [109]. Other alternatives to train neural samplers include normalizing flow [114], autoregressive flow [115], and Stein discrepancy [116]. These are all alternatives worth exploring

in future work. Also related to our work include [117, 118] where the authors optimize the latent space to condition on labels/classes.

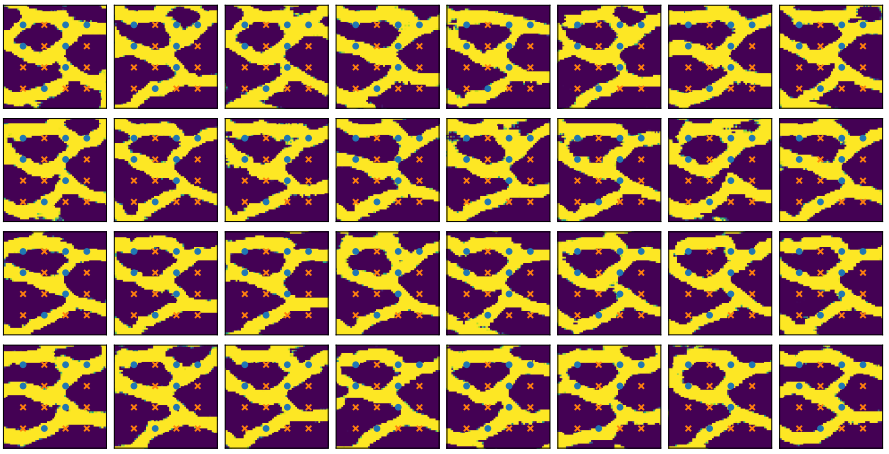
4.6 Appendix

4.6.1 Implementation details

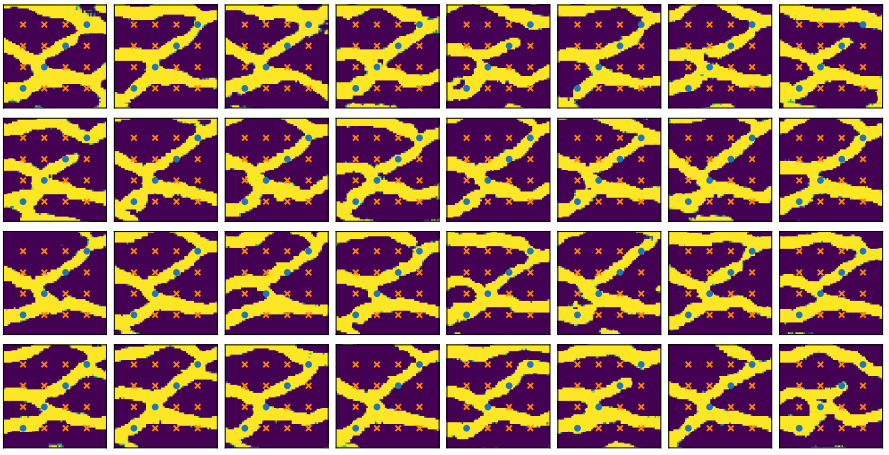
We use the same architecture for the inference network in all our experiments, namely, a fully connected network with 3 hidden layers of size 512, and component-wise leaky ReLU activation $\sigma(x) = x$ if $x > 0$, $\sigma(x) = 0.5x$ otherwise. More specifically, $I\colon \mathbb{R}^{n_w} \to \mathbb{R}^{n_z}$, $I(w) = Af_3(f_2(f_1(w)))$ where $f_i(x) = \sigma(A_i x)$, and $A_2, A_3 \in \mathbb{R}^{512\times 512}$, $A_1 \in \mathbb{R}^{512\times n_w}$, $A \in \mathbb{R}^{n_z\times 512}$. The weights $[A, A_1, A_2, A_3]$ are optimized using the gradient descent scheme Adam with learning rate $1e-4$ and default optimizer parameters ($\beta_1 = 0.5, \beta_2 = 0.999$, see [28]). We use a batch size of 64 sample points in the geological conditioning problem. In the toy problems concerning the mixture of Gaussians, we use a batch size of 256. In all test cases, the inference network converges in between 1000 and 3000 iterations.

The pre-trained generator was obtained following the work presented in Chapter 3, where a convolutional neural network was trained on a set of 1000 realizations of size 64×64 generated by the snesim algorithm [67, 119]. During deployment, both the conditional and unconditional generators generate approximately 5500 realizations per second of size 64×64 using the GPU (we do not find noticeable increase in compute time from G to $G \circ I$).

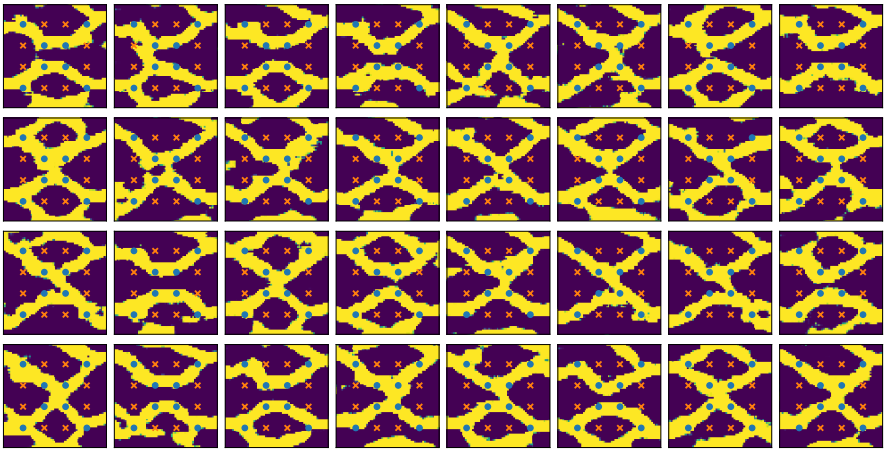
4.6.2 Additional examples



(A) Examples A

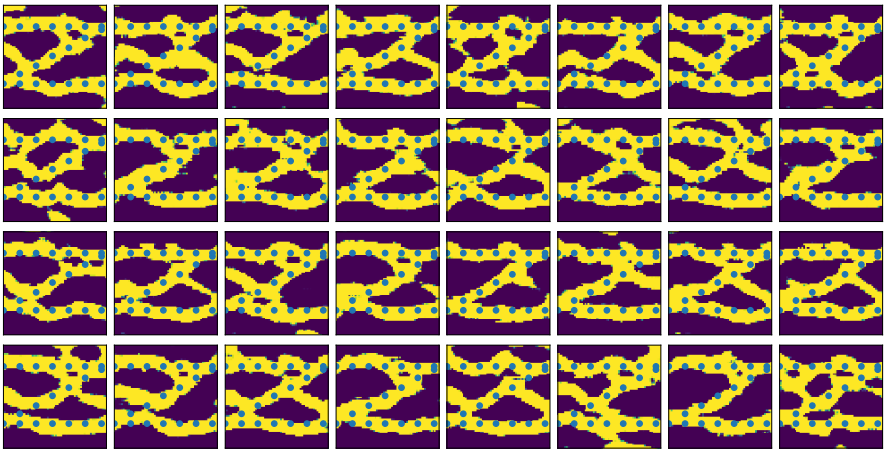


(B) Examples B

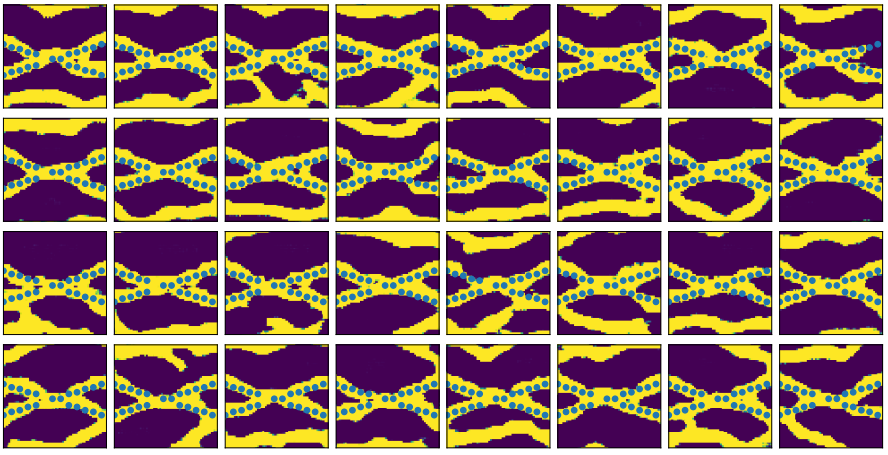


(C) Examples C

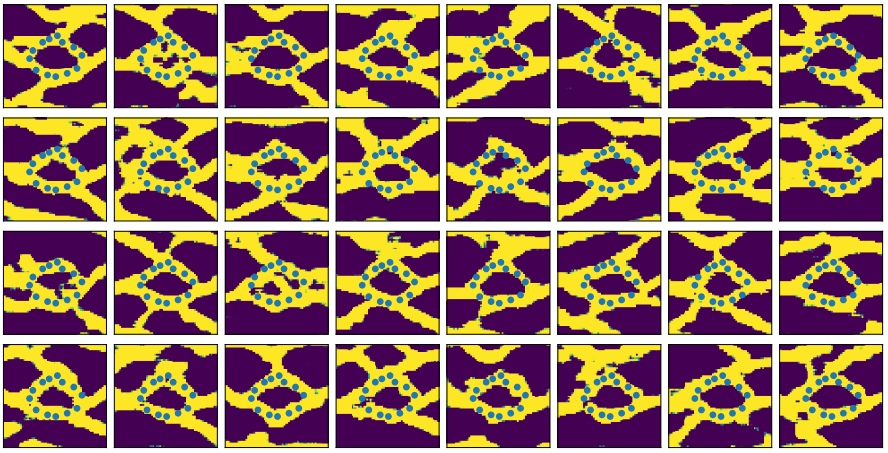
FIGURE 4.5: Additional examples (1/2)



(A) Examples A



(B) Examples B



(C) Examples C

FIGURE 4.6: Additional examples (2/2)

Chapter 5

Exemplar-based parametric synthesis of geology using kernel discrepancies and generative neural networks

We propose a framework for parametric synthesis of geological images based on an exemplar image. We synthesize new realizations such that the discrepancy in the *patch distributions* of the new realizations and the exemplar is minimized. Such discrepancy is quantified using a kernel method for two-sample test called maximum mean discrepancy. In order to obtain a parametrization of the synthesis process, we train a generative neural network to sample solutions of the minimization problem, thus providing an efficient and parametric way of generating realizations during deployment. We assess the framework on a classical benchmark binary image representing channelized subsurface reservoirs, finding that the method is effective in reproducing the visual patterns and spatial statistics (image histogram and two-point probability functions) of the exemplar image.

5.1 Introduction

A challenge in subsurface flow simulations is to obtain a complete and accurate description of subsurface properties, such as permeability and porosity, that are crucial for

In preparation for submission to journal.

accurate flow modeling. Since it is virtually impossible to obtain direct measurements at every point in the domain under study, engineers can only rely on indirect estimations of the subsurface, e.g. from seismic images and sparse measurements obtained from a few wells. Traditionally, the properties are modeled based on their two-point statistics; however, this tends to produce overly smooth Gaussian-like images of the subsurface that are far from realistic. In many scenarios, such as in channelized systems where the properties follow an almost binary distribution and contain strong spatial correlations, two-point statistics are not enough to describe the subsurface.

This shortcoming led to the development of alternative algorithmic approaches to synthesize subsurface images that can capture multipoint statistics. These methods start from an exemplar image (also called training image in the geology literature) that is deemed representative of the subsurface under study, meaning that the spatial statistics of this exemplar is believed to be similar to that of the subsurface under study. Thereafter, a new image is synthesized by querying the exemplar image or deriving statistics from it, and employing some form of randomness during the synthesis process to generate diverse outcomes. In [67], empirical conditional probabilities are derived from the exemplar and used to synthesize the new image each pixel at a time. In [72], a pixel is synthesized by simply querying the exemplar and selecting pixels whose neighboring pixels match that of the current synthesized domain. In [73], the synthesis is based on carefully copying and pasting patches extracted from the exemplar. As seen from the mentioned works, these approaches share many similarities with texture synthesis techniques in image processing [120–122]. These methods, although less theoretically grounded, tend to produce subsurface images that are much more realistic than those obtained in traditional methods based on two-point statistics.

A further challenge in subsurface flow simulations is the need to account for inherent uncertainties in the subsurface. For uncertainty quantification and history matching, it is often necessary to explore multiple plausible solutions by performing simulations for a large number of image realizations. For computational efficiency, it is also desirable that such exploration be smooth in the sense that small changes in the inputs of the simulator results in small changes in the output. For this we need a smooth parametrization of the synthesis process. This is not the case in most current synthesis algorithms where the element of stochasticity is intrinsically attached to pseudo-random number generators which are non-smooth by design. For this reason, current approaches take a two-stage process (see e.g. Chapter 3): First, a large number of realizations is synthesized using one of the many synthesis algorithms available, providing a dataset informing the patterns and variability of the subsurface; thereafter, parametrization [49, 50, 65, 123] is performed using the dataset in a way that retains the visual realism and statistics

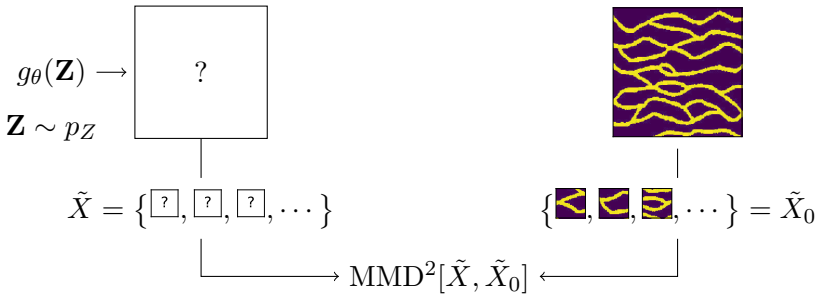


FIGURE 5.1: Overview of methodology.

of the realizations. It is then worth asking whether it is possible to achieve parametric synthesis *directly* from the exemplar.

Recent examples of parametric synthesis of geology from a single exemplar include [75, 77] where the authors train generative adversarial networks [69] on patches extracted from the exemplar image. Once trained, given that the neural networks employed are convolutional, one can artificially increase the size of the input vector to generate larger images (see discussion in Section 3.4.3). However, it remains unclear if this approach generalizes to arbitrary sizes.

In this work, we propose a new parametric synthesis method using a single exemplar image. We assume that the spatial statistics of a geological image can be sufficiently described by the distribution of the patches extracted from it. Informally, we consider two geological images A and B to be equivalent if a bag of patches extracted from A is indistinguishable from a bag of patches extracted from B. Note that this assumption is implicit in most exemplar-based synthesis algorithms. Given a reference exemplar image, a new realization is synthesized in a way that its *patch distribution* match that of the exemplar. The discrepancy in distributions is measured using a kernel method for two-sample test called *maximum mean discrepancy* (MMD) [124], which through the kernel trick is suitable for high-dimensional data. New realizations can be obtained by minimizing this discrepancy using e.g. gradient-based optimization, although such approach is slow and non-parametric. To obtain a parametrization of the synthesis process, as well as improved synthesis speed during deployment (e.g. for uncertainty quantification or history matching), a generative neural network is trained to sample solutions of the minimization problem.

The proposed method is superficially similar to those in previous chapters as well as in other works [75, 77], but is in fact substantially different: using a single exemplar image, we generate global images so as to maintain the patch distribution of the exemplar, rather than generating the patches themselves and then using the artificial expansion

trick (Section 3.4.3); moreover, the spatial statistics are derived using a kernel method. An illustration of the framework is shown in Figure 5.1.

In this study, we limit ourselves to the synthesis of unconditional realizations and leave the conditioning case for future work (examples of conditioning to hard data can be found in [78, 79] as well as our work in Chapter 4). We assess our method using the classical binary channelized image by Strebelle [67] of size 250×250, and we synthesize images of size 256×256 and 512×512. We also study the influence of different kernels in the quality of the synthesis and discuss alternatives for improvement. Although not considered here, we mention that the framework is dimension-agnostic and can be directly applied to 3D images.

The rest of this chapter is organized as follows: In Section 5.2, we describe the maximum mean discrepancy [124], and an approach to train a generative neural network [111]. Our main idea is presented in Section 5.3: new realizations are formulated as solutions to an optimization problem (minimize the discrepancy in patch distributions); thereafter, a generative neural network can be trained for fast parametric synthesis. In Section 5.4, we present results for the synthesis of binary channelized subsurface images based on the classical Strebelle exemplar image. In Section 5.5, we discuss how our framework relates to other works and potential ideas to improve this work. Finally, we state our conclusions and future directions in Section 5.6.

5.2 Background

5.2.1 Maximum mean discrepancy

Our main tool is a kernel method for two-sample test called *maximum mean discrepancy* [103, 124]. Given two samples $X = \{x_1, \cdots, x_m\}$ and $Y = \{y_1, \cdots, y_n\}$, the goal is to determine whether both samples come from the same distribution. The maximum mean discrepancy (MMD) addresses this problem by comparing the sample mean in a feature space,

$$\begin{aligned} \mathrm{MMD}^2[X, Y] &= \left\| \frac{1}{m} \sum_{i=1}^{m} \phi(x_i) - \frac{1}{n} \sum_{j=1}^{n} \phi(y_j) \right\|_{\mathcal{H}}^2 \\ &= \frac{1}{m^2} \sum_{i=1}^{m} \sum_{i'=1}^{m} k(x_i, x_{i'}) + \frac{1}{n^2} \sum_{j=1}^{n} \sum_{j'=1}^{m} k(y_j, y_{j'}) - \frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k(x_i, y_j) \end{aligned} \tag{5.1}$$

where $\phi\colon \mathcal{X} \to \mathcal{H}$ is a mapping to the feature space $\mathcal{H}$, and $k(x,y) \coloneqq \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$. The useful aspect in this formulation is that we do not need to compute $\phi(\cdot)$ – which can be infinite dimensional – as long as we can compute the function $k(\cdot,\cdot)$, called the kernel. The kernel operator can be thought of as a similarity measure, and it must satisfy certain properties in which case it is guaranteed to be associated to some feature mapping/space. For an in-depth treatment, see [124].

Examples Let $x, y \in \mathbb{R}^2$. For the linear kernel $k(x,y) = x^T y$, an associated feature map is $\phi(x) = x$, then the MMD is simply the difference in the sample mean. For the polynomial kernel of degree two $k(x,y) = (x^T y + 1)^2$, an associated feature map is $\phi(x) = (x_{(2)}^2, x_{(1)}^2, \sqrt{2}x_{(2)}x_{(1)}, \sqrt{2}x_{(2)}, \sqrt{2}x_{(1)}, 1)$ where $x = (x_{(1)}, x_{(2)})$, then the MMD captures differences in both mean and covariance. Finally, the Gaussian radial basis function $k(x,y) = \exp\{-\gamma||x-y||^2\}$ is associated with a mapping to infinitely many components (obtained by Taylor expansion) and the corresponding MMD captures all the moments of the distribution.

5.2.2 Generative neural network

We now describe a method to train a generative neural network as proposed in [111]. Let $g_\theta\colon \mathcal{Z} \to \mathcal{X}$ be a neural network parametrized by weights θ to be determined. The input to the neural network are realizations of a random *latent vector* $\mathbf{Z} \sim p_Z$ that provides the source of stochasticity so that $g_\theta(\mathbf{Z})$ is a stochastic simulator. The distribution p_Z is a design choice and is usually an easy-to-sample distribution (e.g. standard normal distribution) so that the cost of sampling $g_\theta(\mathbf{Z})$ is mostly given by the evaluation cost of g_θ. Given a target probability density function p, and a fixed p_Z, the goal is to determine θ such that $g_\theta(\mathbf{Z}) \sim p$. Let us denote the density of $g_\theta(\mathbf{Z})$ under θ by q_θ. The Kullback-Leibler (KL) divergence from p to q_θ is

$$\mathrm{D}_{\mathrm{KL}}(q_\theta \parallel p) = \mathop{\mathbb{E}}_{\mathbf{X}\sim q_\theta} \log \frac{q_\theta(\mathbf{X})}{p(\mathbf{X})} \tag{5.2}$$

$$= \mathop{\mathbb{E}}_{\mathbf{X}\sim q_\theta} -\log p(\mathbf{X}) + \mathop{\mathbb{E}}_{\mathbf{X}\sim q_\theta} \log q_\theta(\mathbf{X}) \tag{5.3}$$

We therefore wish to find θ that minimizes this divergence. The first term of the sum can be approximated as

$$\mathop{\mathbb{E}}_{\mathbf{X}\sim q_\theta} -\log p(\mathbf{X}) \approx \frac{1}{N}\sum_{i=1}^{N} -\log p(X_i) \tag{5.4}$$

where $X_i = g_\theta(Z_i)$, by drawing N realizations of $\mathbf{Z} \sim p_Z$. The second term of the sum, called the (negative) entropy, is problematic since we do not have the analytic form of the density q_θ (g_θ normally contains multiple non-linearities). We circumvent this issue by using a sample entropy estimator [108, 109] over a set of generated realizations $\{X_1, \cdots, X_N\}$,

$$\mathop{\mathbb{E}}_{\mathbf{X}\sim q_\theta} \log q_\theta(\mathbf{X}) \approx -\hat{H}(\{X_i, \cdots, X_N\}) \coloneqq -\frac{1}{N}\sum_{i=1}^{N} c \log \rho(X_i) + \text{const.} \tag{5.5}$$

where c is the number of components of $\mathbf{X}$, and $\rho(X_i)$ is the distance from X_i to its k^{th}-nearest neighbor (with $k \approx \sqrt{N}$ as a good rule of thumb [109]). Essentially, $\hat{H}$ quantifies how spread the realizations are. Putting all together, Equation (5.2) can be approximated as

$$\mathrm{D}_{\mathrm{KL}}(q_\theta \parallel p) \approx \frac{1}{N}\sum_{i=1}^{N} -\log p(X_i) - \frac{1}{N}\sum_{i=1}^{N} c \log \rho(X_i) + \text{const.} \tag{5.6}$$

Minimizing this expression can be done using gradient-based optimization, where the gradients with respect to θ can be obtained using automatic differentiation algorithms. Intuitively, the first term ensures that the generated samples are in the regions of high probability of p, whereas the second term ensures that the samples are diverse.

5.3 Methodology

We denote by X an image realization and $\tilde{X} = \{x_1, \cdots, x_m\}$ the corresponding set of patches extracted from X. Given a reference exemplar image X_0, we aim to match the *patch distribution* inferred by the set $\tilde{X}_0$. Concretely, for a new realization X, the sets $\tilde{X}$ and $\tilde{X}_0$ must be indistinguishable in the sense that their distributions are close. For example, matching the distribution of "1×1 patches" reduces to matching the pixel histogram of the exemplar image. For patches of size $l_1 \times l_2$, the distribution to be matched is given by the multidimensional joint histogram of $l_1 l_2$ variables. The discrepancy in distributions is measured using the maximum mean discrepancy (MMD, Section 5.2.1), which is suitable for high-dimensional distributions via the kernel trick. New realizations can then be obtained by solving a minimization problem,

$$\arg\min_X \mathrm{MMD}^2[\tilde{X}, \tilde{X}_0] \tag{5.7}$$

with MMD^2 defined in Equation (5.1). Note that a patch x_i of an image X is the result of a projection operator, therefore the optimization can be done using gradient-based

methods. Multiple realizations can be obtained by using a local optimizer and different initial guesses for X.

Optimization-based synthesis, however, can be expensive if a large number of realizations is continuously required in the online phase (e.g. for uncertainty quantification or history matching); moreover, it does not provide a smooth parametric way to explore the solution space. We therefore train a generator in an offline phase to sample solutions of the minimization problem efficiently online. We train the generator following the approach described in Section 5.2.2, which requires us to define a target density p. To this end, we adopt a Gibbs distribution model $p(X) \propto \exp\{-\frac{1}{\lambda}\mathcal{L}(X)\}$ where $\mathcal{L}(X) \coloneqq \mathrm{MMD}^2[\tilde{X}, \tilde{X}_0]$ and λ is an unknown "temperature" constant (see [125] or Section 4.1 of [68] for a justification of this choice). This conveniently sets the KL divergence in Equation (5.2) to

$$\mathrm{D}_{\mathrm{KL}}(q_\theta \parallel p) \propto \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(X_i) - \frac{\lambda}{N}\sum_{i=1}^{N} c \log \rho(X_i) \tag{5.8}$$

where we multiplied everything by λ and omitted the irrelevant constants. The first term ensures that the samples minimize the MMD, while the second term ensures that the samples are diverse. Since we do not know the constant λ, in this work we treat it as a hyperparameter to be tuned in the offline training. In practice, λ acts as the trade-off between sample quality and diversity. We summarize the steps to train the generator in Algorithm 3.

Algorithm 3 Generator training g_θ

Require: Exemplar image X_0, kernel $k(\cdot,\cdot)$ of MMD, "temperature" λ, source distribution p_Z, batch size N.

1: **while** θ has not converged **do**
2: Sample $\{Z_1, \cdots, Z_N\} \sim p_Z$
3: Obtain $\{X_1, \cdots, X_N\}$, $X_i = g_\theta(Z_i)$
4: $\mathbb{E}\mathcal{L} \leftarrow \frac{1}{N}\sum_{i=1}^{N} \mathrm{MMD}^2[\tilde{X}_i, \tilde{X}_0]$ ▷ Equation (5.1)
5: $\lambda\hat{H} \leftarrow \frac{\lambda}{N}\sum_{i=1}^{N} c \log \rho(X_i)$
6: $\theta \leftarrow \mathrm{Update}(\theta; \nabla_\theta(\mathbb{E}\mathcal{L} - \lambda\hat{H}))$
7: **end while**

5.3.1 Kernel choice

As in other kernel methods, the kernel choice is critical in the performance of the MMD; specifically, it defines its discriminative power. For characteristic kernels [126, 127], the MMD can distinguish two distributions in the infinite setting [124]. These include the Gaussian radial basis function, the Laplace kernel, and the rational quadratic kernel. In this work, we use the rational quadratic kernel $k_{\mathrm{rq}}(x, y) = (1 + \frac{||x-y||^2}{2\alpha l^2})^{-\alpha}$ due to its

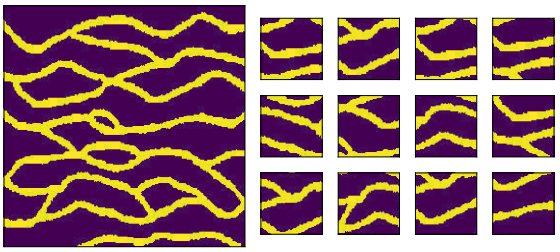


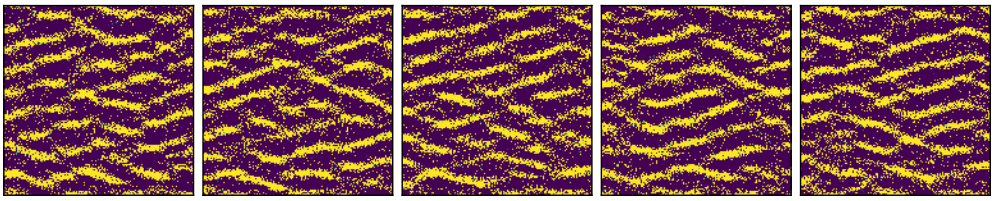
FIGURE 5.2: Exemplar image (by Strebelle [67]) of size 250×250 depicting subsurface channels (left), and a few patches of size 64×64 extracted from the image (right).

better gradient behavior, where α and l are hyperparameters to be tuned during the offline phase (see Section 4.2 of [128] for properties of this and other kernels).

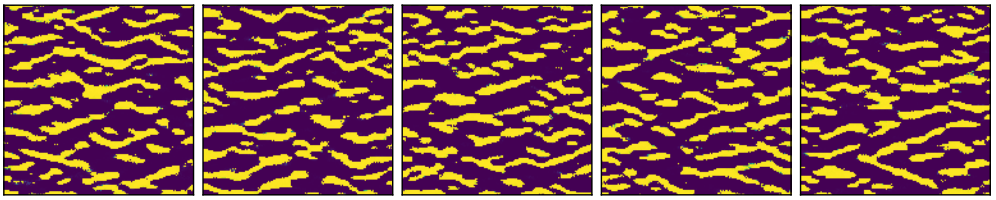
Measuring similarities using kernels, however, can be challenging when the data is very high dimensional [129]. Moreover, distance-based kernels (as functions of $||x - y||$) are not well-suited when applied on the raw pixel representation of images, since differences in pixel values are of little meaning in conveying similarity (e.g. small shifts in pixels would imply large differences while remaining virtually the same). On the other hand, it is often the case that the intrinsic dimensionality of the data is low, albeit embedded in a high dimensional space. For example, geological structures of interest such as channels can be accurately described regardless of the grid resolution, once it is above certain threshold. This suggests us to first project the data to a low dimensional space, e.g. using principal component analysis or even random projections [130], before applying the distance-based kernel. In this work, we use the encoder of an autoencoder trained on patches of the exemplar. The autoencoder [131] is a generalization of principal component analysis using a non-linear combination of basis functions (represented by neural networks). The idea here is to measure distances between patches using their code representations instead of their raw pixel representations. The resulting kernel is $k(\cdot, \cdot) = k_{\text{rq}}(h(\cdot), h(\cdot))$ where $h(\cdot)$ denotes the encoder (note that $k_{\text{rq}}(h(\cdot), h(\cdot))$ *is* a kernel).

5.4 Numerical experiments

We consider the synthesis of geological realizations containing subsurface channels using the classical exemplar image of Strebelle [67] shown in Figure 5.2. Note that our target distribution is discrete (the image is binary); nevertheless, we found good results using a continuous framework. For convenience, we pre-process the image and work in the $[-1, 1]$ range, so that -1 represents the background material (blue) and 1 represents the channel material (yellow). The size of the exemplar image is 250×250, and we use patches of size 64×64. Naturally, the patch size has to be large enough to capture the



(A) $k_{\text{rq,randproj}}$



(B) $k_{\text{rq,pca}}$

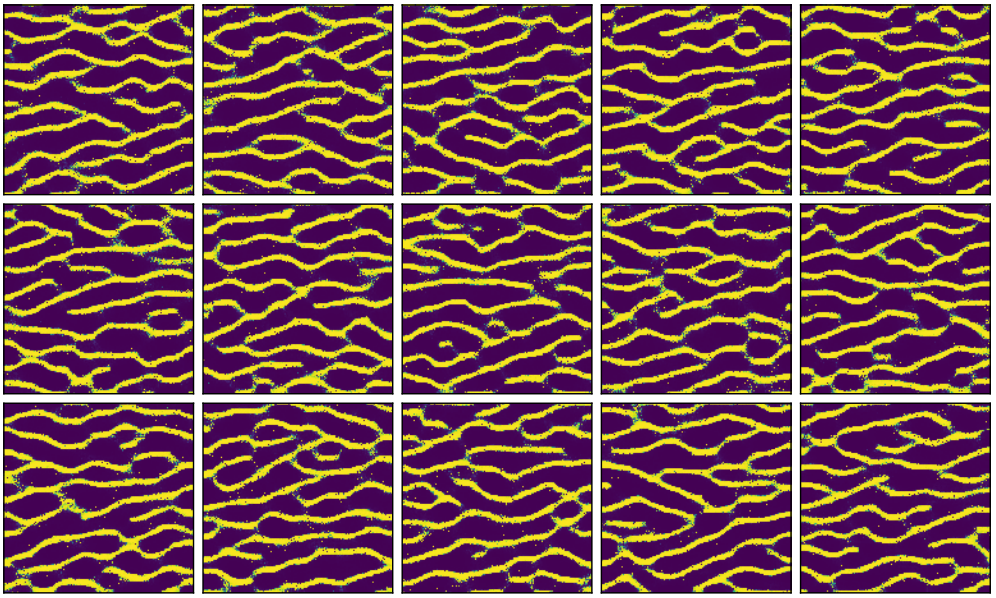
FIGURE 5.3: Optimization-based synthesis using different kernels.

relevant patterns of interest in the exemplar image; however, it should not be too large since this determines the amount and variability of patches given that our exemplar is of finite size in practice. We synthesize images of size 256×256 and 512×512.

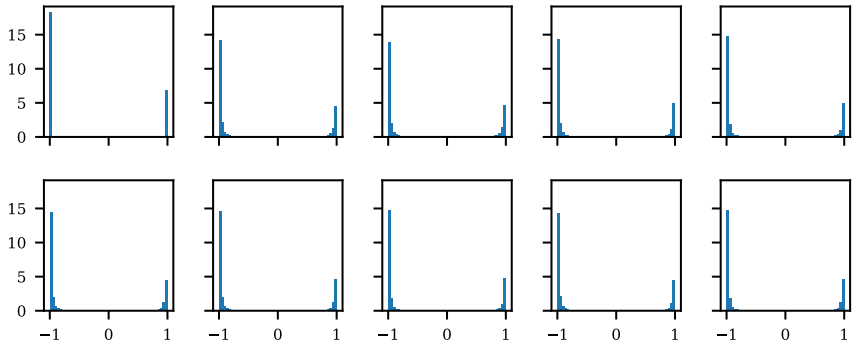
For the MMD, we use a kernel of the form $k(\cdot,\cdot) = k_{\text{rq}}(h(\cdot), h(\cdot))$ where h is a mapping to a lower dimensional space, and k_{rq} is the rational quadratic kernel $k_{\text{rq}}(x,y) = 1 + \frac{||x-y||^2}{2\alpha l^2})^{-\alpha}$. We use $\alpha = 0.5$, and for l (length scale parameter) we use a median heuristic [124]: we use the median distance between the patches in the combined sample – note that this means that our kernel adapts during the training iterations. As for h, we experiment with three choices: a random projection matrix [130], principal component analysis (PCA) trained on patches of the exemplar, and the encoder of an autoencoder trained on patches of the exemplar.

Note that the MMD in Equation (5.1) has a quadratic cost with respect to the sample size (although linear estimates exist [124]) making it expensive to evaluate in the whole set of patches. Since we compute the MMD iteratively, we instead evaluate on a random subset of patches drawn during the iterations. We draw a subset of 128 patches. As a consequence, we found that this procedure tends to undersample patches at the boundary of the domain, so we perform reflection padding on the synthesis domain equal to half a patch width before sampling patches – this may introduce some biases at the synthesis boundaries.

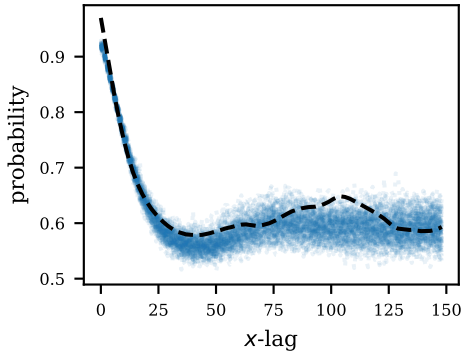
Our implementation is done using Pytorch [110], a python package for automatic differentiation.



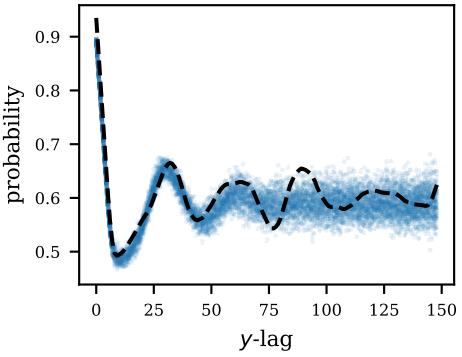
(A) Random realizations (256 × 256, optimization-based synthesis).



(B) Image histogram of 9 random realizations. The first histogram (top left) corresponds to the exemplar image.



(C) Two-point probability in the x direction of 100 realizations.



(D) Two-point probability in the y direction of 100 realizations.

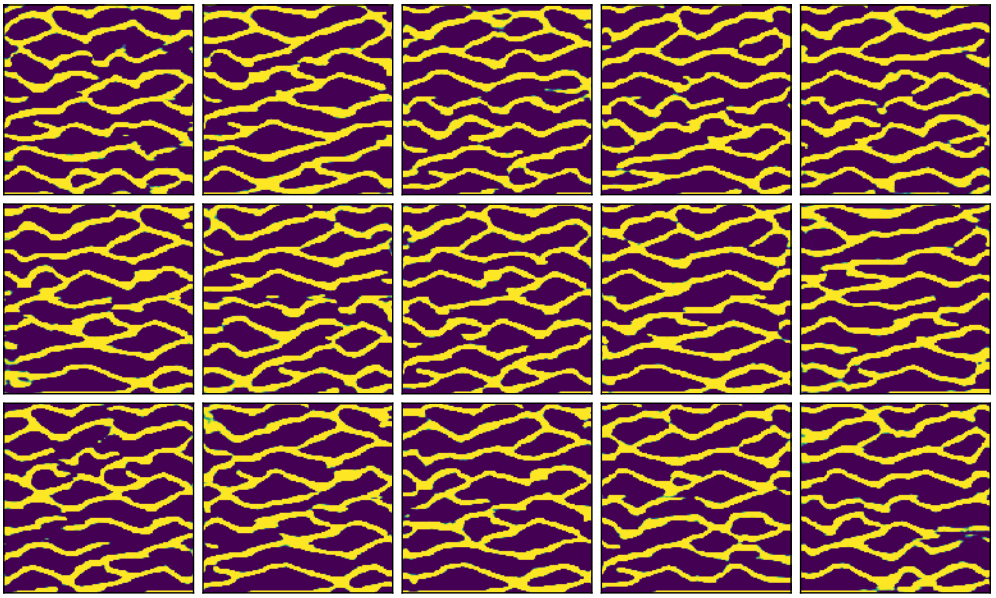
FIGURE 5.4: Results for *optimization-based* synthesis of realizations of size 256 × 256 with $k_{\text{rq,encoder}}$ kernel.

5.4.1 Optimization-based synthesis

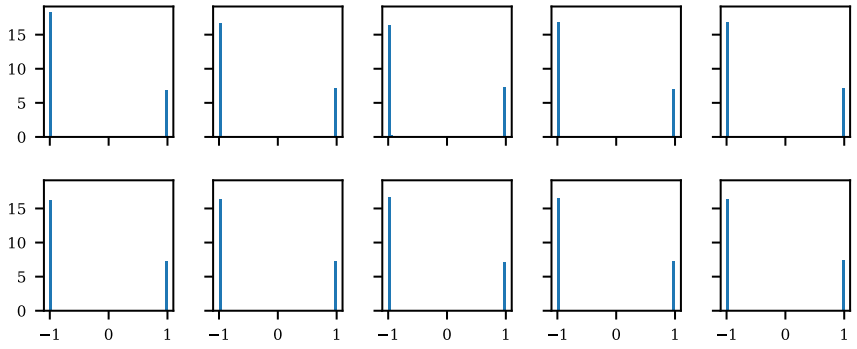
We start by synthesizing realizations using an optimization approach (Equation (5.7)). Since the pixel values are bounded in $[-1, 1]$, rather than using a constrained optimization method, here we simply reparametrize the pixels by $X = \tanh(X')$ and solve for X' instead. We use the Adam optimizer [28, 132] (a variant of stochastic gradient descent). We test different kernels for the MMD: For k_{rq} + random projection ($k_{\text{rq,randproj}}$), we use a low-rank random matrix to project each 64×64 patch to a vector of 512 components. For k_{rq} + principal component analysis (PCA) ($k_{\text{rq,pca}}$), we project each patch to 64 eigencomponents (retaining over 75% of the variance). Synthesis results for size 256×256 are shown in Figures 5.3a and 5.3b. We can see that both random projection and PCA kernels already capture key spatial statistics of the exemplar such as the horizontal correlations defining the channels and the correct pixel values; however, the visual quality of the realizations are still rather poor.

Next, we use the encoder of an autoencoder trained on the patches of the exemplar image ($k_{\text{rq,encoder}}$). The autoencoder is trained to encode each patch into a small code vector of size 8, a number found via experimentation. We experimentally found that smaller codes tend to produce better results (as long as the autoencoder can be trained successfully), presumably by making the distance-based kernel more accurate. Details of the autoencoder implementation are described in Section 5.7.1.

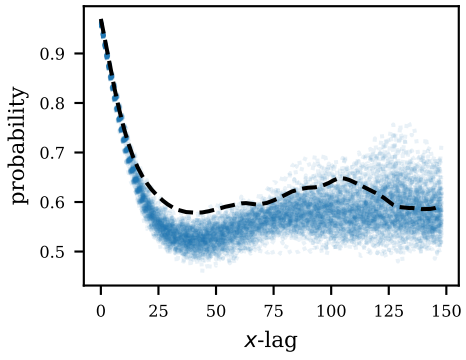
Synthesis results for size 256×256 using the k_{rq} + encoder kernel are summarized in Figure 5.4. Compared to the previous kernels, we see that the visual quality of the realizations are significantly improved, highlighting the impact of the kernel choice. The synthesized images, however, still contain some spurious values such as isolated pixels that the optimization did not manage to remove within the iterations. If required, these could be removed using one of many available image post-processing methods [133]. We show in Figure 5.4b the normalized histogram of pixel values of nine random realizations without thresholding, finding good correspondence with the exemplar histogram. We show the two-point probability functions (PF) [92] in the horizontal and vertical directions in Figures 5.4c and 5.4d, respectively. The dashed black lines indicate the PFs of the exemplar image, and the dotted blue lines are PFs for 100 random realizations. We do perform thresholding in this evaluation to compute the PFs. To compute the PFs on the realizations, we randomly crop a region of size 250×250 from each realization in order to match the exemplar size, and compute the PF in this region. Note that before cropping, we first perform a reflection padding as used in the optimization to reduce potential biases at the boundaries. Overall, we find good agreement between the synthesized images and the exemplar. We show additional results for synthesis of size 512×512 in Section 5.7.2.



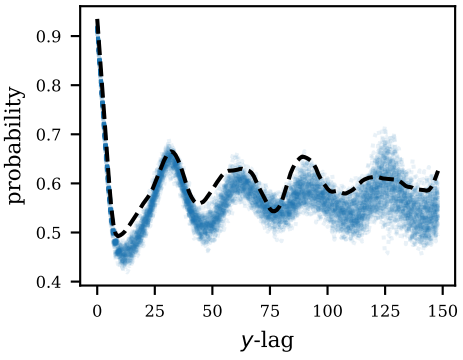
(A) Random realizations (256×256, generated by neural network).



(B) Image histogram of 9 random realizations. The first histogram (top left) corresponds to the exemplar image.



(C) Two-point probability in the x direction of 100 realizations.



(D) Two-point probability in the y direction of 100 realizations.

FIGURE 5.5: Results for *neural synthesis* of realizations of size 256×256 with $k_{\text{rq,encoder}}$ kernel.

5.4.2 Neural synthesis

We next train a generative neural network to synthesize realizations efficiently. Here we only consider the kernel with the encoder of the autoencoder ($k_{\text{rq,encoder}}$). The generator is a convolutional neural network designed following the template provided in [83], which works well for most computer vision tasks. Details of the architecture are given in Section 5.7.1. To synthesize 256×256 images, the generator $g_\theta \colon \mathbb{R}^{256} \to \mathbb{R}^{256 \times 256}$ maps from realizations of a latent vector of size 256 sampled from the standard normal distribution, to image realizations of size 256×256. The size of the latent vector was chosen using a simple heuristic: proportional to the number of non-overlapping patches in the synthesis domain times the encoding size. We train g_θ to minimize the KL divergence in Equation (5.8), where we use a batch size of $N = 4$ and temperature hyperparameter $\lambda = 10^{-8}$. We found that a good initial guess for λ is a number such that the value of the first and second terms in the KL (expected loss and entropy, respectively) stay within the same order of magnitude in the latter iterations of the training, so that the KL is eventually allowed to go to zero. In fact, here we tuned λ from $\{10^{-7}, 10^{-8}, 10^{-9}\}$, although finer tuning is encouraged.

Results of the neural synthesis are summarized in Figure 5.5. Notably, we find that the results using neural synthesis are visually better, e.g. we do not find isolated pixels as in the optimization approach. This can be explained by the locality prior imposed by the convolutional architecture [134, 135]: since the image is parametrized by a neural network, updates in the weights of the neural network affects a whole neighborhood of the output image, in contrast to optimization in the pixel space where pixels are updated individually; moreover, this influence is hierarchical due to the convolutional architecture, since layers closer to the output have a more local influence while layers closer to the input affect the output more globally. Regarding the normalized image histogram (again without thresholding) in Figure 5.5b, we find that it more closely matches the true binary shape of the exemplar histogram. Finally, we show the two-point probability functions for the neural synthesis (computed as in the previous section) in Figures 5.5c and 5.5d. We find a slight bias in the trend of the curves, which may suggest that further tuning of the neural network is necessary. Nonetheless, the results remain close in relative value.

We additionally train a generator $g_\theta \colon \mathbb{R}^{512} \to \mathbb{R}^{512 \times 512}$ to synthesize realizations of size 512×512. For this case, we use a latent vector of size 512 (also with standard normal distribution) and $\lambda = 10^{-9}$. The results are summarized in Section 5.7.2.

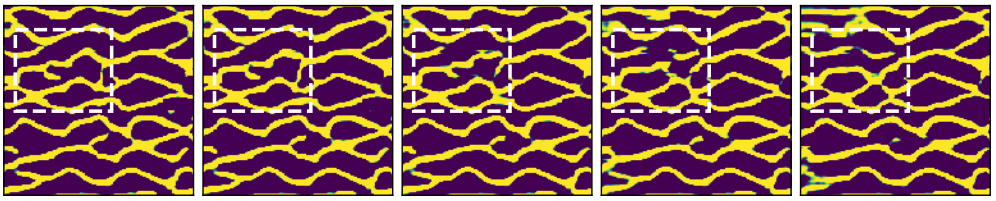
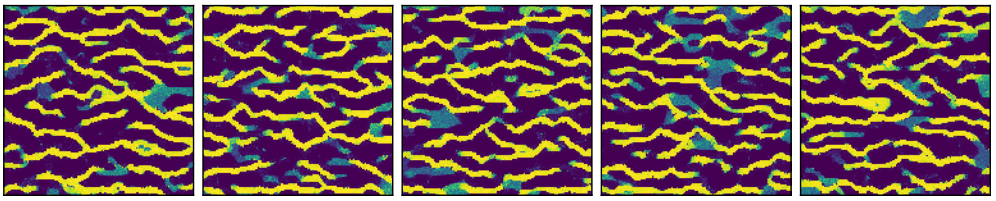


FIGURE 5.6: Linear interpolation of one coordinate of the latent vector.

Smooth transitions Since g_θ is continuous by construction, small changes in the input results in small changes in the output. We verify this in Figure 5.6 where we show the outputs of the generator of size 256×256. Starting from an initial random realization of the latent vector $\mathbf{Z}$, we linearly vary one of its coordinates while fixing the remaining coordinates. Note that unlike methods such as principal component analysis where the latent vector represents the coefficients of the eigenvectors, the latent vector of generative neural networks lacks interpretability. Learning interpretable latent vectors is an ongoing area of research, see e.g. [136].

5.5 Related work

Neural kernels The seminal work in [137] showed that it is possible to synthesize textures from an exemplar by matching statistics of feature responses of a pre-trained neural network evaluated on the exemplar. Briefly, the exemplar is fed into the VGG-net [81] – a very large neural network trained on natural images for classification – and a matrix is formed containing the correlations of feature responses at layers of the neural network. Then, new realizations are synthesized such that their corresponding matrices are close to that of the exemplar. It was later shown in [138] that this is equivalent to computing the maximum mean discrepancy on the feature responses using the polynomial kernel $k(x, y) = (x^T y)^2$. Finally, by noting that each feature response corresponds to a patch of the domain (defined by its receptive field), we conclude that this is an instance of our framework where the kernel is composed of a polynomial kernel and the VGG-net as "encoder". Note that in this case, the encoder is trained on a different task (classification) using large sets of other images, making the approach an example of transfer learning [139, 140]. We show synthesis results using this kernel in Figures 5.7a and 5.7b for our geological image and for a natural texture (peppers; first image in the row), respectively. We see that the kernel performs very well for the image of peppers, but not so well for our binary geological image – presumably because the VGG-net is trained on natural color images.



(A) $k_{\text{poly,vgg}}$

(B) $k_{\text{poly,vgg}}$

FIGURE 5.7: Optimization-based synthesis using the kernel from [137], i.e. VGG encoder + polynomial kernel of second degree. Compare (a) with Figure 5.3a, Figure 5.3b, and Figure 5.4a.

Neural generators The present approach to train a generator is based on [111] where a sample entropy estimator based on the nearest neighbor is used. Here we use a k^{th} nearest neighbor [109] estimation which we found to be numerically more stable. These estimators, however, measure the distance between realizations in the raw representation, which for images may not be well suited. An ad-hoc alternative can be found in [112] where distances are instead computed on the feature responses of a neural network evaluated on the images. Other alternatives include normalizing flow [114], autoregressive flow [115], and Stein variational gradient descent [141]. The latter is an interesting alternative which involves yet another kernel to estimate the average diversity in the sample, making it useful for embedding prior knowledge about the geology.

Adaptive kernels The kernel has a big influence on the quality of the synthesis since it defines the discriminative power of the MMD. In this work, we first reduce the data using a fixed encoder of an autoencoder previously trained on patches of the exemplar image. The same approach is employed in [142] in the context of generative modeling of natural images [104]. This is done following the intuition that distances in the code representation of an autoencoder are more suitable than in the raw pixel representation of images and spatial data. This manual kernel engineering can be circumvented by considering adaptive kernels [143]. The idea here is to iteratively update the kernel encoder during the training iterations, thus serving as an adversary maximizing the MMD whereas the generator is trained to minimize it. This idea can be taken further by considering other functions aside from kernels, e.g. parametrized by neural networks [69]. All these methods involve adversarial training of an additional neural network as well

as dynamic target loss functions, involving numerical challenges in terms of stability as well as computational cost. On the upside, they tend to produce state-of-the-art results in computer vision.

5.6 Conclusion

We introduced a parametric synthesis method for geological images that consists of preserving the patch distribution of a reference exemplar image. An energy function was defined that measures the discrepancy in the patch distributions of new realizations with respect to the exemplar. Parametrization was achieved by training a generative neural network to solve the energy minimization problem. Contrary to the parametrization presented in previous chapters, this parametrization requires only a single exemplar image for training, and it is trained to generate global images rather than patches of the exemplar. We assessed the framework using the classical exemplar image by Strebelle of size 250×250, and synthesized images of sizes 256×256 and 512×512. We found that with an adequate kernel, the visual patterns from the exemplar image are clearly reproduced, and the spatial statistics as measured by the image histogram and the two-point probability functions show very good agreement with respect to the exemplar. Our framework depends on the discriminative power of the MMD, which is highly influenced by the kernel choice, as verified in the comparison using different kernels. Also, the generator is currently trained using a sample entropy estimator that is based on distances in pixel space, which might not be ideal for spatial data. We discussed possible future alternatives to the current approach such as adaptive kernels to avoid manual kernel tuning, and the Stein discrepancy to train the generator.

5.7 Appendix

5.7.1 Implementation details

Autoencoder

The architecture of the autoencoder is designed based on the template provided in [83]: The encoder $h\colon \tilde{\mathcal{X}} \to \mathbb{R}^8$ is a chain of convolutions with leaky ReLU activations, with tanh activation in the last layer. The decoder $d\colon \mathbb{R}^8 \to \tilde{\mathcal{X}}$ is a chain of transposed convolutions with ReLU activations, also with tanh in the final layer. The code size is 8. The architecture is detailed in Table 5.1. We train to minimize $\arg\min_{d,h} ||x - d(h(x))||^2$ on patches of the exemplar image. To reduce overfitting, we data-augment by performing

State size	Layer
$1 \times 64 \times 64$	`Conv(4,2,1), BN, lReLU`
$32 \times 32 \times 32$	`Conv(4,2,1), BN, lReLU`
$64 \times 16 \times 16$	`Conv(4,2,1), BN, lReLU`
$128 \times 8 \times 8$	`Conv(4,2,1), BN, lReLU`
$256 \times 4 \times 4$	`Conv(4,1,0), Tanh`
$8 \times 1 \times 1$	–

(A) Encoder

State size	Layer
$8 \times 1 \times 1$	`ConvT(4,1,0), BN, ReLU`
$256 \times 4 \times 4$	`ConvT(4,2,1), BN, ReLU`
$128 \times 8 \times 8$	`ConvT(4,2,1), BN, ReLU`
$64 \times 16 \times 16$	`ConvT(4,2,1), BN, ReLU`
$32 \times 32 \times 32$	`ConvT(4,2,1), Tanh`
$1 \times 64 \times 64$	–

(B) Decoder

TABLE 5.1: Autoencoder architecture. `Conv/ConvT`=convolution/transposed convolution, the triplet indicates (filter size, stride, padding), `BN`=batch normalization.

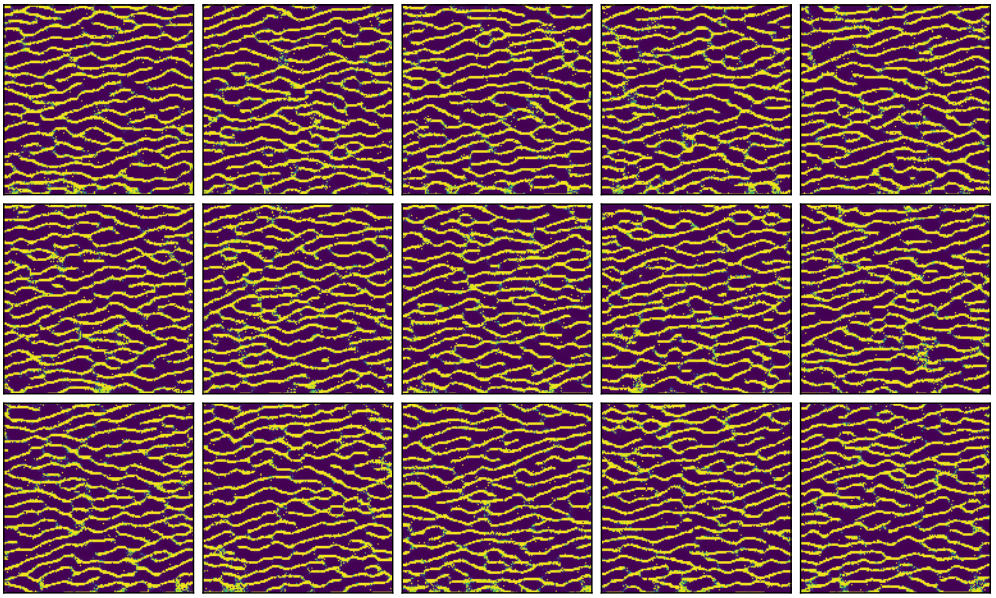
horizontal and vertical flips on the patches, as well as smoothing by adding a small amount of Gaussian noise (with 0.05 standard deviation). We use the Adam optimizer with default parameters and learning rate 10^{-3}, and train for 2000 iterations using a batch size of 32. The model takes a few seconds to train using a GTX Titan X. The decoder is discarded after training and we keep the encoder for the MMD kernel.

Generator

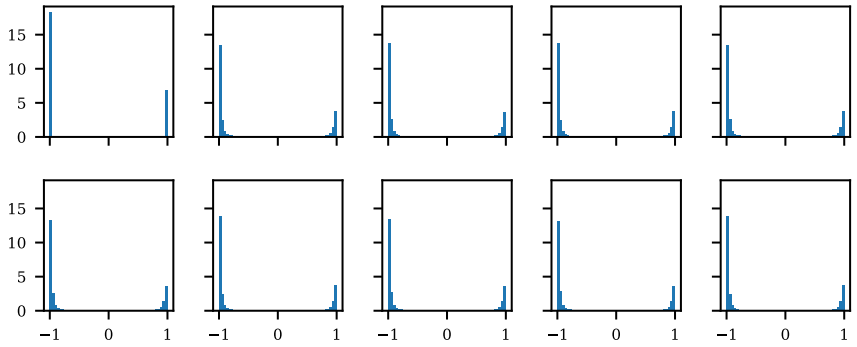
The generator is also designed based on the template provided in [83], but we replace most of the transposed convolutions with upsampling + convolution (motivated by [144]), and add an additional convolving layer before the output. Specifically, the transposed convolutions are replaced by a $\times 2$ nearest neighbor upsampling followed by a convolution. The activation in the last layer is tanh. The architecture is detailed in Table 5.2. We use the Adam optimizer with default parameters and learning rate 10^{-3}, and train for 50,000 iterations for both the 256×256 and 512×512 generators. Using a GTX Titan X, training takes about 2.5 and 5 hours for sizes 256×256 and 512×512, respectively [2]. In the online phase, the generators can synthesize images at the rate of approximately 150/s and 50/s for sizes 256×256 and 512×512, respectively.

5.7.2 Additional results

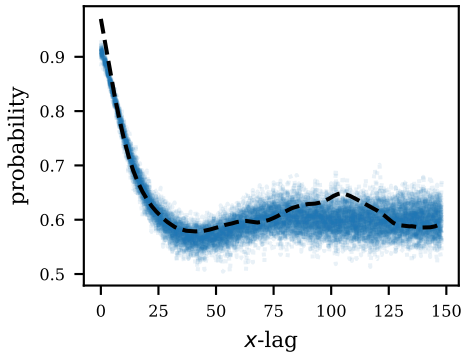
[2] The training is slow in our current implementation due to the way the patches are being extracted.



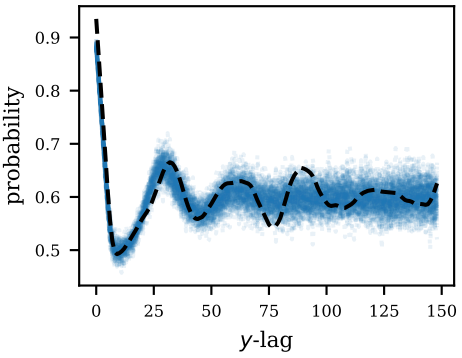
(A) Random realizations (512 × 512, optimization-based synthesis).



(B) Image histogram of 9 random realizations. The first histogram (top left) corresponds to the exemplar image.

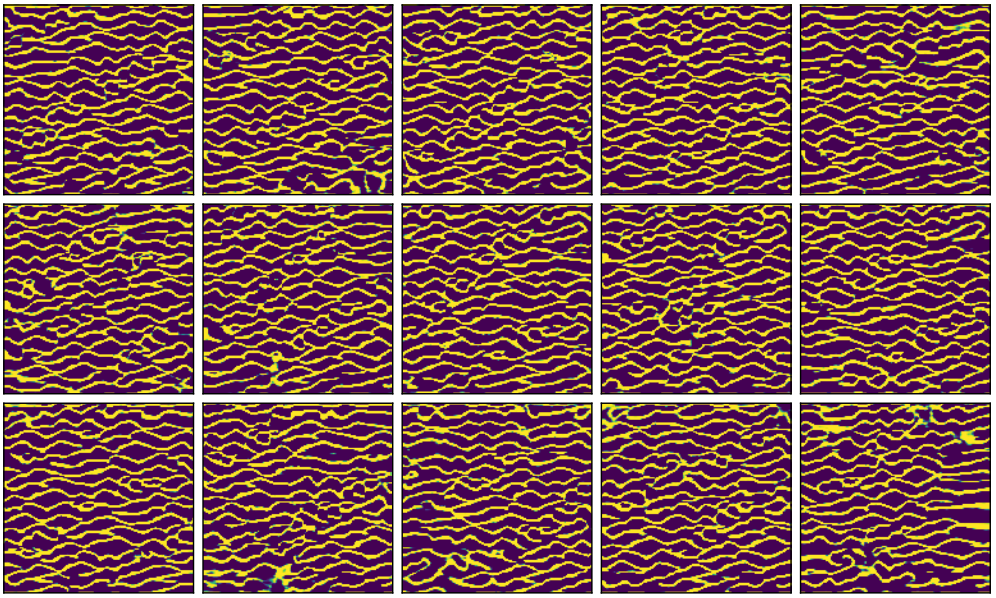


(C) Two-point probability in the x direction of 100 realizations.

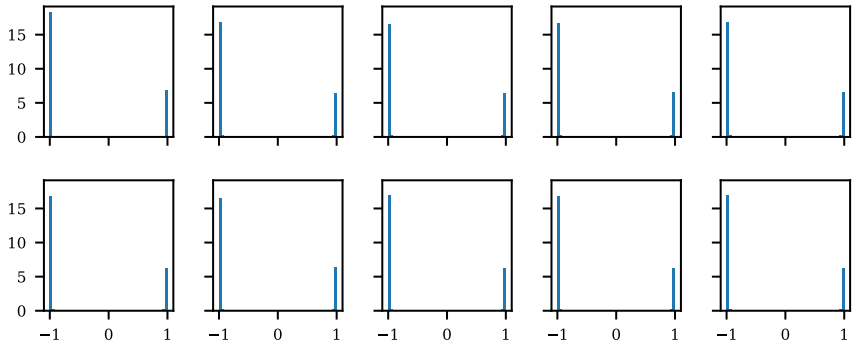


(D) Two-point probability in the y direction of 100 realizations.

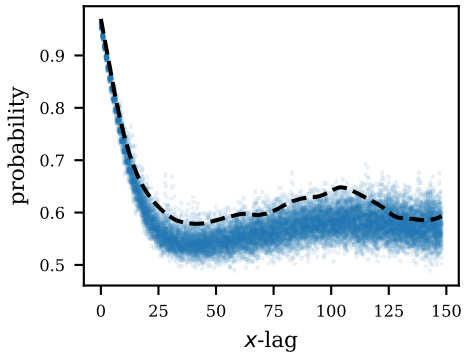
FIGURE 5.8: Results for *optimization-based synthesis* of realizations of size 512 × 512 with $k_{\text{rq,encoder}}$ kernel.



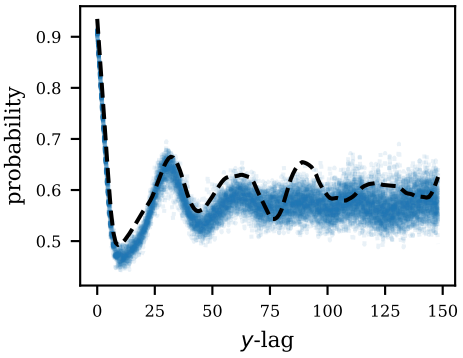
(A) Random realizations (512×512, generated by neural network).



(B) Image histogram of 9 random realizations. The first histogram (top left) corresponds to the exemplar image.



(C) Two-point probability in the x direction of 100 realizations.



(D) Two-point probability in the y direction of 100 realizations.

FIGURE 5.9: Results for *neural synthesis* of realizations of size 512×512 with $k_{\text{rq,encoder}}$ kernel.

State size	Layer
$256 \times 1 \times 1$	`ConvT(4,1,0), BN, ReLU`
$2048 \times 4 \times 4$	`UpConv(3,1,1), BN, ReLU`
$1024 \times 8 \times 8$	`UpConv(3,1,1), BN, ReLU`
$512 \times 16 \times 16$	`UpConv(3,1,1), BN, ReLU`
$256 \times 32 \times 32$	`UpConv(3,1,1), BN, ReLU`
$128 \times 64 \times 64$	`UpConv(3,1,1), BN, ReLU`
$64 \times 128 \times 128$	`UpConv(3,1,1), BN, ReLU`
$64 \times 256 \times 256$	`Conv(3,1,1), Tanh`
$1 \times 256 \times 256$	–

(A) 256×256 generator.

State size	Layer
$512 \times 1 \times 1$	`ConvT(4,1,0), BN, ReLU`
$4096 \times 4 \times 4$	`UpConv(3,1,1), BN, ReLU`
$2048 \times 8 \times 8$	`UpConv(3,1,1), BN, ReLU`
$1024 \times 16 \times 16$	`UpConv(3,1,1), BN, ReLU`
$512 \times 32 \times 32$	`UpConv(3,1,1), BN, ReLU`
$256 \times 64 \times 64$	`UpConv(3,1,1), BN, ReLU`
$128 \times 128 \times 128$	`UpConv(3,1,1), BN, ReLU`
$64 \times 256 \times 256$	`UpConv(3,1,1), BN, ReLU`
$64 \times 512 \times 512$	`Conv(3,1,1), Tanh`
$1 \times 512 \times 512$	–

(B) 512×512 generator.

TABLE 5.2: Generator architecture. `UpConv`=×2 upsample + convolution, `ConvT`=transposed convolution, the triplet indicates (filter size, stride, padding), `BN`=batch normalization.

Chapter 6

Final conclusions

In this thesis, we explored the application of machine learning techniques to enhance uncertainty quantification tasks in reservoir simulation. In Chapter 2, we used supervised learning techniques to accelerate subsurface flow simulations. In Chapters 3 to 5, we focused on improving the simulation results through parametrization using recent unsupervised learning techniques. In the following, we recapitulate the main findings in our study along with possible extensions.

Chapter 2: A machine learning approach for efficient uncertainty quantification using multiscale methods

We performed an image-to-image regression using neural networks to surrogate the computation of localized elliptic problems inside the multiscale finite volume method that are needed to obtain the custom basis functions. This resulted in a speedup of two orders of magnitude in obtaining the basis functions without compromising the simulation results; in fact, for the test cases of the experiment, the results were indistinguishable for the estimated quantities of interest. Possible directions to extend this work include the assessment in more general permeability fields, and studies in multiphase flow.

Chapter 3: Parametrization of stochastic inputs using generative adversarial networks with application in geology

A parametrization based on deep convolutional neural networks was considered to capture the challenging spatial patterns of typical geological models. The parametrization was able to reproduce realizations with very high visual quality which were sometimes indistinguishable from data. More importantly, the parametrization reproduced the reference flow statistics in an uncertainty propagation study – in particular, it was able to match complex multimodal distributions

very well. The parametrization also showed very good results for inversion in parameter estimation, always providing reasonable and plausible realizations even for challenging target images. For limited datasets, we found that the discriminator size needs to be tuned to avoid overfitting. Finally, we verified well-known issues with the standard formulation of GAN such as mode collapse and training instability, and found the Wasserstein formulation to be better suited for our application. Possible directions to extend this work include improving current GAN methods for limited data, and further assessments in other test cases.

Chapter 4: Parametric generation of conditional geological realizations using generative neural networks

We built upon the work in the previous chapter and considered the problem of post-hoc conditioning of a pre-trained generator. The idea was to generate realizations conditioned on new gathered observations using an already trained generator, putting emphasis on parametric generation without having to train a generator from scratch. We used a simple approach that consists of training a small neural network to sample from the Bayesian posterior distribution of the latent vector. This small neural network is later stacked to the original generator, thus obtaining a conditional generator and maintaining the parametrization. This simple approach obtained very good results where the conditional generator was able to honor the conditioning for a variety of test cases considered, maintaining the visual quality and producing diverse realizations. Finally, we discussed several other alternatives to our current approach that differ in the way that the inference network could be obtained.

Chapter 5: Exemplar-based parametric synthesis of geology using kernel discrepancies and generative neural networks

We introduced a different approach to obtain a parametrization of the geology. In contrast to the parametrization considered in the previous chapters that required a large dataset of realizations to inform the patterns and variability of the subsurface, here we used a single exemplar image and derived a parametrization that is based on preserving the distribution of the patches of the exemplar. The presumably high-dimensional distribution is captured using a kernel method that is suitable in high dimensions. We assessed the method using the benchmark Strebelle image of size 250×250 and synthesized images of size 256×256 and 512×512, finding very good performance in both cases for preserving the spatial statistics as given by the image histogram and two point probability functions, as well as producing realizations with high visual quality. We found the method to be sensitive to the kernel choice, as it happens with any kernel method, and discussed a future

alternative based on adaptive kernels. We also indicated the Stein discrepancy as a future alternative to train the neural network that might be more suited for spatial data.

6.1 Remarks and future directions

In our study, we placed a lot of emphasis on neural network models. This was motivated by the high expressive power of neural networks as well as recent advances in machine learning – and computer vision in particular, assisted by current trends where we see increasing data availability and computing resources. Computer vision was of special interest to our study due to many similarities in this field, containing several interesting ideas that could be further investigated – specifically, we argue that many techniques in pattern recognition could be leveraged in geomodeling. More generally, the field of machine learning and subsurface reservoir engineering both share many of the same challenges such as dealing with high-dimensional spatial data (whether they are natural photos or geological images) with non-linear features, unknown or intractable likelihoods and distributions, and sampling complex and high-dimensional distributions. The main difference resides in the availability of data, which dictates the research directions in each field. For example, the emphasis in computer vision is on models for natural images for which the Internet serves as an unlimited source of data. In contrast, the amount of real geological images are still rather limited and expensive to obtain. As a consequence, recent techniques in computer vision can afford extremely large neural network models without much concern about data availability. Another difference is driven by the intended applications. For example, a method for recommender systems puts a lot of emphasis on performance and less emphasis on the interpretability of the model. In contrast, critical engineering applications often require a level of interpretability of the predictive model that is not easily provided in deep neural network models. These are important challenges that are worth addressing to facilitate the wide adoption of machine learning in engineering.

Bibliography

[1] Omnicore Agency. Youtube by the numbers: Stats, demographics & fun facts. https://www.omnicoreagency.com/youtube-statistics/, 2018.

[2] Internet World Stats. World internet users statistics and 2016 world population stats. https://www.internetworldstats.com/stats.htm, 2017.

[3] Siri Team. Hey siri: An on-device dnn-powered voice trigger for apples personal assistant. https://machinelearning.apple.com/2017/10/01/hey-siri.html, 2017.

[4] Google AI. Google duplex: An ai system for accomplishing real-world tasks over the phone. https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html, 2018.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, and Jiakai Zhang. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[7] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, and Adrian Bolton. Mastering the game of go without human knowledge. *Nature*, 550(7676): 354, 2017.

[8] OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

[9] Insights by Stanford Business. Andrew ng: Why ai is the new electricity. http://stanford.io/2mwODQU, 2017.

[10] AI Index. Artificial Inteligence Index, 2017 Annual Report. http://aiindex.org/2017-report.pdf, 2017.

[11] Google Cloud Platform Blog. Google supercharges machine learning tasks with TPU custom chip. https://cloudplatform.googleblog.com/2016/05/Google-supercharges-machine-learning-tasks-with-custom-chip.html, 2016.

[12] Phoebe MR DeVries, Fernanda Viégas, Martin Wattenberg, and Brendan J Meade. Deep learning of aftershock patterns following large earthquakes. *Nature*, 560 (7720):632, 2018.

[13] Angel Cruz-Roa, Hannah Gilmore, Ajay Basavanhally, Michael Feldman, Shridar Ganesan, Natalie NC Shih, John Tomaszewski, Fabio A González, and Anant Madabhushi. Accurate and reproducible invasive breast cancer detection in whole-slide images: A deep learning approach for quantifying tumor extent. *Scientific reports*, 7:46450, 2017.

[14] TGS. TGS Salt Identification Challenge. https://www.kaggle.com/c/tgs-salt-identification-challenge, 2018.

[15] Thomas Strohmer. Surprises in high dimensions. https://www.math.ucdavis.edu/~strohmer/courses/180BigData/180lecture1.pdf, 2017.

[16] P Jenny, SH Lee, and HA Tchelepi. Multi-scale finite-volume method for elliptic problems in subsurface flow simulation. *Journal of Computational Physics*, 187 (1):47–67, 2003.

[17] Olav Møyner and Knut-Andreas Lie. A multiscale method based on restriction-smoothed basis functions suitable for general grids in high contrast media. In *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers, 2015.

[18] Jørg E Aarnes and Yalchin Efendiev. Mixed multiscale finite element methods for stochastic porous media flows. *SIAM Journal on Scientific Computing*, 30(5): 2319–2339, 2008.

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[20] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

[21] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[22] Ivan Lunati and Patrick Jenny. Multiscale finite-volume method for compressible multiphase flow in porous media. *Journal of Computational Physics*, 216(2):616–636, 2006.

[23] Ivan Lunati and Patrick Jenny. Multiscale finite-volume method for density-driven flow in porous media. *Computational Geosciences*, 12(3):337–350, 2008.

[24] Kevin Jarrett, Koray Kavukcuoglu, and Yann Lecun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153. IEEE, 2009.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

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

[27] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

[28] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[30] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

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

[32] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.

[33] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011.

[34] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[35] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

[36] Robert Tibshirani. A comparison of some error estimates for neural network models. *Neural Computation*, 8(1):152–163, 1996.

[37] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

[38] P. Jacquard. Permeability distribution from field pressure data. *Society of Petroleum Engineers*, Dec 1965. doi: 10.2118/1307-PA.

[39] H. O. Jahns. A rapid method for obtaining a two-dimensional reservoir description from well pressure response data. *Society of Petroleum Engineers*, Dec 1966. doi: 10.2118/1473-PA.

[40] Robert Bissell. Calculating optimal parameters for history matching. In *ECMOR IV-4th European Conference on the Mathematics of Oil Recovery*, 1994.

[41] Guy Chavent and Robert Bissell. Indicator for the refinement of parameterization. In *Inverse problems in engineering mechanics*, pages 309–314. Elsevier, 1998.

[42] AA Grimstad, T Mannseth, G Nævdal, and H Urkedal. Scale splitting approach to reservoir characterization. In *SPE reservoir simulation symposium*. Society of Petroleum Engineers, 2001.

[43] Alv-Arne Grimstad, Trond Mannseth, Sigurd Ivar Aanonsen, Ivar Aavatsmark, Alberto Cominelli, and Stefano Mantica. Identification of unknown permeability trends from history matching of production data. In *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers, 2002.

[44] Alv-Arne Grimstad, Trond Mannseth, Geir Nævdal, and Hege Urkedal. Adaptive multiscale permeability estimation. *Computational Geosciences*, 7(1):1–25, 2003.

[45] Sigurd Ivar Aanonsen. Efficient history matching using a multiscale technique. In *SPE reservoir simulation symposium*. Society of Petroleum Engineers, 2005.

[46] GR Gavalas, PC Shah, and John H Seinfeld. Reservoir history matching by bayesian estimation. *Society of Petroleum Engineers Journal*, 16(06):337–350, 1976.

[47] Dean S Oliver. Multiple realizations of the permeability field from well test data. *SPE Journal*, 1(02):145–154, 1996.

[48] Albert C Reynolds, Nanqun He, Lifu Chu, and Dean S Oliver. Reparameterization techniques for generating reservoir descriptions conditioned to variograms and well-test pressure data. *SPE Journal*, 1(04):413–426, 1996.

[49] Pallav Sarma, Louis J Durlofsky, and Khalid Aziz. Kernel principal component analysis for efficient, differentiable parameterization of multipoint geostatistics. *Mathematical Geosciences*, 40(1):3–32, 2008.

[50] Xiang Ma and Nicholas Zabaras. Kernel principal component analysis for stochastic input model generation. *Journal of Computational Physics*, 230(19):7311–7331, 2011.

[51] Hai X Vo and Louis J Durlofsky. Regularized kernel PCA for the efficient parameterization of complex geological models. *Journal of Computational Physics*, 322: 859–881, 2016.

[52] Mehrdad Gharib Shirangi. History matching production data and uncertainty assessment with an efficient tsvd parameterization algorithm. *Journal of Petroleum Science and Engineering*, 113:54–71, 2014.

[53] Mehrdad G Shirangi and Alexandre A Emerick. An improved tsvd-based levenberg–marquardt algorithm for history matching and comparison with gauss–newton. *Journal of Petroleum Science and Engineering*, 143:258–271, 2016.

[54] Reza Tavakoli and Albert Coburn Reynolds. History matching with parametrization based on the svd of a dimensionless sensitivity matrix. In *SPE reservoir simulation symposium*. Society of Petroleum Engineers, 2009.

[55] Reza Tavakoli and Albert C Reynolds. Monte carlo simulation of permeability fields and reservoir performance predictions with svd parameterization in rml compared with enkf. *Computational Geosciences*, 15(1):99–116, 2011.

[56] Stephane G Mallat. Multiresolution approximations and wavelet orthonormal bases of ^{2} (). *Transactions of the American mathematical society*, 315(1):69–87, 1989.

[57] Pengbo Lu and Roland N Horne. A multiresolution approach to reservoir parameter estimation using wavelet analysis. In *SPE annual technical conference and exhibition*. Society of Petroleum Engineers, 2000.

[58] Isha Sahni and Roland N Horne. Multiresolution wavelet analysis for improved reservoir description. *SPE Reservoir Evaluation & Engineering*, 8(01):53–69, 2005.

[59] Behnam Jafarpour and Dennis B. McLaughlin. Efficient permeability parameterization with the discrete cosine transform. *Society of Petroleum Engineers*, February 2007. doi: 10.2118/106453-MS.

[60] Behnam Jafarpour and Dennis B. McLaughlin. Reservoir characterization with the discrete cosine transform. *Society of Petroleum Engineers*, March 2009. doi: 10.2118/106453-PA.

[61] Behnam Jafarpour, Vivek K. Goyal, Dennis B. McLaughlin, and William T. Freeman. Compressed history matching: Exploiting transform-domain sparsity for regularization ofnonlinear dynamic data integration problems. *Mathematical Geosciences*, 42(1):1–27, Jan 2010. ISSN 1874-8953. doi: 10.1007/s11004-009-9247-z. URL https://doi.org/10.1007/s11004-009-9247-z.

[62] David Moreno and Sigurd Ivar Aanonsen. Stochastic facies modelling using the level set method. In *EAGE Conference on Petroleum Geostatistics*, 2007.

[63] Oliver Dorn and Rossmary Villegas. History matching of petroleum reservoirs using a level set technique. *Inverse Problems*, 24(3):035015, 2008. URL http://stacks.iop.org/0266-5611/24/i=3/a=035015.

[64] Haibin Chang, Dongxiao Zhang, and Zhiming Lu. History matching of facies distribution with the enkf and level set parameterization. *Journal of Computational Physics*, 229(20):8011 – 8030, 2010. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2010.07.005. URL http://www.sciencedirect.com/science/article/pii/S0021999110003748.

[65] Mohammadreza Mohammad Khaninezhad, Behnam Jafarpour, and Lianlin Li. Sparse geologic dictionaries for subsurface flow model calibration: Part i. inversion formulation. *Advances in Water Resources*, 39:106–121, 2012.

[66] Mohammadreza Mohammad Khaninezhad, Behnam Jafarpour, and Lianlin Li. Sparse geologic dictionaries for subsurface flow model calibration: Part ii. robustness to uncertainty. *Advances in water resources*, 39:122–136, 2012.

[67] Sebastien B Strebelle and Andre G Journel. Reservoir modeling using multiple-point statistics. In *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers, 2001.

[68] Gregoire Mariethoz and Jef Caers. *Multiple-point geostatistics: stochastic modeling with training images*. John Wiley & Sons, 2014.

[69] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[70] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3381–3387. IEEE, 2008.

[71] Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

[72] Gregoire Mariethoz, Philippe Renard, and Julien Straubhaar. The direct sampling method to perform multiple-point geostatistical simulations. *Water Resources Research*, 46(11), 2010.

[73] Pejman Tahmasebi, Ardeshir Hezarkhani, and Muhammad Sahimi. Multiple-point geostatistical modeling based on the cross-correlation functions. *Computational Geosciences*, 16(3):779–797, 2012.

[74] Shing Chan and Ahmed H Elsheikh. Parametrization and generation of geological models with generative adversarial networks. *arXiv preprint arXiv:1708.01810*, 2017.

[75] Lukas Mosser, Olivier Dubrule, and Martin J Blunt. Reconstruction of three-dimensional porous media using generative adversarial neural networks. *arXiv preprint arXiv:1704.03225*, 2017.

[76] Lukas Mosser, Olivier Dubrule, and Martin J Blunt. Stochastic reconstruction of an oolitic limestone by generative adversarial networks. *arXiv preprint arXiv:1712.02854*, 2017.

[77] Eric Laloy, Romain Hérault, Diederik Jacques, and Niklas Linde. Efficient training-image based geostatistical simulation and inversion using a spatial generative adversarial neural network. *arXiv preprint arXiv:1708.04975*, 2017.

[78] Emilien Dupont, Tuanfeng Zhang, Peter Tilke, Lin Liang, and William Bailey. Generating realistic geology conditioned on physical measurements with generative adversarial networks. *arXiv preprint arXiv:1802.03065*, 2018.

[79] Lukas Mosser, Olivier Dubrule, and Martin J Blunt. Conditioning of three-dimensional generative adversarial networks for pore and reservoir-scale models. *arXiv preprint arXiv:1802.05622*, 2018.

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[81] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[82] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[83] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[84] Andrej Karpathy. CS231n convolutional neural networks for visual recognition. http://cs231n.github.io/convolutional-networks/.

[85] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[86] David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[87] Guo-Jun Qi. Loss-sensitive generative adversarial networks on lipschitz densities. *arXiv preprint arXiv:1701.06264*, 2017.

[88] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. How to train your dragan. *arXiv preprint arXiv:1705.07215*, 2017.

[89] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.

[90] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv preprint arXiv:1701.07875*, 2017.

[91] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5769–5779, 2017.

[92] S. Torquato and G. Stell. Microstructure of twophase random media. i. the npoint probability functions. *The Journal of Chemical Physics*, 77(4):2071–2077, 1982. doi: 10.1063/1.444011.

[93] Raymond Yeh, Chen Chen, Teck Yian Lim, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with perceptual and contextual losses. *arXiv preprint arXiv:1607.07539*, 2016.

[94] Jean Marçais and Jean-Raynald de Dreuzy. Prospective interest of deep learning for hydrological inference. *Groundwater*, 55(5):688–692, 2017.

[95] J Nagoor Kani and Ahmed H Elsheikh. Dr-rnn: A deep residual recurrent neural network for model reduction. *arXiv preprint arXiv:1709.00939*, 2017.

[96] Hector Klie. Physics-based and data-driven surrogates for production forecasting. In *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers, 2015.

[97] Valentin G Stanev, Filip L Iliev, Scott Hansen, Velimir V Vesselinov, and Boian S Alexandrov. Identification of release sources in advection–diffusion system by machine learning combined with greens function inverse method. *Applied Mathematical Modelling*, 60:64–76, 2018.

[98] Wenyue Sun and Louis J Durlofsky. A new data-space inversion procedure for efficient uncertainty quantification in subsurface flow problems. *Mathematical Geosciences*, 49(6):679–715, 2017.

[99] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 2018.

[100] Manuel Valera, Zhengyang Guo, Priscilla Kelly, Sean Matz, Adrian Cantu, Allon G Percus, Jeffrey D Hyman, Gowri Srinivasan, and Hari S Viswanathan. Machine learning for graph-based representations of three-dimensional discrete fracture networks. *arXiv preprint arXiv:1705.09866*, 2017.

[101] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. Generalization and equilibrium in generative adversarial nets (gans). *arXiv preprint arXiv:1703.00573*, 2017.

[102] Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

[103] Arthur Gretton, Karsten M Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *Advances in neural information processing systems*, pages 513–520, 2007.

[104] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

[105] Youssef Mroueh and Tom Sercu. Fisher gan. In *Advances in Neural Information Processing Systems*, pages 2510–2520, 2017.

[106] Youssef Mroueh, Chun-Liang Li, Tom Sercu, Anant Raj, and Yu Cheng. Sobolev gan. *arXiv preprint arXiv:1711.04894*, 2017.

[107] Youssef Mroueh, Tom Sercu, and Vaibhava Goel. Mcgan: Mean and covariance feature matching gan. *arXiv preprint arXiv:1702.08398*, 2017.

[108] LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987.

[109] Mohammed Nawaz Goria, Nikolai N Leonenko, Victor V Mergel, and Pier Luigi Novi Inverardi. A new class of random vector entropy estimators and its applications in testing statistical hypotheses. *Journal of Nonparametric Statistics*, 17(3): 277–297, 2005.

[110] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[111] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *Proc. CVPR*, 2017.

[112] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Diversified texture synthesis with feed-forward networks. In *Proc. CVPR*, 2017.

[113] Taesup Kim and Yoshua Bengio. Deep directed generative models with energy-based probability estimation. *arXiv preprint arXiv:1606.03439*, 2016.

[114] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[115] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in Neural Information Processing Systems*, pages 4743–4751, 2016.

[116] Dilin Wang and Qiang Liu. Learning to draw samples: With application to amortized mle for generative adversarial learning. *arXiv preprint arXiv:1611.01722*, 2016.

[117] Anh Nguyen, Jason Yosinski, Yoshua Bengio, Alexey Dosovitskiy, and Jeff Clune. Plug & play generative networks: Conditional iterative generation of images in latent space. *arXiv preprint arXiv:1612.00005*, 2016.

[118] Jesse Engel, Matthew Hoffman, and Adam Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. *arXiv preprint arXiv:1711.05772*, 2017.

[119] N Remy, A BOUCHER, and J WU. Sgems: Stanford geostatistical modeling software. *Software Manual*, 2004.

[120] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *iccv*, page 1033. IEEE, 1999.

[121] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346. ACM, 2001.

[122] Gregoire Mariethoz and Sylvain Lefebvre. Bridges between multiple-point geostatistics and texture synthesis: Review and guidelines for future research. *Computers & Geosciences*, 66:66–80, 2014.

[123] Hai X Vo and Louis J Durlofsky. A new differentiable parameterization based on principal component analysis for the low-dimensional representation of complex geological models. *Mathematical Geosciences*, 46(7):775–813, 2014.

[124] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[125] Ying Nian Wu, Song Chun Zhu, and Xiuwen Liu. Equivalence of julesz and gibbs texture ensembles. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1025–1032. IEEE, 1999.

[126] Bharath K Sriperumbudur, Arthur Gretton, Kenji Fukumizu, Bernhard Schölkopf, and Gert RG Lanckriet. Hilbert space embeddings and metrics on probability measures. *Journal of Machine Learning Research*, 11(Apr):1517–1561, 2010.

[127] Bharath K Sriperumbudur, Kenji Fukumizu, and Gert RG Lanckriet. Universality, characteristic kernels and rkhs embedding of measures. *Journal of Machine Learning Research*, 12(Jul):2389–2410, 2011.

[128] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[129] Aaditya Ramdas, Sashank Jakkam Reddi, Barnabás Póczos, Aarti Singh, and Larry A Wasserman. On the decreasing power of kernel and distance based nonparametric hypothesis tests in high dimensions. In *AAAI*, pages 3571–3577, 2015.

[130] Ella Bingham and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250. ACM, 2001.

[131] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[132] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. 2018.

[133] Mehmet Sezgin and Bülent Sankur. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic imaging*, 13(1): 146–166, 2004.

[134] Andrew M Saxe, Pang Wei Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *ICML*, pages 1089–1096, 2011.

[135] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. *arXiv preprint arXiv:1711.10925*, 2017.

[136] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[137] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[138] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer. *arXiv preprint arXiv:1701.01036*, 2017.

[139] Lorien Y Pratt. Discriminability-based transfer between neural networks. In *Advances in neural information processing systems*, pages 204–211, 1993.

[140] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[141] Yihao Feng, Dilin Wang, and Qiang Liu. Learning to draw samples with amortized stein variational gradient descent. *arXiv preprint arXiv:1707.06626*, 2017.

[142] Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727, 2015.

[143] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2203–2213, 2017.

[144] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016. doi: 10.23915/distill.00003. URL http://distill.pub/2016/deconv-checkerboard.