MSc in Mathematical Engineering

Course: Numerical Analysis for Partial Differential Equations

Lecturer: Prof Paola F. Antonietti
Teaching Assistant: Dr. F. Regazzoni
Tutor: Dr. C.B. Leimer Saglio

Type B project

A.Y. 2023/2024

Variational Autoencoders to solve Diffusion Inverse Problems

Authors:

Gabriele Bruni
Riccardo Rota
Nicola Zanetti

Tutors:

Prof. Luca Dedè
Dr. Andrea Tonini

Abstract

In this report we apply Variational Autoencoders (VAE) in order to solve a Bayesian Inverse Problem. We consider a steady-state diffusion problem with a known force, and try to estimate the unknown diffusion coefficient varying in all the domain only by knowing the solution in some of the points. Our work reproduces some of the results obtained by Hwan Goh, Sheroze Sheriffdeen, Jonathan Wittmer and Tan Bui-Thanh in their article "Bayesian Neural Networks to solve inverse problems". We implement a framework for training neural networks capable of modelling the posterior distribution representing the unknown Parameter of Interest. This model is able to take advantage of all the information that are usually given in scientific Bayesian Inverse Problems.
Keywords: Machine Learning, Uncertainty Quantification, Bayesian Inverse Problems, Variational Autoencoders, Diffusion Problem, Finite Element Method.

Contents

1 Introduction

Inverse problems are a key issue in a wide range of scientific fields. Tipically this type of problems require to estimate a parameter starting from the knowledge of the observable data depending on the parameter. It is important to note that this task is computationally challenging, because it often includes ill-posed equations that do not have a unique solution [14]. Moreover, the solution can be very sensitive to small variations in the input data [2]. For this reason traditional numerical methods may not lead to satisfactory results, and different approaches have been tried in the past years, also involving neural networks.
In this report the following problem is addressed:

$$\begin{cases} -\nabla \cdot (q(\mathbf{x})\nabla y(\mathbf{x})) = f(\mathbf{x}) & \mathbf{x} \in \Omega := (-2,2) \times (-2,2) \\ y(\mathbf{x}) = 0 & \mathbf{x} \in \partial\Omega \end{cases} \tag{1}$$

where $q(\mathbf{x})$ is the diffusion coefficient, $y(\mathbf{x})$ is the observable solution of the problem and $f(\mathbf{x})$ is a known function. These equations represent a non homogeneous steady-state diffusion problem where the coefficient is not constant over the domain. It finds application in different situations, such as modelling a substance spreading in a liquid or heat diffusion in a closed room.
We approach the problem using variational autoencoders. This type of neural networks has been introduced by Kingma and Welling in 2013 [10] and has been widely used in the last years in different fields: for example they are an important tool in noise reduction and in image classification. The use of this method in solving inverse problems has been proposed in 2021 by [8] and it seems to be a promising technique that may be further exploited.
Adapting the model of [8] to our problem, we obtain a good reconstruction of the parameter of interest, as shown in section 4.

2 Methods

2.1 Variational Autoencoders

Variational autoencoders (VAE) are a family of deep generative models with use cases that span many applications, from image processing to bioinformatics. As their name suggests, VAE are a type of autoencoder, which is a model that takes a vector, $\mathbf{x} \in R^D$, compress it into a lower-dimensional vector, $\mathbf{z} \in \mathbb{R}^J$, where $J < D$, which is called **latent state**, and then decompress it back into **x'** (fig. 1). In order to obtain the process, they use two functions, $\mathbf{h}_\phi(\mathbf{x})$ and $\mathbf{f}_\theta(\mathbf{z})$, usually two neural networks, with parameters respectively ϕ and θ.

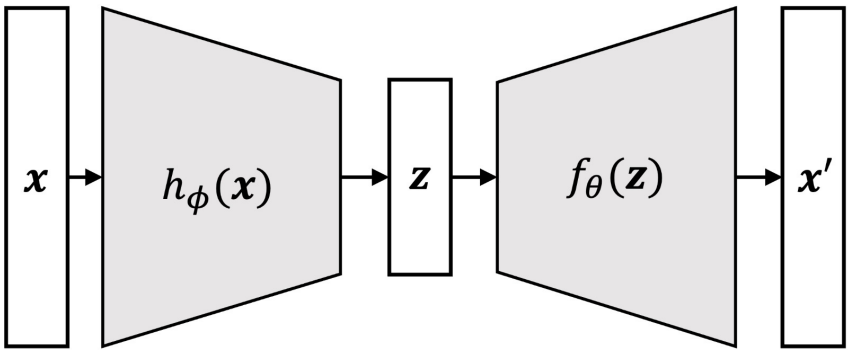

Fig. 1. Autoencoder

VAE introduce probabilistic models in this process. **Latent space** is represented by a parameterized family of probability distribution $p(\mathbf{z})$. Sample $\mathbf{x}$ are mapped through the encoder network

$\mathbf{h}_\phi(\mathbf{x})$ into the parameters of such distribution. The generative process goes as follow: first sample a latent variable $\mathbf{z}$ from $p(\mathbf{z})$, then use the decoder network f_θ to map $\mathbf{z}$ to parameters of another distribution used to sample $\mathbf{x}$ (fig. 2).

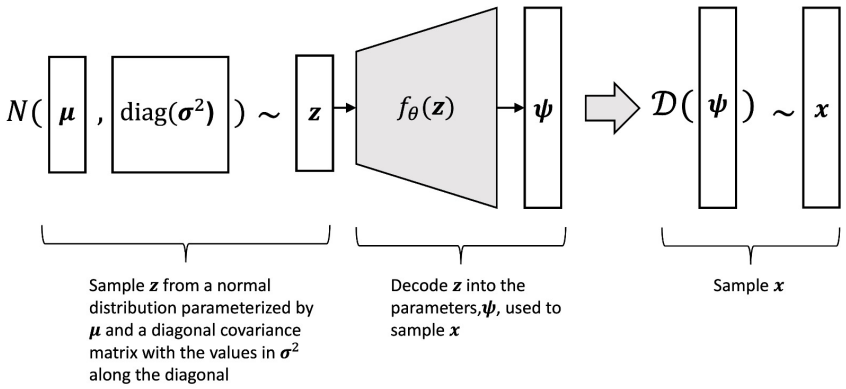

Fig. 2. VAE decoder

The goal of the training proces is to find the posterior distribution $p(\mathbf{z}|\mathbf{x})$, i.e reconstruct the latent space from data.
Given a dataset consisting of data points $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathbf{R}^D$, recalling the Bayes Theorem, we get:

$$p(\mathbf{z}_i|\mathbf{x}_i) = \frac{p(\mathbf{x}_i|\mathbf{z}_i)p(\mathbf{z}_i)}{\int p(\mathbf{x}_i|\mathbf{z}_i)p(\mathbf{z}_i)d\mathbf{z}_i}$$

The problem is that computing the denominator requires solving an integral over all the dimension of the latent space.
VAE find approximating solution to this intractable inference problem using **variational inference**, a method for performing such approximation by first choosing a set of probability distribution Π called the **variational family** and then finding the distributions $\pi_\phi(\mathbf{z}_i|\mathbf{x}_i) \in \Pi$ "closest" to $p(\mathbf{z}_i|\mathbf{x}_i)$. KL-Divergence between $\pi_\phi(\mathbf{z}_i|\mathbf{x}_i)$ and $p(\mathbf{z}_i|\mathbf{x}_i)$ is usually used as measure of "closeness". Thus the goal of variational inference is to minimize the KL divergence, which is equivalent to maximizing the evidence lower bound (ELBO) [4], defined as

$$\begin{aligned}\text{ELBO}(\pi) &:= \mathbb{E}_{\mathbf{z}_1,...,\mathbf{z}_n\sim\pi}\left[\sum_{i=1}^{n}\log\, p(\mathbf{x}_i,\mathbf{z}_i) - \sum_{i=1}^{n}\log\,\pi_\phi(\mathbf{z}_i|\mathbf{x}_i)\right] \\ &= \sum_{i=1}^{n}\mathbb{E}_{\mathbf{z}_i\sim q}\left[\log\, p(\mathbf{x}_i,\mathbf{z}_i) - \log\,\pi_\phi(\mathbf{z}_i|\mathbf{x}_i)\right]\end{aligned}$$

VAE use a variational family of the form [10]

$$\Pi := \left\{N(\mu_\phi, \sigma^2_\phi)|\phi \in \mathbb{R}^R\right\}$$

where

$$\begin{aligned}\boldsymbol{\mu}_\phi &= h^{(1)}_\phi(\mathbf{x}) \\ \boldsymbol{\sigma}^2_\phi &= \text{diag}(\exp(h^{(2)}_\phi(\mathbf{x})))\end{aligned}$$

and R is the number of parameters of this neural network (fig. 3).
That is, we use a single set of neural network parameters ϕ to encode the posterior distribution $\pi_\phi(\mathbf{z}|\mathbf{x}_i)$.

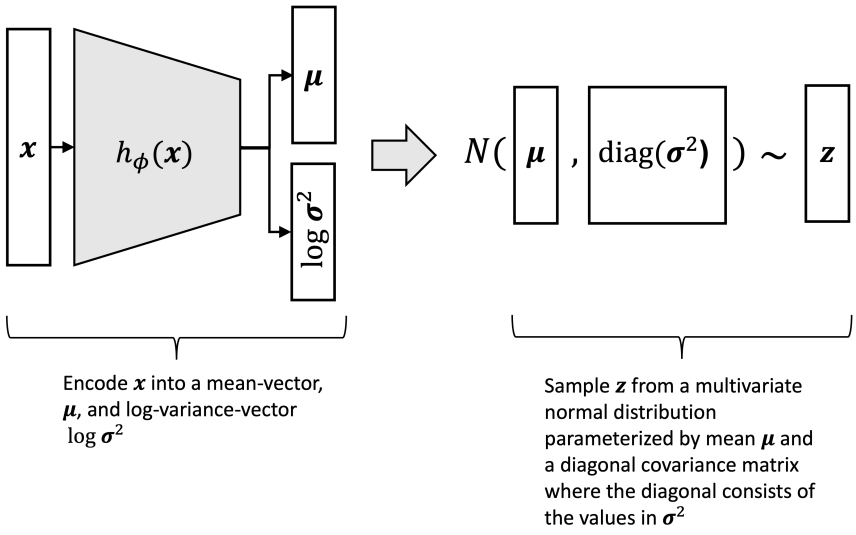

Fig. 3. VAE encoder

In other words we define $\pi_\phi(\mathbf{z}|\mathbf{x})$ as

$$\pi_\phi(\mathbf{z}|\mathbf{x}) := N(h_\phi^{(1)}(\mathbf{x}), \text{diag}(\exp(h_\phi^{(2)}(\mathbf{x}))))$$

Thus, maximizing the ELBO over Π reduces to maximizing the ELBO over the neural network parameters ϕ:

$$\hat{\phi} = \arg\max_\phi \text{ELBO}(\phi)$$

i.e., using the definition of ELBO:

$$\arg\max_\phi \sum_{i=1}^{n} E_{\mathbf{z}_i \sim \pi_\phi(\mathbf{z}_i|\mathbf{x}_i)} [\log p(\mathbf{x}_i, \mathbf{z}_i) - \log \pi_\phi(\mathbf{z}_i \mid \mathbf{x}_i)]$$

The expectation in the ELBO formulation makes it difficult to maximize as it requires integrating over all possible values for $\mathbf{z}$. We address this by introducing the **reparameterization gradient method**. It maximizes the ELBO via stochastic gradient ascent in which stochastic gradients are formulated by first performing the **reparameterization trick** followed by Monte Carlo sampling. It works as follows: we "reparameterize" the distribution $\pi_\phi(\mathbf{z}_i|\mathbf{x}_i)$ in terms of a surrogate random variable $\boldsymbol{\epsilon}_i \sim D$ and a deterministic function g_ϕ in such a way that sampling $\mathbf{z}$ from $\pi_\phi(\mathbf{z}_i|\mathbf{x}_i)$ is performed as follows:

$$\boldsymbol{\epsilon}_i \sim D$$
$$\mathbf{z}_i := g_\phi(\epsilon_i, \mathbf{x}_i)$$

One way to think about this is that instead of sampling $\mathbf{z}$ directly from our variational posterior $\pi_\phi(\mathbf{z}_i|\mathbf{x}_i)$, we "re-design" the generative process of $\mathbf{z}_i$ such that we first sample a surrogate random variable ϵ_i and then transform ϵ_i into $\mathbf{z}_i$, all while ensuring that, in the end, the distribution of $\mathbf{z}_i$ still follows $\pi_\phi(\mathbf{z}_i|\mathbf{x}_i)$. In this way we can rewrite the ELBO as follow:

$$\text{ELBO}(\phi) := \sum_{i=1}^{n} E_{\epsilon_i \sim \mathscr{D}} [\log p(\mathbf{x}_i, g_\phi(\boldsymbol{\epsilon}_i, \mathbf{x}_i)) - \log \pi_\phi(g_\phi(\boldsymbol{\epsilon}_i, \mathbf{x}_i) \mid \mathbf{x}_i)]$$

and then approximate it via Monte Carlo sampling, i.e. for each sample, we first sample random variables from our surrogate distribution,

$$\boldsymbol{\epsilon}'_{i,1}, \ldots, \boldsymbol{\epsilon}'_{i,L} \sim \mathscr{D}$$

and then compute a Monte Carlo approximation of the ELBO as follow

$$\tilde{\text{ELBO}}(\phi) := \frac{1}{n}\sum_{i=1}^{n}\frac{1}{L}\sum_{l=1}^{L}\left[\log p(\mathbf{x}_i, g_\phi(\boldsymbol{\epsilon}'_{i,l}, \mathbf{x}_i)) - \log \pi_\phi(g_\phi(\boldsymbol{\epsilon}'_{i,l}, \mathbf{x}_i) \mid \mathbf{x}_i)\right]$$

Since $\pi_\phi(\mathbf{z}|\mathbf{x}) := N(h_\phi^{(1)}(\mathbf{x}), \text{diag}(\exp(h_\phi^{(2)}(\mathbf{x})))$, we can reparameterize as

$$\begin{aligned}\boldsymbol{\epsilon}_i &\sim N(\mathbf{0}, \boldsymbol{I})\\ \mathbf{z}_i &:= h_\phi^{(1)}(\mathbf{x})_i + \boldsymbol{\epsilon}_i\sqrt{\exp(h_\phi^{(2)}(\mathbf{x}))_i}\end{aligned}$$

and because this ELBO formulation is differentiable w.r.t ϕ and θ, we can form the gradient

$$\nabla_\phi\tilde{\text{ELBO}}(\phi) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{L}\sum_{l=1}^{L}\nabla_\phi\left[\log p(\mathbf{x}_i, g_\phi(\epsilon'_{i,l}, \mathbf{x}_i)) - \log \pi_\phi(g_\phi(\epsilon'_{i,l}, \mathbf{x}_i) \mid \mathbf{x}_i)\right]$$

that can be used to train our deep learning model.

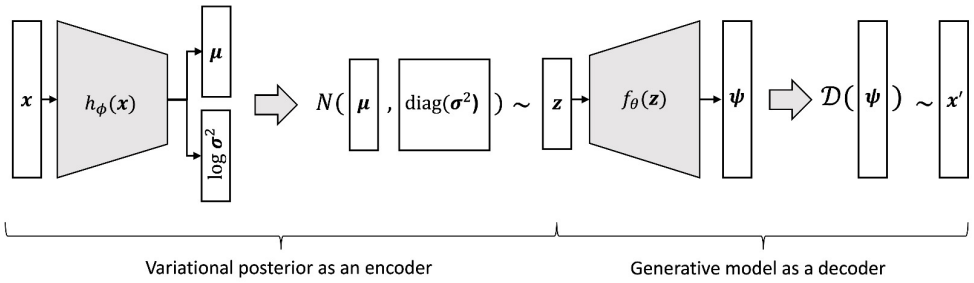

Fig. 4. VAE as autoencoder

2.2 Inverse Problems

In the setting of physical systems, scientific Inverse Problems task us with determining some Parameters-of-Interest (PoI) given observations of a state variable. Consider the following additive noise model

$$\mathbf{y} = F(\mathbf{u}) + \mathbf{e}$$

where $\mathbf{u}$ are the PoI, F is the Parameter-to-Observable (PtO) map and $\mathbf{y}$ represent observational data.
The goal here is to determine the parameter of interest $\mathbf{u}$ given the observational data $\mathbf{y}$ [9].
Solving the Inverse Problem usually involves solving an optimization problem of the form

$$\min_{\mathbf{u}}\|\mathbf{y} - F(\mathbf{u})\|_2^2 + R(\mathbf{u})$$

where $R(\mathbf{u})$ is a regularization term to reduce the size of the solution space, since the problem is usually ill-posed (many possible solution exist that are coherent with our data).
Usually the optimization of such functional is computationally expansive.
A possible solution is learning a data-driven solver that after an offline training stage (expansive but done only one time), output estimate of our PoI.
Given as training set $\{(\mathbf{u}^{(m)}, \mathbf{y}^{(m)})\}_{m=1}^M$, if we use a neural network Ψ then our solver is parametrized by the weight of the network $\mathbf{W}$. This solver require the optimization of the following functional:

$$\min_{\mathbf{W}}\frac{1}{M}\sum_{m=1}^{M}\|\mathbf{u}^{(m)} - \Psi(\mathbf{y}^{(m)}, \mathbf{W})\|_2^2 + R(\mathbf{W})$$

Instead of regularizing the weights of the network directly, we can regularize the output of the network, informing the optimization procedure of the inversion task, of the properties of the noise afflicting our observational data and the knowledge about some physical properties of the PoI we posses. This lead to the following optimization objective

$$\min_{\mathbf{W}} \frac{1}{M} \sum_{m=1}^{M} \|\mathbf{u}^{(m)} - \Psi(\mathbf{y}^{(m)}, \mathbf{W})\|_2^2 + \|\mathscr{M}(\mathbf{y} - F(\Psi(\mathbf{y}^{(m)}, \mathbf{W})))\|_2^2 + \|\mathscr{P}(\Psi(\mathbf{y}^{(m)}, \mathbf{W}))\|_2^2$$

where M, P are some mapping representing regularization about noise and PoI respectively.

2.3 Bayesian Inverse Problems

By adopting a probabilistc framework instead of deterministic, the question asked by the Inverse Problem essentially changes from "what is the value of our parameter?" to "how accurate is the estimate of our parameter?". This is normally referred in the literature as **uncertainty quantification** [9] [6]. In this setting, Inverse Problems deal with the following observational model

$$Y = F(U) + E$$

where F is **Parameter-to-Observable(PtO)** map and Y, U, E are random variables representing respectively the observational data, the **Parameter-to-Observable(PtO)** and the noise model.
In this statistical framework the goal is to model the posterior distribution $P(\mathbf{u}|\mathbf{y})$, i.e "given the observational data, what is the distribution of the Parameters of Interest?"
Using Bayes' Theorem we obtain

$$P(\mathbf{u}|\mathbf{y}) \propto P(\mathbf{y}|\mathbf{u})P(\mathbf{u})$$

The assumptions usually made are:

- $E \sim N(\mu_E, \Gamma_E)$
- $U \sim N(\mu_{pr}, \Gamma_{pr})$,
- $E \perp U$

leading to the following likelihood model:

$$p(\mathbf{y}|\mathbf{u}) = p_E(\mathbf{y} - F(\mathbf{u})).$$

Methods to compute such posterior probability often require a potentially expensive derivative-based iterative optimization procedure [9]. Markov Chain Monte Carlo (MCMC) methods can also be employed but such methods notably suffer the curse of dimensionality [6].
This desire for efficient uncertainty quantification also motivates the work of [8]. Although the training procedure of a Bayesian Inverse Problem solver may be computationally demanding, it represents an offline stage. Once trained, the online stage of uncertainty quantification has a computational cost governed only by the forward propagation of the observational data through the trained neural network; which is often a very computationally efficient procedure. Additionally, the PoI data $\mathbf{u}^{(m)}$ in the training dataset $\left\{\left(\mathbf{u}^{(m)}, \mathbf{y}_{\text{obs}}^{(m)}\right)\right\}_{m=1}^{M}$ usually plays no role in traditional methods. Information about the PoI is often only encoded in the prior model.
This motivates the data-driven aspect of this approach in that we not only use prior information but we also take advantage of PoI data in the cases where paired PoI-observation datasets are readily obtainable.

2.4 Solving Bayesian Inverse Problems via Variational Autoencoders

Computing the above posterior probability is often intractable. A possible solution is using **variational inference** to approximate it.
Denoting by $P(\mathbf{u}|\mathbf{y})$ the target posterior density we want to approximate, variational inference perform such approximation choosing a set of probability distribution $Q_\phi(\mathbf{u}|\mathbf{y})$ parameterized by a parameter ϕ and then finding the distribution in such family "closest" to the target one [7].
To do that some notion of distance between $Q_\phi(\mathbf{u}|\mathbf{y})$ and $P(\mathbf{u}|\mathbf{y})$ must be introduced. Normally the Kullback-Leibler divergence (KLD) is used to obtain the evidence lower bound (ELBO), an useful lower bound on the log-likelihood of some observed data, and the goal of variational inference is to minimize the KL divergence, or, equivalently, maximizing the evidence lower bound (ELBO) [7].
Following the work in the work of Goh et al. [8], instead of KLD, the following family of Jensen-Shannon Divergence (JSD) [12] is used,

$$JS_\alpha(q\|p) = \alpha KL(q\|(1-\alpha)q + \alpha p) + (1-\alpha)KL(p\|(1-\alpha)q + \alpha p)$$

which leads, to perform the task of variational inference, to the minimization with respect to ϕ of the following quantity

$$\frac{1-\alpha}{\alpha} KL(P(\mathbf{u}|\mathbf{y})|Q_\phi\mathbf{u}|\mathbf{y})) - \mathbb{E}_{\mathbf{u}\sim Q_\phi}\left[\log(P(\mathbf{y}|\mathbf{u}))\right] + KL(Q_\phi(\mathbf{u}|\mathbf{y})|P(\mathbf{u}))$$

where $KL(\cdot,\cdot)$ denote the Kullback-Leibler divergence.
From the perspective of standard VAEs, notice that the second and third terms together form the negative of the ELBO. From the perspective of Bayesian Inverse Problems, the second term in is the likelihood model containing the PtO map and the third term contains the prior model p($\mathbf{u}$) representing information about the PoI. The first term represents information regarding the target posterior. This is the key achievement of utilizing the JSD as the notion of distance between the model posterior and target posterior as it is this term that allows for the inclusion of paired PoI-observation datasets.
The objective is now to derive from the above optimization problem a differentiable regularized loss function suitable for the training of a neural network, of the form described in chapter 2.2. Starting from the first term in the above expression, we note that

$$\min_\phi KL(P(\mathbf{u}|\mathbf{y})|Q_\phi\mathbf{u}|\mathbf{y})) = \mathbb{E}_{\mathbf{u}\sim p(\mathbf{u}|\mathbf{y}^{(m)}_{\text{obs}})}\left[-\log\left(q_\phi(\mathbf{u}|\mathbf{y}^{(m)}_{\text{obs}})\right)\right]$$

since P is constant w.r.t. to ϕ. Than the minimization of the KLD between the empirical and model distributions is equivalent to the minimization of the negative of the likelihood function with respect to ϕ [4]. Next we see that, using Bayes theorem

$$\begin{aligned}\mathbb{E}_{\mathbf{u}\sim p(\mathbf{u}|\mathbf{y}^{(m)}_{\text{obs}})}\left[-\log\left(q_\phi(\mathbf{u}|\mathbf{y}^{(m)}_{\text{obs}})\right)\right] &= -\int \ln q_\phi\left[p(\mathbf{y}^{(m)}_{\text{obs}}|\mathbf{u})p(\mathbf{u})\right] \\ &= -\int \ln q_\phi\left[p_E(\mathbf{y}^{(m)}_{\text{obs}} - F(\mathbf{u}))p(\mathbf{u})\right] \\ &\leq -\int \ln q_\phi p(\mathbf{u}) = \mathbb{E}_{\mathbf{u}\sim p(\mathbf{u})}\left[-\log\left(q_\phi(\mathbf{u}|\mathbf{y}^{(m)}_{\text{obs}})\right)\right]\end{aligned}$$

since, as stated in the previous chapter, $p(\mathbf{y}|\mathbf{u}) = P_E(\mathbf{y} - F(\mathbf{u}))$ and $E \sim \mathcal{N}(\mathbf{u}_E, \mathbf{\Gamma}_E)$.
Finally, we form a Monte-Carlo estimation using the PoI data and adopt a Gaussian distribution for the model posterior: $q_\phi\left(\mathbf{u}|\mathbf{y}^{(m)}_{\text{obs}}\right) = \mathcal{N}\left(\mathbf{u} \mid \mu^{(m)}_{\text{post}}, \mathbf{\Gamma}^{(m)}_{\text{post}}\right)$. These assumptions can be summarized by the following chain of equations:

$$\mathbb{E}_{\mathbf{u}\sim p(\mathbf{u}|\mathbf{y}_{\text{obs}}^{(m)})}\left[-\log\left(q_\phi(\mathbf{u}|\mathbf{y}_{\text{obs}}^{(m)})\right)\right] \lesssim \mathbb{E}_{\mathbf{u}\sim p(\mathbf{u})}\left[-\log\left(q_\phi(\mathbf{u}|\mathbf{y}_{\text{obs}}^{(m)})\right)\right] \approx -\log\left(q_\phi(\mathbf{u}|\mathbf{y}_{\text{obs}}^{(m)})\right)$$

$$= \frac{D}{2}\log(2\pi) + \frac{1}{2}\log\left|\mathbf{\Gamma}_{\text{post}}^{(m)}\right| + \frac{1}{2}\left\|\mu_{\text{post}}^{(m)} - \mathbf{u}^{(m)}\right\|^2_{\mathbf{\Gamma}_{\text{post}}^{(m)-1}}.$$

Similar steps can be carried out also on the second a third term.
Next, the work in [8] incorporate deep learning into the framework. Consider a neural network that takes in the observation data $\mathbf{y}_{\text{obs}}^{(m)}$ as an input to output the statistics $\left(\mu_{\text{post}}^{(m)}, \mathbf{\Gamma}_{\text{post}}^{(m)}\right)$ of our posterior model.
In doing so, we are reparametrizing the statistics of our Gaussian posterior model represented by ϕ with the weights $\mathbf{W}$ of the neural network Ψ; thereby increasing the learning capacity of our model.
As mentioned in Section 2.2, it is common to adopt Gaussian noise and prior models $\mathscr{N}(\mu_E, \mathbf{\Gamma}_E)$ and $\mathscr{N}(\mu_{\text{pr}}, \mathbf{\Gamma}_{\text{pr}})$. With this and our approximation above, we obtain the following optimization problem:

$$\begin{aligned}\min_{\mathbf{W}} \frac{1}{M}\sum_{m=1}^{M}\frac{1-\alpha}{\alpha}\left(\log\left|\mathbf{\Gamma}_{\text{post}}^{(m)}\right| + \left\|\mu_{\text{post}}^{(m)} - \mathbf{u}^{(m)}\right\|^2_{\mathbf{\Gamma}_{\text{post}}^{(m)-1}}\right)\\ + \left\|\mathbf{y}_{\text{obs}}^{(m)} - \mathscr{F}\left(\mathbf{u}_{\text{draw}}^{(m)}(\mathbf{W})\right) - \mu_E\right\|^2_{\mathbf{\Gamma}_E}\\ +\text{tr}\left(\mathbf{\Gamma}_{\text{pr}}^{-1}\mathbf{\Gamma}_{\text{post}}^{(m)}\right) + \left\|\mu_{\text{post}}^{(m)} - \mu_{\text{pr}}\right\|^2_{\mathbf{\Gamma}_{\text{pr}}^{-1}} + \log|\mathbf{\Gamma}_{\text{pr}}| - \log\left|\mathbf{\Gamma}_{\text{post}}^{(m)}\right|.\end{aligned}$$

where

$$\left(\mu_{\text{post}}^{(m)}, \mathbf{\Gamma}_{\text{post}}^{(m)}\right) = \Psi\left(\mathbf{y}_{\text{obs}}^{(m)}, \mathbf{W}\right),$$

$$\mathbf{u}_{\text{draw}}^{(m)}(\mathbf{W}) = \mu_{\text{post}}^{(m)} + \mathbf{\Gamma}_{\text{post}}^{(m)\frac{1}{2}}\epsilon^{(m)},$$

$$\epsilon^{(m)} \sim \mathscr{N}(0, \mathbf{I}_D).$$

During the training procedure, the repeated operation of the PtO map on our dataset $\{(\mathbf{u}^{(m)}, \mathbf{y}^{(m)})\}_{m=1}^{M}$ may incur a significant computational cost. To alleviate this, the PtO map is replaced with another neural network Ψ_d parameterized by weights $\mathbf{W}_d$. This is reflected in the following modification of the loss function

$$\|\mathbf{y}^{(m)} - F(\mathbf{u}_{draw}^{(m)}(\mathbf{W})) - \mu_E\|^2_{\Gamma_E^{-1}} \to \|\mathbf{y}^{(m)} - \Psi_d(\mathbf{u}_{draw}^{(m)}(\mathbf{W}), \mathbf{W}_d) - \mu_E\|^2_{\Gamma_E^{-1}}$$

The parameter α can be varied in order to balance the weights of the difference terms of the loss function ([8], [11], [3]). A choice of $\alpha \approx 0$ promotes the influence of the PoI data on the optimization problem over the reconstruction of the data. A choice of $\alpha \to 1$ causes the loss function to have some instability. To balance these two effects we take the value $\alpha = 0.5$. Notice that the resulting optimization problem resembles those typically used to train an autoencoders. Since the learned model Ψ_d of the PtO map F is a neural network, the evaluation of this map during the training phase is computationally inexpensive. However, this efficiency comes at the expense of utilizing knowledge of the governing physics of the inverse problems. Thus, the modified framework is almost purely data-driven and resembles more the original utility of VAEs for generative modelling. A schematic of this framework with learned PtO map is displayed in (fig. 5).

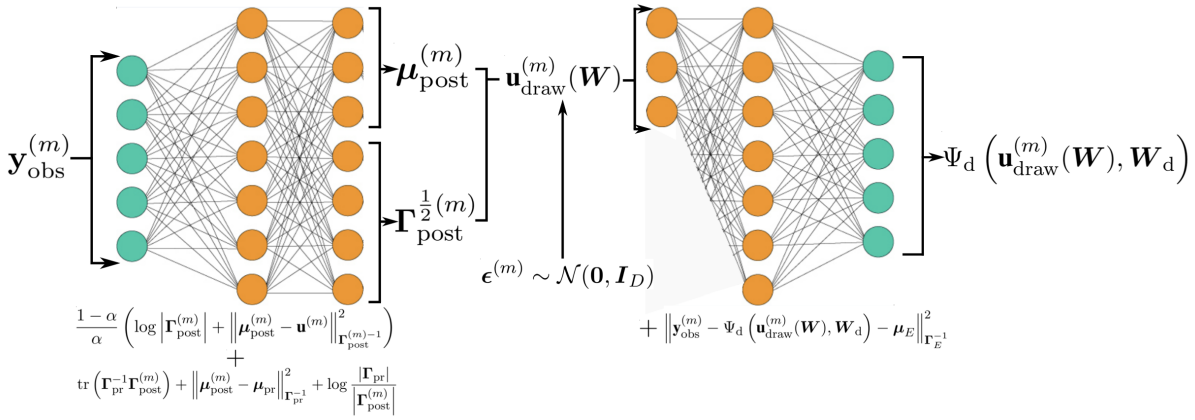

Fig. 5. Schematic Vae framework for the learned Pto map

2.5 FEM for Diffusion Problems

To generate couples coefficient-solution for our dataset we need to develop a numerical solver for Problem (1). We addressed the problem via **Finite Element Method** (FEM).
The first step is to write the **weak formulation** of the problem. We do it by multiplying the equation for a function $v \in H_0^1(\Omega)$ and integrating over the whole domain Ω. We get the following formulation:

$$\text{Find } y \in H_0^1(\Omega) \text{ such that: } \int_\Omega q\nabla y \cdot \nabla v \, d\Omega = \int_\Omega f v \, d\Omega \quad \forall v \in H_0^1(\Omega)$$

We can prove that, if $q \in L^\infty(\Omega)$, $q(\mathbf{x}) \geq q_0 > 0 \quad \forall \mathbf{x} \in \Omega$ and $f \in L^2(\Omega)$, then the formulation is well-posed, i.e. there exists a unique solution y to the problem. Moreover, the solution is stable:

$$\|y\|_{H^1(\Omega)} \leq \frac{\|f\|_{L^2(\Omega)}}{q_0}.$$

We get this result by Lax-Milgram lemma ([13], *section 2*), since:

- $V = H_0^1(\Omega)$ is a Hilbert space, with the norm $\|v\|_V = \|\nabla v\|_{L^2} \quad \forall v \in V$
- $a(y, v) = \int_\Omega q\nabla y \cdot \nabla v \, d\Omega$ is continuous: $|a(y, v)| \leq \|q\|_{L^\infty}\|y\|_V\|v\|_V \quad \forall y, v \in V$
- $a(y, v)$ is coercive: $a(v, v) \geq q_0\|v\|_V^2 \quad \forall v \in V$
- $F(v) = \int_\Omega f v \, d\Omega$ is continuous: $F(v) \leq \|f\|_{L^2}\|v\|_V \quad \forall v \in V$.

We can now apply the **Galerkin method** to the problem: given a sequence of finite dimensional subspaces $V_h \subseteq V$ we can obtain a sequence of finite dimensional problems whose solutions y_h converge to the exact solution y:

$$\text{Find } y_h \in V_h \text{ such that: } \int_\Omega q\nabla y_h \cdot \nabla v_h \, d\Omega = \int_\Omega f v_h \, d\Omega \quad \forall v_h \in V_h.$$

The result of convergence is guaranteed by Céa's lemma ([13], *section 4*):

$$\|y - y_h\|_V \leq \frac{||q||_{L^\infty(\Omega)}}{q_0} \inf_{v_h \in V_h} \|y - v_h\|_V.$$

We apply the Galerkin method with finite elements. We introduce a triangulation $\mathscr{T}_h = \bigcup K$, with $K_i \cap K_j = \emptyset \quad \forall i \neq j$ and the FE space $V_h^r := \{v_h \in V : v_h|_K \in \mathbb{P}^r(K) \quad \forall K \in \mathscr{T}_h\}$.

For our work we use a linear finite element method ($r = 1$).
In the space V_h we can take the Lagrange interpolant for the exact solution y. We can use the interpolation bound result ([13], *section 4*) to estimate the error:

$$\|v - \Pi_h^r v_h\|_V \lesssim h^r |v|_{H^{r+1}(\Omega)}$$
$$\|v - \Pi_h^r v_h\|_{L^2(\Omega)} \lesssim h^{r+1} |v|_{H^{r+1}(\Omega)}.$$

In the linear case, if $v \in H^2(\Omega)$, we get:

$$\|v - \Pi_h^r v_h\|_V \lesssim h |v|_{H^2(\Omega)}$$
$$\|v - \Pi_h^r v_h\|_{L^2(\Omega)} \lesssim h^{r+1} |v|_{H^2(\Omega)}.$$

Combining this result with the previous one, we get an error estimate for our problem:

$$\|y - y_h\|_V \lesssim h |y|_{H^2(\Omega)}$$
$$\|y - y_h\|_{L^2(\Omega)} \lesssim h^2 |y|_{H^2(\Omega)}.$$

3 Experimental Settings

3.1 Two Dimensional Steady State Heat Conduction Problem

We considered the heat equation with heat conductivity as the PoI and temperature as the state. The governing PDE and associated boundary conditions are displayed below

$$\begin{cases} -\nabla \cdot (q(\mathbf{x}) \nabla y(\mathbf{x})) = f(\mathbf{x}) & \mathbf{x} \in \Omega := (-2, 2) \times (-2, 2) \\ y(\mathbf{x}) = 0 & \mathbf{x} \in \partial\Omega \end{cases}$$

where $q(\mathbf{x})$ is the diffusion coefficient, $y(\mathbf{x})$ is the observable solution of the problem and $f(\mathbf{x})$ is a known function.
We consider a computational domain consisting of D degrees of freedom. The observation data corresponds to sensor measurements from O randomly selected locations and is afflicted with Gaussian distributed additive noise on all sensors.

3.2 Probabilistic Models

As prior model for the parameter of interest we set

$$P(\mathbf{u}) = N(\mu_{pr}, \Gamma_{pr})$$

with $\mu_{pr} = 2\mathbf{I}_D$ and Γ_{pr} s.t.

$$\Gamma_{i,j} = \sigma^2 \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|_2^2}{2\rho^2})$$

with σ^2, $\rho = 0.5$, as done in [5].
For the noise model, we set

$$P(E) = N(\mu_E, \Gamma_E)$$

with $\mu_E = 0\mathbf{I}_O$ and $\Gamma_E = \sigma^2 \mathbf{I}_O$, with $\sigma = \eta \max |\mathbf{y}|$.
Finally we assume a diagonal matrix for the posterior covariance Γ_{post}.

3.3 Dataset Generation

First, we draw the parameter of interest $\mathbf{u} := \mathbf{q(x)}$ from the Gaussian distribution with mean $\mu = 2$ and covariance $\Gamma_{i,j}$ defined in the previous section 3.2.
Notice that, to have stability, we need the bilinear form $a(h, v)$ to be coercive, which is guaranteed if each $\mathbf{q(x)}$ is greater than a certain $q_0 > 0$. So, when we generate the parameter of interest, we use an iterative loop and repeat the procedure till each $\mathbf{q(x)}$ is bigger than $q_0 = 0.05$.
Then we generate the corresponding observation $\mathbf{y}$ using the FEM solver previously developed.
Finally, we generate the observation points of $\mathbf{y}$ that we will use to train the VAE. We choose a number O of points sampling it from a random distribution in the domain (see section 4.2) and then we create in this way both the training set $\{(\mathbf{u}^{(m)}, \mathbf{y}^{(m)})\}_{m=1}^{M}$ and test set $\{(\mathbf{u}^{(l)}, \mathbf{y}^{(l)})\}_{l=1}^{L}$.

3.4 Neural Network Architecture

The architecture of our neural network Ψ consists of 5 hidden layers of 500 nodes with the ReLU activation function. No activation function was used at the output layer. The input layer has O nodes, in order to match the dimension of the observational data which consists of the same number O of measurement points. The output layer has $2D$ nodes, where D is the dimension of the PoI, to represent the estimated posterior mean μ_{psot} and the diagonal of the posterior covariance Γ_{post}.
When the parameter-to-observable map is learnt, the corresponding decoder network Ψ_d has 2 hidden layers, also with 500 nodes and the ReLU activation function. Again, no activation function was used at the output layer. The input layer has D number of nodes to represent a draw from the learned posterior and the output layer has O number of nodes to match the dimension of the observational data.

3.5 Training

For optimization, we use a batch size of 20. Therefore, the loss is averaged over the number of PoI and observation pairs and the gradient is calculated for each batch. Optimization is conducted using the Adam optimizer which performs mini-batch momentum-based stochastic gradient descent.
This training procedure was repeated for 100 epochs.
The metric used to measure the accuracy on the test set is the Mean Relative Error (MRE)

$$\frac{1}{L}\sum_{l=1}^{L}\frac{\|\mathbf{y}^{(l)} - \Psi_d(\boldsymbol{\mu}_{\text{post}}(\mathbf{y}^{(l)}))\|_2^2}{\|\mathbf{y}^{(l)}\|_2^2}$$

where $\boldsymbol{\mu}_{\text{post}}(\mathbf{y}^{(l)})$ is the estimated posterior mean from the encoder network Ψ taking a datapoint $\mathbf{y}^{(l)}$ as input.

3.6 Software and Library

To develop the deep learning part of the project we use the Tensorflow framework from Google (https://www.tensorflow.org/?hl=it), while to implement the finite element solver we adapt the FEM solver used in The Numerical Analysis for Partial Differential Equation course offered at Politecnico di Milano [1]. The result of our work is visible at https://github.com/Riccardo-Rota/Napde-VAE.

4 Results

4.1 Convergence of the numerical method

We perform a test to see the convergence rate of the numerical FEM solver. In order to compare the numerical result with the exact solution, we take a problem whose solution can be computed analytically:

$$\begin{cases} -\nabla \cdot (q(\mathbf{x})\nabla y(\mathbf{x})) = f(\mathbf{x}) & \mathbf{x} \in \Omega := (0,1) \times (0,1) \\ y(\mathbf{x}) = 0 & \mathbf{x} \in \partial\Omega \end{cases} \tag{2}$$

with:

$$q(x_1, x_2) = \exp(x_1 + x_2)$$
$$f(x_1, x_2) = \exp(x_1 + x_2)[x_2(1 - x_2)(1 + 2x_1) + x_1(1 - x_1)(1 + 2x_2)].$$

The exact solution is given by $y(x_1, x_2) = x_1 x_2 (1 - x_1)(1 - x_2)$. We compute the numerical solution with 2, 3, 4, 5 refinements, so that the side of the square is divided respectively into 4, 8, 16, 32 segments. The error is given by the difference between the real solution and the computed solution. Taking the H^1 and L^2 norms of the error vector we obtain the results reported in (table 1). As expected from the analysis in section 2.5, the H^1 error scales with the same rate as h, while the L^2 error scales with the same rate as h^2, thus their convergence orders are respectively 1 and 2. A visual representation of this result is given by (fig. 6).

h	**H^1 Error**	**L^2 Error**
1.767×10^{-1}	5.928×10^{-2}	5.838×10^{-3}
8.839×10^{-2}	3.022×10^{-2}	1.542×10^{-3}
4.419×10^{-2}	1.519×10^{-2}	3.910×10^{-4}
2.210×10^{-2}	7.604×10^{-3}	1.026×10^{-4}

Table 1. Errors in H^1 and L^2 norm with 2,3,4,5 refinements.

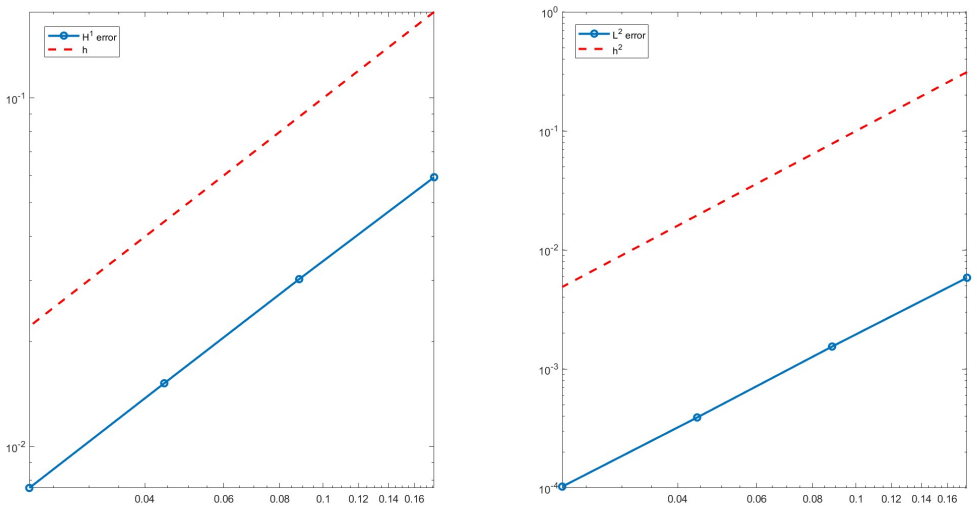

Fig. 6. Plot of errors in H^1 and L^2 norm compared with h and h^2

4.2 Reconstruction of Parameters of Interest and solutions

We run different tests to see how the autoencoder is able to estimate the true coefficient q and reconstruct the true solution y starting from the observation that it takes as input. For our tests we apply the model to the problem:

$$\begin{cases} -\nabla \cdot (q(\mathbf{x})\nabla y(\mathbf{x})) = f(\mathbf{x}) & \mathbf{x} \in \Omega := (-2,2) \times (-2,2) \\ y(\mathbf{x}) = 0 & \mathbf{x} \in \partial\Omega \end{cases}$$

with:

$$f(x_1, x_2) = \exp\left(\frac{x_1+2}{4} + \frac{x_2+2}{4}\right)\left(16 + 2x_1 + 2x_2 - \frac{1}{2}x_1x_2^2 - \frac{1}{2}x_1^2x_2 - 2x_1^2 - 2x_2^2\right)$$

- In tests 1-4 we vary the number of refinements (hence the number D of points in the grid), and the number O of points in which the solution is observed
- In tests 5-6 we apply our model to a diffusion problem with a different force $f(x_1, x_2)$.

For each test plot the real solution y, and the O sampled observation points (fig. 7). Then we see one example of the reconstruction of the coefficient q from the test set, plotting the real coefficient q and the mean and the variance of the reconstructed one, with a cross-sectional view along an horizontal line (fig. 8).
Variance and Confidence Interval are computed repeatedly sampling from the posterior. To construct the 95% credibility interval, we take the 2.5th and 97.5th percentiles of the sampled values. The lower bound is set at the 2.5th percentile, and the upper bound at the 97.5th percentile. This interval represents the range within which 95% of the posterior draws fall.
We also test the ability of the Variational Autoencoder to reconstruct the solution, plotting the real and the reconstructed solution in the observation points. (fig. 9).
For tests 1-4, the observation points are sampled from a gaussian distribution centered in the central point of the domain, with a suitable covariance. This is to better capture the shape of the solutions, which have peaks in the center of the domain.
For tests 5-6, the observation points are sampled from a uniform distribution. This is because of the particular shape of the solution due to the different forcing term used.
In table 2 we report the Mean Relative Error of the reconstruction of the solution y through the different tests. We observe that increasing the number O of observation points leads to a better reconstruction, while increasing the number of refinements does not.

Test	**O**	$\mathbf{n}_{\text{ref}}$	**MRE**
Test1	10	3	0.02249
Test2	20	3	0.01272
Test3	20	4	0.01725
Test4	75	4	0.00446
Test5	10	3	0.01409
Test6	20	3	0.01126

Table 2. Mean Relative Error measured on solution reconstruction

4.2.1 Test 1

The first test is conducted with the following experimental setup:

- $n_{\text{ref}} = 3$ (grid with $D = 81$ points)
- $O = 10$
- $f(x_1, x_2) = \exp\left(\frac{x_1+2}{4} + \frac{x_2+2}{4}\right)\left(16 + 2x_1 + 2x_2 - \frac{1}{2}x_1x_2^2 - \frac{1}{2}x_1^2x_2 - 2x_1^2 - 2x_2^2\right)$

We note from fig. 9 that taking only 10 observation points leads to a poor reconstruction of the y's, as such a small set of observed data is not enough to highlight the shape of the solution. However, the quality of the reconstruction of the latent variable q is not significantly negative affected by the small number of observed samples. (fig. 8).

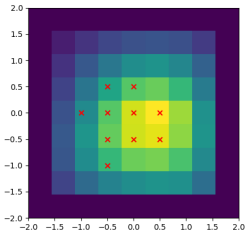

Fig. 7. Test 1: solution observation points

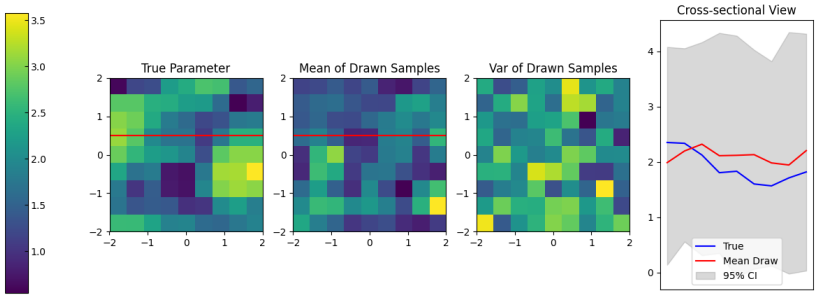

Fig. 8. Test 1: reconstruction of the diffusion coefficient q

.

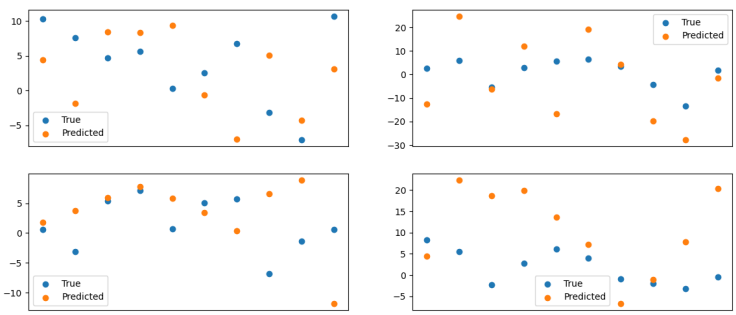

Fig. 9. Test 1: reconstruction of the solution y in the observation points

4.2.2 Test 2

The second test is conducted with the same setup as Test 1, but choosing 20 observation points:

- $n_{\text{ref}} = 3$ (grid with $D = 81$ points)
- $O = 20$
- $f(x_1, x_2) = \exp\left(\frac{x_1+2}{4} + \frac{x_2+2}{4}\right)\left(16 + 2x_1 + 2x_2 - \frac{1}{2}x_1x_2^2 - \frac{1}{2}x_1^2x_2 - 2x_1^2 - 2x_2^2\right)$

We note a good ability to reconstruct the coefficient q and also the solution y (fig. 11, fig. 12), improving the results of Test 1. We can observe that using 20 observation points is enough to recover the shape of the solution function.

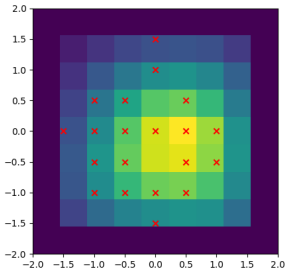

Fig. 10. Test 2: solution observation points

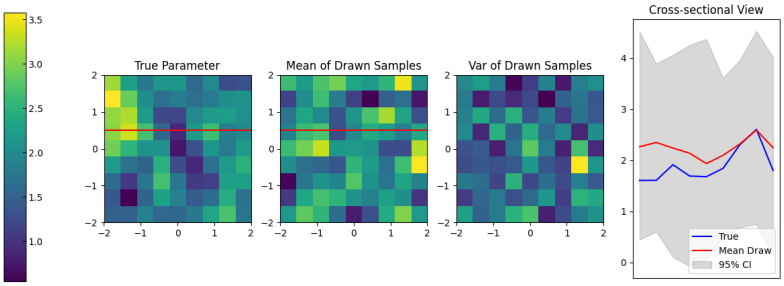

Fig. 11. Test 2: reconstruction of the diffusion coefficient q

.

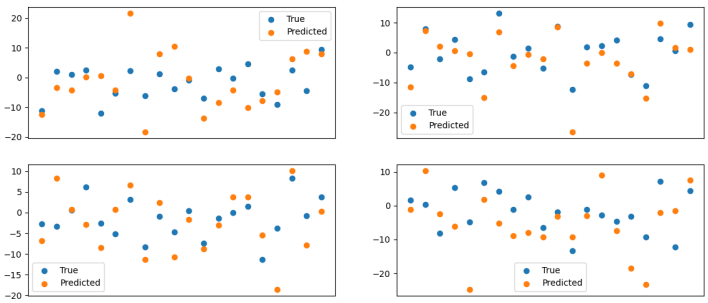

Fig. 12. Test 2: reconstruction of the solution y in the observation points

4.2.3 Test 3

The third test is conducted with the same setup as Test 2, but increasing the number of refinements to 4. The parameters of the test are:

- $n_{\text{ref}} = 4$ (grid with $D = 289$ points)
- $O = 20$
- $f(x_1, x_2) = \exp\left(\frac{x_1+2}{4} + \frac{x_2+2}{4}\right)\left(16 + 2x_1 + 2x_2 - \frac{1}{2}x_1x_2^2 - \frac{1}{2}x_1^2x_2 - 2x_1^2 - 2x_2^2\right)$

Increasing the number of refinement but maintaining constant the number of observation allows us to highlight the difference with Test 2. From the cross-sectional view in (fig. 14), is evident how increasing the number of refinement deteriorates the reconstruction of the parameter.

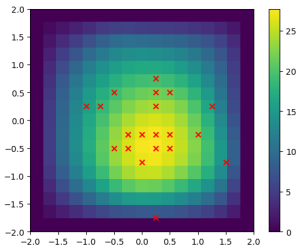

Fig. 13. Test 3: solution and sampled observation points

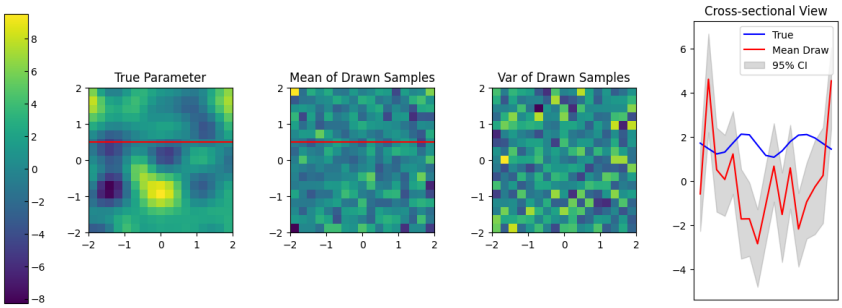

Fig. 14. Test 3: reconstruction of the diffusion coefficient q

.

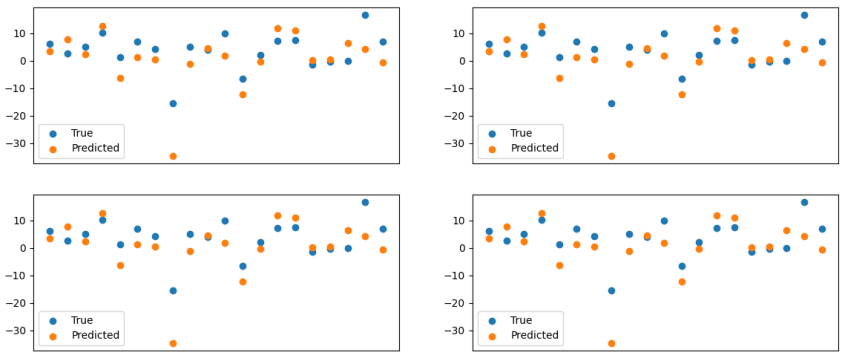

Fig. 15. Test 3: reconstruction of the solution y in the observation points

4.2.4 Test 4

The fourth test is conducted with the same setup as Test 3, but increasing the number of observed point to 75. The parameters of the test are:

- $n_{\text{ref}} = 4$ (grid with $D = 289$ points)
- $O = 75$
- $f(x_1, x_2) = \exp\left(\frac{x_1+2}{4} + \frac{x_2+2}{4}\right)\left(16 + 2x_1 + 2x_2 - \frac{1}{2}x_1x_2^2 - \frac{1}{2}x_1^2x_2 - 2x_1^2 - 2x_2^2\right)$

It is evident from (fig. 17) that increasing the number of observed data points does not improve the learning of the correct Parameters of Interests. This can be related to the diagonal covariance assumption done in [8], which does not allow learning of complex relationship in data.

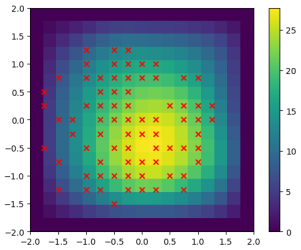

Fig. 16. Test 4: solution and sampled observation points

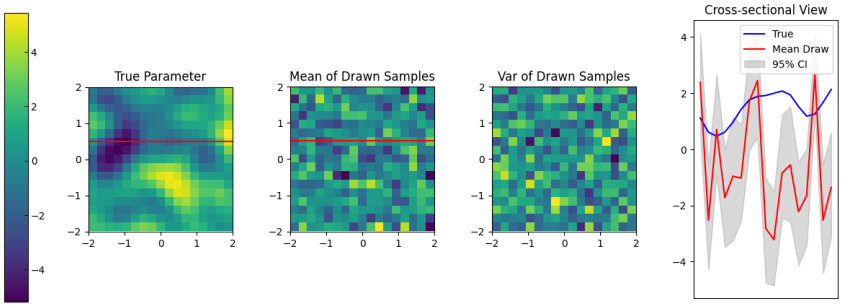

Fig. 17. Test 4: reconstruction of the diffusion coefficient q

.

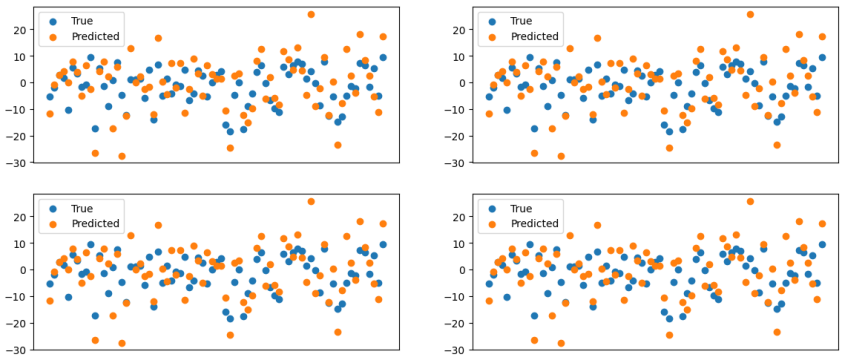

Fig. 18. Test 4: reconstruction of the solution y in the observation points

4.2.5 Test 5

Here we take the same setting as Test 1, but we vary the external force of the diffusion problem, taking a sinusoidal function. Hence, the parameters of the test are:

- $n_{\text{ref}} = 3$ (grid with $D = 81$ points)
- $O = 10$
- $f(x_1, x_2) = 10 \sin\left(\frac{\pi}{2} x_1\right) \sin\left(\frac{\pi}{2} x_2\right)$

As in test 1, the results in (fig. 20) show that the quality of the reconstruction of q does not decrease significantly. However, the reconstruction of the solution y using only 10 observation points gets slightly worse, as shown in (fig. 21).

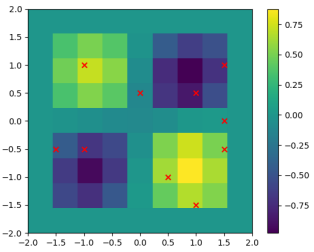

Fig. 19. Test 5: solution observation points

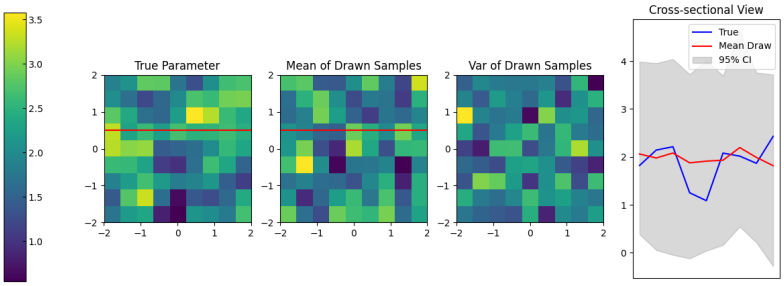

Fig. 20. Test 5: reconstruction of the diffusion coefficient q

.

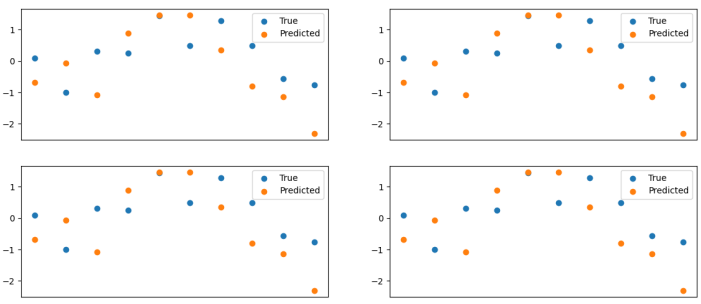

Fig. 21. Test 5: reconstruction of the solution y in the observation points

4.2.6 Test 6

We repeat the previous test increasing the number of observation points. The parameters of the test are:

- $n_{\text{ref}} = 3$ (grid with $D = 81$ points)
- $O = 20$
- $f(x_1, x_2) = 10 \sin\left(\frac{\pi}{2} x_1\right) \sin\left(\frac{\pi}{2} x_2\right)$

As in Test 2, the reconstruction of the coefficient q and the solution y is good. We can observe that using 20 uniform observation points is enough to recover the shape of the solution function.

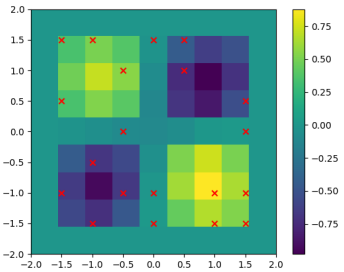

Fig. 22. Test 6: solution observation points

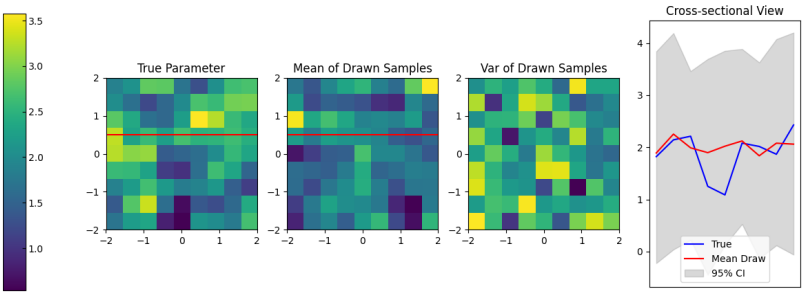

Fig. 23. Test 6: reconstruction of the diffusion coefficient q

.

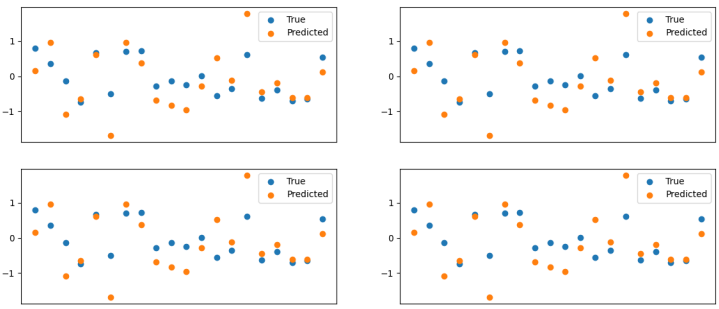

Fig. 24. Test 6: reconstruction of the solution y in the observation points

5 Conclusion

In this project we have tried to reproduce the work of [8], applying it to a steady state diffusion problem. We have used Variational Autoencoder to perform the task of solving the inverse problem.
The Variational Autoencoder is originally developed to reconstruct the input via a stochastic process, and in our work we have shown that it can reconstruct well the solution of the problem given as input.
Moreover, we have shown that it can also be used to reconstruct the diffusion coefficient, which in this context is the latent variable of the VAE. We have noticed that this second task presents some issues when the dimension gets bigger. This issues can be related to some assumptions that we have made on the covariance matrices. In particular, as the dimension grows, the eigenvalues of the prior covariance matrix get close to 0, leading to numerical problems. Furthermore, the assumption made by [8] of having a diagonal posterior covariance matrix (needed to keep low the computational cost) can lead to a poor representation of high-dimensional data.
Future studies could try to change the prior distribution or to assume a full covariance matrix, taking as output of the Variational Autoencoder its Cholesky factorization. This could be done with a higher computational power, as it would significantly increase the dimension of the output.

References

[1] P. F. Antonietti, C. B. Leimer Saglio, and F. Regazzoni. Numerical analysis for partial differential equations, course material, 2024.

[2] M. Bertero and P. Boccacci. *Introduction to Inverse Problems in Imaging.* Institute of Physics Publishing, 1998.

[3] C. M. Bishop. *Pattern recognition and machine learning.* Springer, 2006.

[4] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 2017.

[5] Y. Daon and G. Stadler. Mitigating the influence of the boundary on pde-based covariance operators. 2016.

[6] M. Dashti and A. Stuart. The bayesian approach to inverse problems. 2013.

[7] A. Ganguly and S. W. F. Earp. An introduction to variational inference. 2021.

[8] H. Goh, S. Sheriffdeen, J. Wittmer, and T. Bui-Thanh. Solving bayesian inverse problems via variational autoencoders. *Proceedings of Machine Learning Research*, 2021.

[9] J. Kaipio and E. Somersalo. *Statistical and computational inverse problems, volume 160.* Springer Science and Business Media, 2006.

[10] D. Kingma and M. Welling. Auto-encoding variational bayes. 2013.

[11] K. P. Murphy. *Machine learning: a probabilistic perspective.* MIT Press, 2012.

[12] F. Nielsen. A family of statistical symmetric divergences based on jensen's inequality. 2010.

[13] A. Quarteroni. *Numerical Models for Differential Problems.* Springer, 2017.

[14] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems.* Winston & Sons, Washington, DC, 1977.