Doubly Latent Neural Networks: Data-Driven Black-Box UQ for Boundary Value Problems

Grant Norman[1]

[1]Ann and H.J. Smead Aerospace Engineering Sciences, University of Colorado Boulder, CO 80303

9 May 2024

Abstract

In science and engineering, stochastic PDEs model PDE systems along with uncertainty. These stochastic PDEs often rely upon identifying uncertain inputs and their distributions. However, this task is often nontrivial, requiring calibration experiments and expensive estimation of parameter distributions. Further, this process relies upon extensive domain-specific expertise, to understand the governing PDE, identify uncertain parameters, and to find appropriate distributions for them. In cases where the underlying structure (such as the PDE) is partially or fully unknown, practitioners turn to general methods such as generative modeling through Variational Autoencoders (VAE) or invertible neural networks to generate more samples resembling their dataset. However, these methods often require a large dataset and provide very little interpretability. Motivated by these two extremes, we propose the Doubly Latent Neural Network, which empowers a VAE with a general PDE structure. This allows for training with less data, and more interpretability, without needing to identify uncertain parameters or even knowing the governing PDE. We present a numerical example of 1D steady-state heat conduction with an uncertain material. Our model learns an underlying stochastic PDE, allowing for new samples to be generated. Further, we demonstrate the robustness of our method to mesh refinement, by generating accurate samples on a mesh 10 times finer than the training mesh.

1 Introduction

Quantifying uncertainty in systems described by partial differential equations (PDEs) is a common and often costly problem. Countless fields within science and engineering rely upon partial differential equations to model phenomena. Further, there is almost always some degree of uncertainty, potentially regarding the initial conditions (ICs), boundary conditions (BCs), or the PDE itself. Characterizing the uncertainty in a PDE is itself often difficult. For instance, in computational mechanics, we can possibly attribute uncertainty to material parameters that our governing PDE relies upon, in either a homogeneous or spatially-dependent way. Even for a homogeneous material parameter, we need to have some estimate of the its uncertainty, which is not a trivial task. For instance, in Bayesian calibration, we begin with our prior distribution for the material parameter, perform experiments, and use the results of these experiments to calibrate the posterior distribution of the material parameter. Further, this entire effort was just to characterize the material, and we still need to propagate this posterior distribution through our forward model, to quantify the uncertainty for our problem of interest. Often, our PDE model of the system may be missing physics, or inaccurate. This poses a new challenge, where the uncertainty is not just about the parameters that the PDE includes, but also about the form of the PDE itself. A simple example would be describing a system with the advection-diffusion equation, but where the true system is actually better described by including a small reaction term. In other words, it is difficult to characterize the uncertainty arising from our model missing physics.

2 Motivation and Background

In the above, we alluded to the common uncertainty quantification workflow. Using domain specific knowledge, we first identify uncertain parameters. Then, often through some form of calibration, we quantify the uncertainty in each of these input parameters. For example, this could be through solving a Bayesian inverse problem, which uses data to update a prior distribution to a posterior distribution. Further, for expensive forward simulations, this problem often involves constructing a surrogate model, mapping from the parameters to some quantity of interest. After this inverse problem, the forward problem then aims to propagate the posterior distributions through the forward model. Again, this forward model is often expensive, requiring another surrogate model mapping from the parameters to a new quantity of interest. For instance, we can use a Polynomial Chaos Expansion (PCE) as either surrogate model. For both the inverse surrogate or the forward surrogate, we can determine the coefficients of the PCE either through compressed sampling approaches or through regression, depending on how many forward simulations we can run, how many uncertain parameters there are, and the degree of the PCE. Changing to a new forward simulation requires repeating the entire process to determine the PCE forward surrogate model. We must sample the posterior distributions, propagate these samples through an expensive forward model, and determine coefficients of the PCE for this new problem. Note, this standard workflow does not account for missing physics. In other words, all the uncertainty is characterized explicitly through the uncertain parameters we select.

We propose a new method of modeling the uncertainty in PDE systems, by letting whole form of the PDE characterize the uncertainty, and by using this PDE itself as part of our surrogate model. Rather than constructing multiple surrogate models for multiple tasks (inverse problem, forward problem, modification of the forward problem), we propose constructing a single surrogate model that can then be generalized to different problems, with minimal changes. Critically, this approach fundamentally aims to capture missing physics. This approach combines neural network PDEs with Variational Autoencoders (VAEs) [1, 2].

2.1 Neural Network PDEs

Consider some state $u(x, t; \mathbf{q})$, where x represents the spatial dimension, t corresponds to time, and $\mathbf{q} \in \mathbb{R}^{n_q}$ is random vector, corresponding to uncertainty in the state. If we assume that u is governed by a stochastic PDE, then it can be described by

$$u_t = \mathcal{N}(x, u, u_x, u_{xx}; \mathbf{q}), \tag{1}$$

with some further assumptions on the continuity of both u and relative to $\mathbf{q}$. Closely related, for some state $u(x)$, we can consider the Stochastic ODE or Boundary Value Problem (BVP), where

$$0 = \mathcal{N}(x, u, u_x, u_{xx}; \mathbf{q}). \tag{2}$$

We will focus primarily on the Stochastic ODE, although this can be easily generalized to the Stochastic PDE. Eq. 2 must be paired with sufficient boundary condition information to have a unique solution. For now, suppose we have Dirichlet BCs,

$$\text{BCs} = \begin{cases} u(0) = 0, \\ u(1) = 0. \end{cases} \tag{3}$$

Without further specifying the form of $\mathcal{N}$, we may or may not be able to determine u analytically for a given $\mathbf{q}$, and a solution may not even exist. Supposing that such a solution exists, we denote the solution operator as $\mathcal{S}$. In other words,

$$u(x, \mathbf{q}) = \mathcal{S}\left[\mathcal{N}, \text{BCs}\right]. \tag{4}$$

As an example, consider $\mathcal{N} = u_{xx} - q$. Then,

$$u(x, q) = \mathcal{S}\left[u_{xx} - q, \text{BCs}\right] = \frac{q}{2}(x-1)(x). \tag{5}$$

However, due to the unspecified nature of $\mathcal{N}$, such an operator is hard to further specify and even more difficult to know. As such, we will use a numerical method to approximate the solution of the problem, $\mathcal{S}^h$

$$u(x, \mathbf{q}) \approx \mathcal{S}^h\left[\mathcal{N}, \text{BCs}\right]. \tag{6}$$

We will present one such example of $\mathcal{S}^h$, but countless numerical methods can be used.

2.1.1 Finite Difference Solution Operator

Consider an equispaced mesh with $n_x + 1$ total points, given by $\{x_i\}_{i=0}^{n_x}$, and spacing Δx. We will represent the state of u at these points as v_i, or in other words, $u(x_i) = v_i$. From the zero Dirichlet boundary conditions, we have $v_0 = 0$ and $v_1 = 0$. We will approximate the derivatives using 3-point, 2nd-order, centered methods. Thus,

$$f(v_i) = \mathcal{N}\left(x_i, v_i, \frac{v_{i+1} - v_{i-1}}{2\Delta x}, \frac{v_{i+1} - 2v_i + v_{i-1}}{\Delta x^2}; \mathbf{q}\right) \approx 0, \quad i = 1, 2, \ldots, n_x - 1. \tag{7}$$

Or for the vector versions, with $\mathbf{v} = \left[v_1, \ldots v_{n_x-1}\right]^T$, and $\mathbf{f}(\mathbf{v}; \mathbf{q}) = \left[f(v_1; \mathbf{q}), \ldots, f(v_{n_x-1}; \mathbf{q})\right]^T$,

$$\mathbf{f}(\mathbf{v}; \mathbf{q}) \approx \mathbf{0}.$$

To solve $\mathbf{f}(\mathbf{v}) = \mathbf{0}$, we use Newton's method. With some initial guess for the states $\mathbf{v}_0$, the iterative solution is computed through

$$\mathbf{v}_{j+1} = \mathbf{v}_j - (\nabla_{\mathbf{v}} \mathbf{f}(\mathbf{v}_j; \mathbf{q}))^{-1} \mathbf{f}(\mathbf{v}_j; \mathbf{q}), \quad j = 0, 1, \ldots, n_{\text{iters}} - 1 \tag{8}$$

where $\nabla_{\mathbf{v}} \mathbf{f}(\mathbf{v}_j)$ is the Jacobian of $\mathbf{f}$ with respect to $\mathbf{v}$, evaluated at the current iterate, $\mathbf{v}_j$. We take the solution from the Newton iterations as $\mathbf{v}_{n_{\text{iters}}}$ (or use a separate stopping criteria than reaching the maximum iterations). Note that $\mathbf{v}_{n_{\text{iters}}} = \mathbf{v}_{n_{\text{iters}}}(\mathbf{q})$, through the dependence of $\mathbf{f}$ on $\mathbf{q}$ (which is then through the dependence on it by $\mathcal{N}$). Pairing this vector and the BCs with an interpolant, we can construct the approximate solution $u^h(x, \mathbf{q}) = \mathcal{S}^h[\mathcal{N}, \text{BCs}] \approx \mathcal{S}[\mathcal{N}, \text{BCs}]$. For a given mesh, we also refer to the state of $u^h(x, \mathbf{q})$ at the mesh points as $\mathbf{u}^h(\mathbf{q})$.

2.2 Variational Autoencoders

Variational Autoencoders are a method to approximate a distribution, using variational inference [2–4]. Our goal is to construct an approximation to the data-generating distribution $p(\mathbf{u})$, from which new samples of $\mathbf{u}$ can be drawn. This distribution is approximated through a parameterization $p^\theta(\mathbf{u})$, where θ are selected to maximize the likelihood of observed data. This objective is equivalent to maximizing $\log(p^\theta(\mathbf{u}))$, as log is a monotonic function (where $\mathbf{u}$ is the data, as opposed to a random variable). Next derive the Evidence Lower Bound (ELBO). We introduce an approximate posterior $q^\phi(\mathbf{z}|\mathbf{u})$, to avoid computing the denominator of Bayes' Rule necessary to get $p^\theta(\mathbf{z}|\mathbf{u})$.

$$\log(p^\theta(\mathbf{u})) \leq \underbrace{\mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(\frac{p^\theta(\mathbf{u}, \mathbf{z})}{q^\phi(\mathbf{z}|\mathbf{u})}\right)}_{\text{ELBO}} \tag{9}$$

A few steps in the derivation of this inequality are included in the appendix. The KL divergence is given as

$$\text{KL}\left(q^\phi(\mathbf{z}|\mathbf{u}) || p^\theta(\mathbf{z}|\mathbf{u})\right) = \mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(\frac{q^\phi(\mathbf{z}|\mathbf{u})}{p^\theta(\mathbf{z}|\mathbf{u})}\right).$$

Next, note that through $p^\theta(\mathbf{u}, \mathbf{z}) = p^\theta(\mathbf{u}|\mathbf{z}) \cdot p^\theta(\mathbf{z})$ and properties of log, the ELBO can be rewritten as two terms.

$$\begin{aligned} \text{ELBO} &= \mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(\frac{p^\theta(\mathbf{u}, \mathbf{z})}{q^\phi(\mathbf{z}|\mathbf{u})}\right) \\ &= \mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(p^\theta(\mathbf{u}|\mathbf{z})\right) - \text{KL}\left(q^\phi(\mathbf{z}|\mathbf{u}) || p(\mathbf{z})\right) \end{aligned} \tag{10}$$

As motivated in [3], through posing the second term as a constraint with multiplier β, we will also consider the modified objective

$$\text{ELBO} = \mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(p^\theta(\mathbf{u}|\mathbf{z})\right) - \beta \cdot \text{KL}\left(q^\phi(\mathbf{z}|\mathbf{u}) || p(\mathbf{z})\right). \tag{11}$$

In the above, the first term promotes matching data, and the second term encourages the variational inference to more closely reproduce the true prior.

Next, we continue with further specification of $q^\phi(\mathbf{z}|\mathbf{u})$ and $p^\theta(\mathbf{u}|\mathbf{z})$. These conditional distributions are the *probabilistic encoder* and *probabilistic decoder*, respectively. Taking $p(\mathbf{z}) = N(\mathbf{0}, \text{I})$ (or $p(\mathbf{z}) = N(\mathbf{0}, \beta^{-1}\text{I})$), we parameterize $q^\phi(\mathbf{z}|\mathbf{u})$ through a mean and diagonal covariance, $N(\mu^\phi(\mathbf{u}), \mathrm{diag}(\sigma^\phi(\mathbf{u}))^2)$. We define $p^\theta(\mathbf{u}|\mathbf{z})$ as the pushforward of z through D^θ: $p^\theta(\mathbf{u}|\mathbf{z}) = D^\theta \# p(\mathbf{z})$. For $\mathbf{z} \in \mathbb{R}^d$ and $\mathbf{u} \in \mathbb{R}^{n_x}$, the aforementioned deterministic functions map as follows:

$$D^\theta : \mathbb{R}^d \to \mathbb{R}^{n_x}, \quad \mu^\phi : \mathbb{R}^{n_x} \to \mathbb{R}^d, \quad \sigma^\phi : \mathbb{R}^{n_x} \to \mathbb{R}^d. \tag{12}$$

In a standard VAE, these functions are chosen as neural networks. The crux of this work is to consider a different form of D^θ.

3 Solution Approach: Doubly Latent Neural Networks

3.1 Uncertain Neural Network PDEs

The form of the partial differential equation has thus far only been specified as $u_t = \mathcal{N}(x, u, u_x, u_{xx}; \mathbf{q})$, or $0 = \mathcal{N}(x, u, u_x, u_{xx}; \mathbf{q})$. Here, we further specify this. The operator $\mathcal{N}$ is parameterized as $\mathcal{N}^\psi$. There are several choices for the parameterization, such as with polynomials [5–7] or symolic neural networks [8]. We elect to follow Raissi [1], and parameterize $\mathcal{N}^\psi$ as a neural network. Importantly, unlike Raissi, we do not use a PINNs-like representation of the state when training $\mathcal{N}^\psi$ [9]. Instead, the state is computed through the solution operator $\mathcal{S}^h$. In other words,

$$u^\psi(x, \mathbf{q}) = \mathcal{S}^h[\mathcal{N}^\psi, \mathrm{BCs}]. \tag{13}$$

We will later use a gradient-based optimizer to update the parameters based upon this function. Thus, we require that u^ψ and hence $\mathcal{S}^h[\mathcal{N}^\psi, \mathrm{BCs}]$ is differentiable with respect to the parameters ψ. We use Newton's method within $\mathcal{S}^h$, which itself involves Jacobians of $\mathcal{N}^\psi$. Nontheless, this can still be handled through automatic differentiation in PyTorch [10]. An alternative method could compute this through the adjoint method [11], likely for lower memory cost.

3.1.1 Deterministic Example

Here, we briefly present the deterministic case of training $\mathcal{N}^\psi$ with this method involving the solution operator. Let $u(x)$ be the state of some system governed by a PDE $\mathcal{N}$, with Dirichlet boundary conditions. Consider the dataset $\{x_i, u_i\}_{i=1}^{n_x}$, such that $u(x_i) = u_i$ (we do not consider noise in our later studies). To determine $\mathcal{N}^\psi$, we train the parameters on the following loss function, which directly solves for u^ψ

$$L(\psi) = \frac{1}{n_x} \sum_{i=1}^{n_x} \left(\underbrace{\mathcal{S}^h[\mathcal{N}^\psi, \mathrm{BCs}]}_{u^\psi} (x_i) - u_i \right)^2 \tag{14}$$

The minimization of $L(\psi)$ over ψ can be done through the gradient-based method Adam [12]. Further implementation details will be discussed later.

3.2 Doubly Latent Neural Network

Here, we combine the previously mentioned ideas to give the main result of this work. We use the term "Doubly Latent Neural Network" to describe a particular VAE, where the decoder is the solution of an uncertain neural network PDE. That is, we take

$$[D^\theta(\mathbf{z})]_i = \mathcal{S}^h[\mathcal{N}^\psi, \mathrm{BCs}]\ (x_i, \mathbf{z}) \quad \text{for } i = 1, 2, \ldots, n_x. \tag{15}$$

In the above, we sample the approximation u^ψ at the spatial mesh points $\{x_j\}_{j=1}^{n_x}$ so that D^θ still maps $\mathbb{R}^d \to \mathbb{R}^{n_x}$. We will highlight later, that this is not necessary and is in fact a benefit of this approach – our architecture can generate solutions on any number of meshes. We also see that parameters of the decoder include the parameters defining the PDE, i.e. $\Psi \in \theta$. In our case, we even have $\{\psi\} = \theta$, as $\mathcal{S}^h$ and the BCs do not introduce any new parameters, although this is a viable option, as long as the solution is differentiable with respect to these parameters. For instance, we could parameterize $\mathcal{S}^h$ so that it weights forward, centered, and backward finite difference schemes. The parameters controlling these weights would then be added to θ too. This example would implement a form of upwinding [13].

Note also in this formulation, we purposefully use $\mathbf{z}$, as opposed to $\mathbf{q}$, where $\mathbf{z} \in \mathbb{R}^d$, $\mathbf{q} \in \mathbb{R}^{n_q}$, and generally $d \neq n_q$. This important distinction will be explored more later.

3.2.1 First Latent Space: VAE

In the Doubly Latent Neural Network, the first latent space is readily apparent. The Variational Autoencoder Structure includes a latent space. Within this latent space, we have $\mathbf{z} \in \mathbb{R}^d$, and we also regularize so that $\mathbf{z} \sim N(\mathbf{0}, \mathrm{I})$ (or $N(\mathbf{0}, \beta^{-1}\mathrm{I})$). Further, in practice we will always use $d < n_x$, making it clear that the data is compressed into this smaller latent space. The second latent space is more subtle and less concretely defined.

3.2.2 Second Latent Space: PDE

We begin by first considering an introductory example. Consider $f(\xi_1, \xi_2, \xi_3) = 0$, with $\xi_1, \xi_2, \xi_3 \in \mathbb{R}$. This defines a surface in $\mathbb{R} \times \mathbb{R} \times \mathbb{R} = \mathbb{R}^3$, but where the surface is at most defined by two directions in a local sense (there may need to be some further restrictions on f related to manifold theory). Thus, the equation fundamentally describes an object within $\mathbb{R}^3$, and this object locally resembles $\mathbb{R}^2$. With this example in mind, we return to the discussion of the second latent space. The PDE representation defines this second latent space, in a similar form to the previous example. We have that $\mathcal{N}^\psi(x, u, u_x, u_{xx}; \mathbf{z}) = 0$. For simplicity, we will focus on the case where $\mathbf{z}$ is fixed, and the PDE is assumed to be spatially invariant, giving $\mathcal{N}^\psi(u, u_x, u_{xx}) = 0$. Suppose that there is a solution u. (Possibly under some further assumptions), we have that $u \in \mathcal{C}_2$, $u_x \in \mathcal{C}_1$, and $u_{xx} \in \mathcal{C}_0$, where $\mathcal{C}_i$ is the space of functions having at least i bounded derivatives. Noting that $\mathcal{C}_2 \subset \mathcal{C}_1 \subset \mathcal{C}_0$, the input to $\mathcal{N}^\psi$ is in $\mathcal{C}_0 \times \mathcal{C}_0 \times \mathcal{C}_0 = (\mathcal{C}_0)^3$. By a sketchy continuation of the logic from the $\mathbb{R}$ example to our case of $\mathcal{C}_0$, we conclude that the PDE equation describes an object within $(\mathcal{C}_0)^3$, and this object locally resembles $(\mathcal{C}_0)^2$. Either of these spaces may be considered our second latent space. In this example, that latent space is either 2 or 3-dimensional, but where each dimension is a function space, rather than $\mathbb{R}$. Note, this analogy has fundamental issues, as $\mathbb{R}$ is topologically different from $\mathcal{C}_0$ (open, closed, complete). A more rigorous explanation might involve an analysis using Sobolev spaces such as $\mathcal{H}^1$, which have more comparable topological properties. This rigorous explanation would also rely on manifolds and Stochastic PDE theory [14].

Alternatively, the PDE can be seen as a latent space by considering its numerical solution. The solution of the PDE lies in $\mathbb{R}^{n_x}$, but there are only a few inputs to the PDE (u, u_x, u_{xx}). Thus, we can think of the PDE as mapping from a lower dimensional space (3 inputs) to the higher dimensional $\mathbb{R}^{n_x}$. Further, note that these two latent spaces are connected, as $\mathbf{z}$ is an input to the PDE. In other words, $\mathcal{N}^\psi$ is a Stochastic PDE, as $\mathbf{z}$ is random.

4 Results

We now consider an example to demonstrate this method, and to further explain its strengths and weaknesses. Consider the problem of 1D, steady-state thermal heat conduction. The (unknown) governing equation used to generate data is given as

$$\frac{\partial}{\partial x}\left(\kappa(x)\frac{\partial u}{\partial x}\right) - 1 = 0, \quad x \in (0,1). \tag{16}$$

The known, Dirichlet boundary conditions are

$$u(0) = 0, \qquad u(1) = 0.$$

The material is characterized by a Karhunen–Loève-like expansion

$$\kappa(x; \mathbf{q}) = \bar{K} + \sigma \sum_{i=1}^{n_q} \sqrt{\lambda_i}\phi_i(x) q_i, \tag{17}$$

taking $\bar{K} = 1$, $\sigma = 0.1$, $n_q = 10$, and $\{\lambda_i, \phi_i(x)\}_{i=1}^{n_q}$ as the analytically derived eigen-pairs of the covariance kernel

$$C_{KK}(x_1, x_2) = \exp\left(\frac{-|x_1 - x_2|}{\ell}\right), \quad (x_1, x_2) \in (0,1)^2. \tag{18}$$

Each component of random vector $\mathbf{q}$ is i.i.d. $U[-1,1]$. We can rewrite the governing equation in a more consistent form, as

$$\mathcal{N}(x, u, u_x, u_{xx}; \mathbf{q}) = u_x \kappa_x(x, \mathbf{q}) + u_{xx}\kappa(x, \mathbf{q}) - 1 = 0. \tag{19}$$

Using a 2nd-order centered finite difference method (with an iterative nonlinear solver), we solve this problem for $N = 100,000$ (Monte Carlo) realizations of $\mathbf{q}$, giving the dataset $\{\mathbf{u}^{(k)}\}_{k=1}^{N}$. Note, no values of $\mathbf{q}$ will be used for the training.

We will try to minimize the following objective function, derived from the ELBO with Eq. 14, and over multiple training instances:

$$L(\phi, \theta) = \underbrace{\sum_{k=1}^{N_{\text{train}}} \mathop{\mathbb{E}}_{\mathbf{z}\sim q^{\phi}(\mathbf{z}|\mathbf{u})} \left[\frac{1}{n_x}\sum_{i=1}^{n_x}\left(\underbrace{\mathcal{S}^h[\mathcal{N}^{\psi}, \text{BCs}]}_{u^{\psi}}(x_i; \mathbf{z}) - u_i^{(k)}\right)^2\right]}_{\text{match data}} + \beta \cdot \underbrace{\text{KL}\left(q^{\phi}(\mathbf{z}|\mathbf{u})||p(\mathbf{z})\right)}_{\text{regularize latent space}}. \tag{20}$$

Here, $N_{\text{train}} \ll N$. However, simply training the above often results in issues, due to the nature of $\mathcal{S}^h$. We will make a brief aside to highlight this issue and propose a temporary fix.

4.1 Difficulties of $\mathcal{S}^h$

The operator $\mathcal{S}^h[\mathcal{N}^{\psi}, \text{BCs}]$ involves several iterations of Newton's method (Eq. 8). This in turn, involves computing a Jacobian based on $\mathcal{N}^{\psi}$ and inverting it. We do not do this directly on $\mathcal{N}^{\psi}$, but instead on $\mathbf{f}$ (see Eq. 7), this will eventually involve derivatives of $\mathcal{N}^{\psi}$ with respect to the inputs of $\mathcal{N}^{\psi}$, and dependence can be clearly expressed through applying chain rule. Thus, we are concerned with inverting a matrix that depends on derivatives of $\mathcal{N}^{\psi}$, and back-propagating parameter derivative information through this operation. As outlined by SIREN [15], common activation functions like tanh and ReLU are poorly suited to this task, due to the lack of expressivity of their derivatives. Thus, we use the architecture proposed therein, which uses sin activation functions. We found this intervention to make quite a significant impact on the success of this method.

Further, even with the above parameterization, the equation we are trying to solve (Eq. 7) might not even have a solution. In other words, we may begin with $\mathcal{N}^{\psi}$, such that our initial values of ψ do not give $\mathcal{N}^{\psi} = 0$, for any u. In this case, $\mathcal{S}^h[\mathcal{N}^{\psi}, \text{BCs}]$ will not behave as expected. This may happen for just one $\mathbf{z}$, or for all $\mathbf{z}$, but in our experience, success on a few instances of $\mathbf{z}$ is often a good predictor for success on many more $\mathbf{z}$. In the early iterations of training, we believe that this issue manifests first as computing extremely large gradients. Another possible explanation for this case could be attributed to poor performance of Newton's method for this problem. While this is a reasonable assumption, we implemented a few checks to rule this option out. We experimented with different variants of Newton's method. Replacing the Jacobian with $\epsilon^{-1}\mathbf{I}$ gives gradient descent with step-size determined by ϵ. This term can also be added to the Jacobian. This addition helps avoid saddle-points (and helps stabilize the matrix inversion), which is an issue of Newton's method [16]. We can also add a damping factor to reduce the step taken by Newton's method. This helps in non-convex problems where the Jacobian quickly becomes a poor approximation farther from the current iterate. This idea can be extended to include a line search, but this was omitted due to the memory cost of also needed to perform automatic differentiation on this solution operator. However, these interventions proved unsuccessful to deal with the issue of exploding gradients.

Instead, upon creating $\mathcal{N}^{\psi}$, we iterate through random instances of ψ, discarding iterations where gradients explode. This is done offline and generally only requires a handful of iterations. As mentioned above, this is not a guarantee that $\mathcal{S}^h[\mathcal{N}^{\psi}, \text{BCs}]$ will work for all $\mathbf{z}$, although heuristically we do not see many issues once we have some initial success. Other alternatives might look at the residual values involved in the Newton solve or may implement a gradient clipping strategy. It is worth noting that this issue does resemble the problem of exploding gradients in deep neural networks. By allowing more iterations of the Newton solve, we generally see larger gradients (if $\mathcal{N}^{\psi}$ is not carefully initialized), just as adding more layers with certain activation functions causes exploding gradients.

4.2 Example Problem Results

We now present preliminary results of the example problem given above. We train on $N_{\text{train}} = 80$ instances of $\mathbf{u}$, for $n_x = 101$, with no knowledge of the underlying random variable $\mathbf{q}$. This is an important point – the method identifies the relevant stochastic inputs and their distributions. This representation may be different (possibly more efficient) than other choices to represent the uncertainty. While κ linearly depends on $\mathbf{q}$, the method may find a more efficient, *nonlinear* parameterization $\mathbf{z}$ with $d < n_q$. This is one potential benefit of this method. We use the loss in Eq. 20. The latent space dimension was set as $d = 5$. The architecture of $\mathcal{N}^{\psi}(x, u, u_x, u_{xx}, \mathbf{q})$ was a SIREN with 2 hidden layers, each of 10 hidden features. The ω parameters are set to 1. The encoder has the same architecture, but with $n_x - 2$ inputs, instead of the 9 in $\mathcal{N}^{\psi}$. The $\mu^{\phi}(\mathbf{u})$ and $\sigma^{\phi}(\mathbf{u})$ are just taken by adding extra linear layers to the end of the encoder. The full model was trained with $\beta \approx 10^{-5}$.

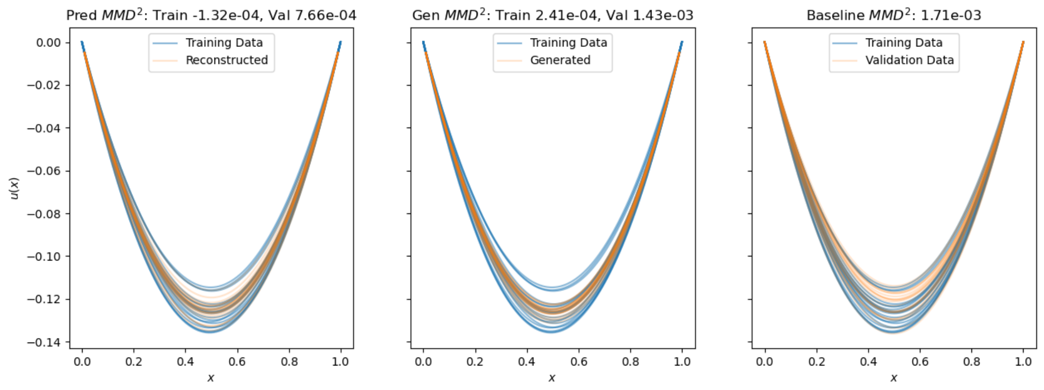

Figure 1: The left image compares the training data (blue) to the attempted reconstruction (orange). The center image shows the same training data against samples generated by the Doubly Latent Neural Network. The right figure shows a baseline comparison, where the orange are from the true distribution, and are not used at all during training. Each comparison shows 20 curves of each color.

Fig. 1 shows a proofof concept, but leaves much to be desired. First, the reconstructed and generated curves resemble one another, in a distributional sense, which is a good sign. However, as highlighted by Fig. 2, the generated samples have a smaller variance than the true samples. This is a common issue for VAEs, arising from the use of the KL divergence in the loss function, which can be combatted by using MMD or InfoVAE. We see also that the generated distribution at $u(0.5)$ is overly Gaussian, as opposed to the true distribution, which is closer to a uniform distribution.

Importantly, this method is proposed to allow for a shared framework with PDEs. Thus, as opposed to many deep learning methods, we can increase n_x, *at inference time*, and theoretically, also change the boundary conditions. However, the latter gives terrible results, even when just changing $u(1) = 0.01$, so we note that this method has failed in this regard, for this uninformative parameterization and relatively sparse training data. We show the results of the former, refining the mesh to now be 10 times as fine! This is shown in Fig. 3 below, where the results at this higher resolution resemble those at the lower resolution. Fig. 4 in the Appendix shows a comparison similar to Fig. 2, but just for this refined version. These figures demonstrate the proposed model architecture can at least rely on some of the helpful properties of numerical solutions of PDEs.

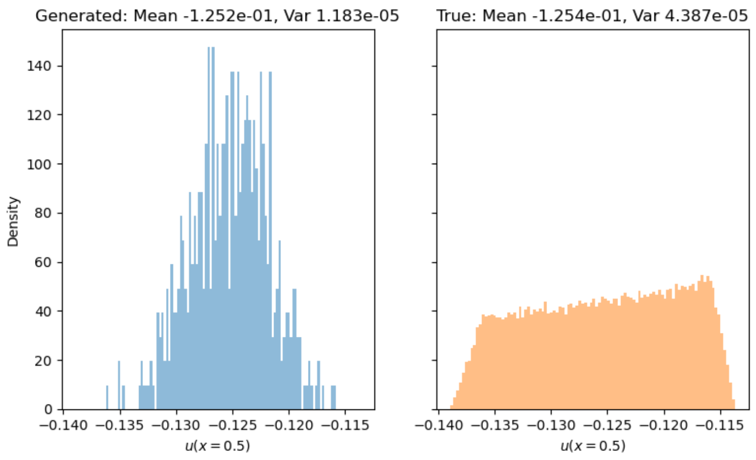

Figure 2: We consider the scalar $u(0.5)$. The left shows 500 generated samples from our model, and the right shows 500 samples from the true distribution.

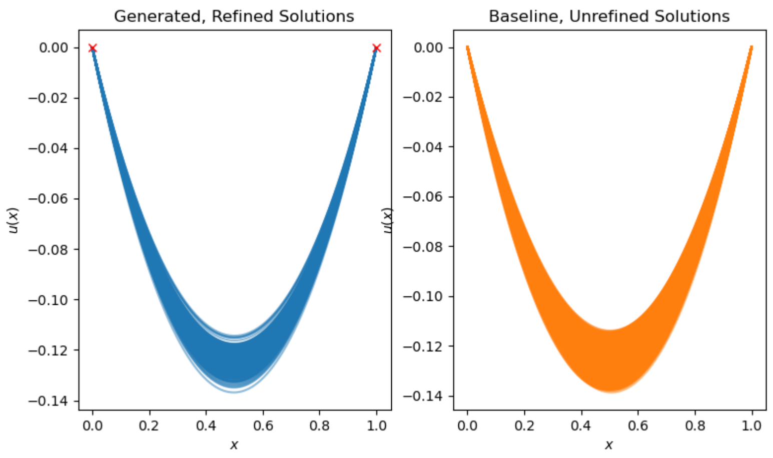

Figure 3: Similar to the right two images in Fig. 1, we compare generated samples from our model versus samples from the true distribution. We show 500 samples of each. However, the left plots are from the *refined mesh* with $10 \cdot n_x$ mesh points, while the model was only trained on n_x mesh points.

5 Conclusions and Discussions

We presented the Doubly Latent Neural Network, which proposes the decoder of a VAE as the solution of a neural network PDE. This method is motivated by the difficulty to identify the relevant inputs and determine their distributions for Stochastic PDEs. Instead, the relevant inputs are assumed to Gaussian i.i.d. ($\mathbf{z}$), and their nonlinear mapping in the PDE is learned. We explained the two-latent spaces involved in this model: the regular VAE latent space and the space involving the manifold defined by the PDE. We presented some challenges with the training of these models, and proceeded to a numerical example. The numerical example of steady-state heat conduction with an uncertain material demonstrates a proof of concept for the method, beyond the theoretical motivations. However, the method has issues common to VAEs, such as underestimating variance and sensitivity to the hyperparameter β. Unfortunately, the model does not extrapolate to even slight changes in the boundary conditions. Yet, the method performs remarkably well under mesh refinement: still generating accurate solutions for meshes 10 times finer than the training mesh.

References

[1] Maziar Raissi. *Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations*. Tech. rep. arXiv: 1801.06637v1. 2018. URL: `https://github.com/maziarraissi/DeepHPMs` (visited on 06/20/2022).

[2] Diederik P. Kingma and Max Welling. *Auto-Encoding Variational Bayes*. arXiv:1312.6114 [cs, stat]. Dec. 2022. URL: `http://arxiv.org/abs/1312.6114` (visited on 04/05/2024).

[3] Irina Higgins et al. " -VAE: LEARNING BASIC VISUAL CONCEPTS WITH A CONSTRAINED VARIATIONAL FRAMEWORK". en. In: (2017).

[4] Nuojin Cheng et al. *Bi-fidelity Variational Auto-encoder for Uncertainty Quantification*. arXiv:2305.16530 [cs, math, stat]. Oct. 2023. DOI: `10.48550/arXiv.2305.16530`. URL: `http://arxiv.org/abs/2305.16530` (visited on 04/28/2024).

[5] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: 113.15 (2016). DOI: `10.1073/pnas.1517384113`.

[6] Jacqueline Wentz and Alireza Doostan. "Derivative-based SINDy (DSINDy): Addressing the challenge of discovering governing equations from noisy data". In: *Computer Methods in Applied Mechanics and Engineering* 413 (Aug. 2023). arXiv:2211.05918 [math], p. 116096. ISSN: 00457825. DOI: `10.1016/j.cma.2023.116096`. URL: `http://arxiv.org/abs/2211.05918` (visited on 03/13/2024).

[7] Daniel A. Messenger and David M. Bortz. "Weak SINDy for partial differential equations". In: *Journal of Computational Physics* 443 (Oct. 2021). arXiv: 2007.02848 Publisher: Academic Press, p. 110525. ISSN: 0021-9991. DOI: `10.1016/J.JCP.2021.110525`. (Visited on 06/13/2022).

[8] Zichao Long, Yiping Lu, and Bin Dong. "PDE-Net 2.0: Learning PDEs from Data with A Numeric-Symbolic Hybrid Deep Network". In: *Journal of Computational Physics* 399 (Dec. 2019). arXiv:1812.04426 [physics, stat], p. 108925. ISSN: 00219991. DOI: `10.1016/j.jcp.2019.108925`. URL: `http://arxiv.org/abs/1812.04426` (visited on 12/08/2023).

[9] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (Feb. 2019), pp. 686–707. ISSN: 0021-9991. DOI: `10.1016/j.jcp.2018.10.045`. URL: `https://www.sciencedirect.com/science/article/pii/S0021999118307125` (visited on 05/09/2024).

[10] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: (2019). arXiv: 1912.01703v1. (Visited on 07/09/2023).

[11] Richard C. Aster, Brian Borchers, and Clifford H. Thurber. "Nonlinear Inverse Problems". In: *Parameter Estimation and Inverse Problems* 8 (2019). ISBN: 9780128046517, pp. 257–278. DOI: `10.1016/b978-0-12-804651-7.00015-8`.

[12] Diederik P Kingma and Jimmy Lei Ba. "ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION". In: (2014). arXiv: 1412.6980v9.

[13] Randall J. LeVeque. "Advection Equations and Hyperbolic Systems". In: *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2007, pp. 201–231. ISBN: 978-0-89871-629-0. URL: `https://epubs.siam.org/terms-privacy`.

[14] Martin Hairer. *An Introduction to Stochastic PDEs*. arXiv:0907.4178 [math]. July 2023. URL: `http://arxiv.org/abs/0907.4178` (visited on 03/18/2024).

[15] Vincent Sitzmann et al. "Implicit Neural Representations with Periodic Activation Functions". In: (2020). arXiv: 2006.09661v1. (Visited on 07/08/2023).

[16] Stephen Boyd and Lieven Vandenberghe. "Convex Optimization". In: (2004). ISBN: 9780521833783. URL: `http://www.cambridge.org` (visited on 08/05/2022).

A Appendix

A.1 ELBO Derivation

$$\begin{aligned}\log(p^\theta(\mathbf{u})) &= \mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(\frac{p^\theta(\mathbf{u},\mathbf{z})}{p^\theta(\mathbf{z}|\mathbf{u})}\frac{q^\phi(\mathbf{z}|\mathbf{u})}{q^\phi(\mathbf{z}|\mathbf{u})}\right)\\ &= \mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(\frac{q^\phi(\mathbf{z}|\mathbf{u})}{p^\theta(\mathbf{z}|\mathbf{u})}\right) + \mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(\frac{p^\theta(\mathbf{u},\mathbf{z})}{q^\phi(\mathbf{z}|\mathbf{u})}\right)\\ &= \mathrm{KL}\left(q^\phi(\mathbf{z}|\mathbf{u})||p^\theta(\mathbf{z}|\mathbf{u})\right) + \mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(\frac{p^\theta(\mathbf{u},\mathbf{z})}{q^\phi(\mathbf{z}|\mathbf{u})}\right)\\ &\leq \underbrace{\mathop{\mathbb{E}}_{q^\phi(\mathbf{z}|\mathbf{u})} \log\left(\frac{p^\theta(\mathbf{u},\mathbf{z})}{q^\phi(\mathbf{z}|\mathbf{u})}\right)}_{\text{ELBO}}\end{aligned} \tag{21}$$

A.2 Refinement Figure

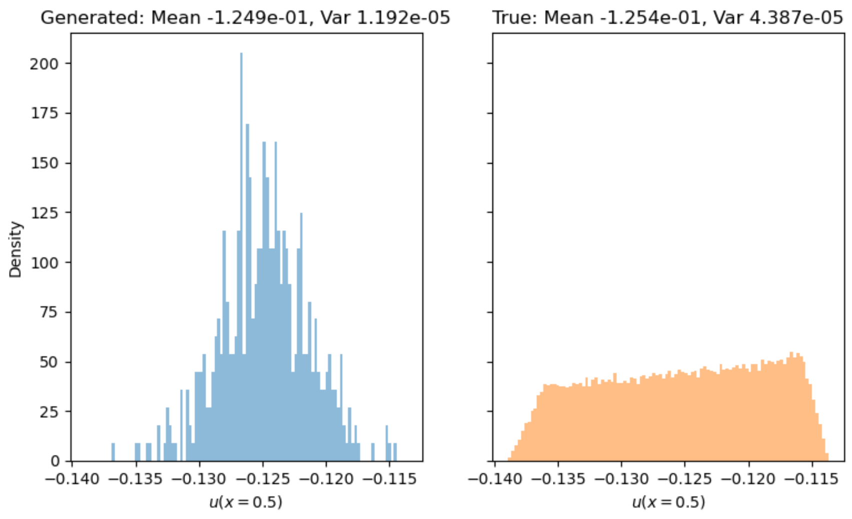

Figure 4: *For the refined case*, We consider the scalar $u(0.5)$. The left shows 500 generated samples from our model, using $10 \cdot n_x$ mesh points, and the right shows 500 samples from the true distribution, on the training mesh of n_x points. Our model behaves about the same as for the original n_x points (Fig. 2), but now sampled at a much higher resolution at inference time.