POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Report

Uncertainty Quantification in Scientific Machine Learning

Numerical Analysis for Partial Differential Equations

Authors: Andrea Bonifacio

Academic year: 2022-2023

---

1. Introduction

In this report I'll try to review the main Uncertainty Quantification (UQ) methods in Neural Networks (NNs), with a focus on Physics Informed Neural Networks (PINNs). To do so, I'll present a brief recap on UQ and NNs, followed by the methodology to quantify uncertainty while solving SDEs using PINNs. [1]

1.1. Neural Networks

Today NNs are all over the news, they are the tools behind many new inventions, like large language models. Here I'll describe the simplest possible architecture for a neural network.
In its most basic configuration, a feed-forward neural network is made up of three layers:

- Input layer,
- Hidden layer,
- Output layer.

While the first and last layers are quite self-explanatory (the input layer takes a vector $\boldsymbol{x}$ D-dimensional as an input and the output layer returns an object compatible with the dimensions of the dataset), the hidden layer is made up of multiple units, called neurons, which are basically vector transformations, to which is applied a nonlinear function. The general idea behind is to minimize the loss between the predictions of the model and the real values of the training dataset, by continuously modifying the neurons. Once a satisfying loss between the prediction and real results is reached, the model is ready to be used.
One of the main features of NNs is that they work really well as function approximators, so one of the logical step after their introduction was to use them to solve PDEs, Lagaris et al 1998, but, at the time, it didn't lead anywhere until recently, when the advent of new tools and machine learning frameworks made possible to work again on solving PDEs using neural networks built up for the task.

1.2. Uncertainty Quantification

To describe uncertainty in a model, we must distinguish between the two different kinds. The model itself cannot represent reality as a whole, it will always be built up on assumptions that adds up creating epistemic uncertainty, which must be acknowledged, as it is irreducible. In the real world, data are collected by humans and sensors, which are both prone to errors and noise. This is the second kind of uncertainty, called aleatoric. To sum up, the total uncertainty in a model is made of two components:

$$\mathrm{TU} = \mathrm{EU} + \mathrm{AU}.$$

To deal with epistemic uncertainty, it is possible to use a probability distribution over the model parameters. Given a dataset $D = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ where $\boldsymbol{x}_i \in \mathbb{R}^D$ are the inputs and $y_i \in \{1, \dots, C\}$ are the corresponding outputs in C different classes. The idea is to optimize the parameters ω of a function $y = f^\omega(\boldsymbol{x})$. The Bayesian

approach defines a model likelihood $p(y|\boldsymbol{x}, \omega)$, from which we can obtain the posterior distribution for a given dataset, thanks to Bayes' theorem:

$$p(\omega|X, Y) = \frac{p(Y|X, \omega)p(\omega)}{p(Y|X)}.$$

Now, given a test sample $\boldsymbol{x}^*$, a class label can be predicted as

$$p(y^*|\boldsymbol{x}^*, X, Y) = \int p(y^*|\boldsymbol{x}^*, \omega)p(\omega|X, Y)\, d\omega.$$

2. Problem setup

The problem can be formulated in the following way

$$\begin{cases} \mathcal{N}_x\left[u(x;\omega); k(x;\omega)\right], & x \in \mathcal{D}, \omega \in \Omega, \\ \mathcal{B}_x\left[u(x;\omega)\right] = 0, & x \in \Gamma, \end{cases} \tag{1}$$

where $\mathcal{N}_x$ is a differential operator, $\mathcal{D}$ the domain, Ω the random space and $u(x;\omega)$ the solution. On the boundary Γ we have the boundary conditions imposed by the operator $\mathcal{B}_x$. $k(x;\omega)$ represent the random parameter.
There are two possible problems:

- Forward problems: in this case, we know the distribution of $k(x;\omega)$ everywhere on the domain, and the quantity of interest is $u(x;\omega)$.
- Inverse problems: when the information about $k(x;\omega)$ is partially known, but there is a lot more information about $u(x;\omega)$ it is possible to infer the full distribution of k.

3. Methodology

3.1. PINNs for deterministic systems

Here is how to solve differential equations using PINNs. To do so, (1) must be rewritten replacing the random inputs with a set of finite parameters, to transform it in a deterministic problem.

$$\begin{cases} \mathcal{N}_x\left[u;\eta\right], & x \in \mathcal{D}, \\ \mathcal{B}_x\left[u\right] = 0, & x \in \Gamma. \end{cases} \tag{2}$$

The neural network will be denoted by $\hat{u}(x;\theta)$, which is the approximation of the solution $u(x)$ with a specific set of parameters θ dependent on the network. In a classical deep learning setting, the network should have one constraint, which is to reproduce the values in the dataset. In a physics informed settings, there is a second constraint, which is that the network should comply with the physical laws imposed by (2).
To do so, a second network is defined

$$\hat{f}(x;\theta,\eta) := \mathcal{N}_x\left[\hat{u}(x;\theta);\eta\right], \tag{3}$$

which is computed straightforwardly from $\hat{u}$. The parameters of the second network are the same of the first one. Assuming a dataset with N_u observations on u collected at $\{x_u^{(i)}\}_{i=1}^{N_u}$ and N_c the number of collocation point in which $\hat{f}$ will be evaluated.

Algorithm 1 PINN for solving differential equations

Step 1: Specify the training set

$$\hat{u} : \{(x_u^{(i)}, u(x_u^{(i)}))\}_{i=1}^{N_u}, \quad \hat{f} = \{x_f^{(i)}, 0\}_{i=1}^{N_c}.$$

Step 2: Construct a NN $\hat{u}(x;\theta)$ with random initialized parameters θ.
Step 3: By using automatic differentiation, construct the residual network $\hat{f}(x;\theta,\eta)$.
Step 4: Specify a loss function that includes both networks

$$\mathcal{L}(\theta,\eta) = \frac{1}{N_u}\sum_{i=1}^{N_u}[\hat{u}(x_u^{(i)};\theta) - u(x_u^{(i)})]^2 + \frac{1}{N_c}\sum_{i=1}^{N_c}\hat{f}(x_f^{(i)};\theta,\eta)^2. \tag{4}$$

Step 5: Train the networks to find the best parameters by minimizing the loss function:

$$\theta = \arg\min \mathcal{L}(\theta,\eta)$$

3.2. Moving to a stochastic setting

The idea now is to analyze a stochastic problem, which brings back (1). A new dataset is needed to collect all the random events. This dataset will be made up of N_k measurement at $x_k^{(i)}$ locations in the domain. Given N measurements, the random instance at the s^{th} event will be called ω_s. Now, defining $k_s^{(i)} = k(x_k^{(i)};\omega_s)$ and $u_s^{(i)} = u(x_u^{(i)};\omega_s)$, the new dataset is built up like this:

$$\mathcal{S}_t = \{\{(x_k^{(i)}, k_s^{(i)})\}_{i=1}^{N_k}, \{(x_u^{(i)}, u_s^{(i)})\}_{i=1}^{Nu}, \{(x_f^{(i)}, 0)\}_{i=1}^{N_f}\}_{s=1}^{N}.$$

The next step is the arbitrary polynomial chaos (aPC) expansion, which will require:

1. Dimension reduction;
2. Constructing the aPC basis;
3. Building the NN-aPC network as a surrogate model and train the network for each mode.

3.2.1 Dimension reduction with PCA

To find a lower dimensional space, the proposed technique is principal component analysis (PCA), which finds a smaller number of modes able to explain almost all the dataset with enough accuracy. In this case, the reduced dataset will be the one about k. Let K be the $N_k \times N_k$ covariance matrix of the measurements on k

$$K_{i,j} = \text{Cov}(k^{(i)}, k^{(j)}).$$

Given λ_l and ϕ_l be the l-largest eigenvalue and its eigenvector, it is possible to write

$$K = \mathbf{\Phi}^T \mathbf{\Lambda} \mathbf{\Phi},$$

where $\mathbf{\Phi} = \phi_1, \phi_2 \ldots, \phi_{N_k}$ is an orthonormal matrix and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{N_k})$ is a diagonal matrix.

4. Numerical Results

5. Conclusions

References

[1] CTAN. BiBTeX documentation, 2017.