Physics informed neural networks for inverse problems in acoustic

January 6, 2021

Pre-Study Master Project
Fall 2020

OLSEN Nils
nils.olsen@epfl.ch

Professor : B. Lecampion
Professor : S. Chaillat

Contents

1 Introduction **1**

2 Literature Review **2**

3 Wave equation **4**

3.1 Acoustic wave propagation . . . 4

3.2 Elastic wave equation . . . 5

4 Physics Informed Neural Network **6**

4.1 Resources . . . 9

4.1.1 Libraries . . . 9

4.1.2 Github repositories . . . 10

5 Methodology and Objectives **11**

5.1 Methodology . . . 11

5.2 List of Objectives . . . 12

List of Figures

1 Velocity models, (a) homogeneous, (b) layered and (c) Earth-Realistic - Source : Moseley et al. [2020] . . . 5

2 Wave propagation in homogeneous isotropic elastic medium when a Gaussian Pulse is used as a source at the centre of the mesh - Source : Goyal [2017] . . . 5

3 Dense block with $L = 3$ layers and growth rate $K = 2$ - Source : Zhu and Zabaras [2018] . . . 7

4 Illustration of the encoder/decoder architecture - Source : Zhu and Zabaras [2018] 8

5 Results from the 1D Burger's equation modelled with FEM and solved with SciANN . . . 10

Abstract

Finite elements methods (FEMs) and finite difference methods (FDMs) have always been used to solve partial differential equations (PDEs). In the recent years with the advance in computational power and the availability of software, artificial neural networks (ANN) can be used as a mesh free methods to approximate physical systems. Combining sparse or even no data, physics informed neural networks (PINNs) can be trained simultaneously on available data and the governing differential equations to fit a specific model, to compute the solution of an ordinary and partial differential equations or to identify a parameter (model inversion).

The pre-study will focus on the main techniques used to train a PINN for surrogate modelling and inverse problem for the wave equation in geotechnics problems.

1 Introduction

Over the past decade, major breakthroughs were made in recording data and computational power. This allowed many fields in machine learning (ML) and more specifically problems using artificial neural networks (ANN) such as in image classification and other computer vision tasks, speech recognition and autonomous driving to be revolutionised. This increased the interest and the availability of data and open source softwares for machine learning applications such as Sklearn (Pedregosa et al. [2011]), Tensorflow (Abadi et al. [2016]), Keras (Gulli and Pal [2017]) and PyTorch (Paszke et al. [2019]). This data driven framework has made advancements in engineering applications such as in earthquake detection (Ross et al. [2019]), fluid mechanics and turbulence modelling (Haghighat and Juanes [2021]). Those traditional ML methods use deep neural networks and are trained using a lot of data. In those image classification tasks or autonomous driving, the large amount of data allow the networks to discover patterns in the data. One can use them to classify for example different geological units and predict the system behaviour under wide range of conditions.

It is convenient to look at how much data are available versus how much physics are known from the problem. For some engineering problems with fully known physics, their solutions can be modelled with differential equations describing conservation laws (Tartakovsky et al. [2020]). However, in most engineering applications data acquisition is an expensive and time demanding task. Data are often, if not always scarce. However some physics are present in these problems (Karimpouli and Tahmasebi [2020]). Despite huge advance in those areas, deep learning has not yet been widely used in the field of scientific computing and predictive problems with not fully known physics and sparse data presents a significant challenge. In this small data regime, if the state-of-the-art machine learning techniques (e.g., deep, convolutional or recurrent neural networks) are trained using only the available data, they will fail to provide any guarantees of convergence and lack robustness. The networks would fail to converge mostly due to the complexity of the system and un-optimised parameters (Geneva and Zabaras [2019]). Because of high spatial heterogeneity of natural systems, sparse measurements do not allow to cover the whole distribution of the properties (Tartakovsky et al. [2020]). For example, it is impossible to accurately learn the whole spatial distribution of porosity in a sub-surface system from few data only.

Solving partial differential equations (PDE) in the weak or in the strong form via deep learning has emerged as a new sub-field in the machine learning under the name of Physics-Informed Machine Learning (PIML) according to Raissi et al. [2019], Physics-Constrained Machine Learning (PCML) for Zhu et al. [2019] or Scientific Machine Learning (SciML) according to Lu et al. [2020]. Approximating PDEs is normally the task of the finite difference method (FDM) and the finite element method (FEM) which are mesh-based methods. Compared to the traditional mesh-based methods, deep learning could be a mesh-free approach by taking advantage of the automatic differentiation, and could break the curse of dimensionality to approximate PDEs (Lu et al. [2020]).

In many situations, such as optimization or inverse problems, a large number of repeated simulations are required prioritizing the computational efficiency of the numerical simulator. Often surrogate models are used to ease this computational burden by providing a fast approximate model that can imitate a standard numerical solver at a significantly reduced computational cost (Geneva and Zabaras [2019]). In PINN, the networks are trained simultaneously on available data and the governing differential equations. Once the models are trained, they can be used as a surrogate model to approximate a physical system but also for uncertainty quantification and propagation in problems governed by PDEs. The networks can represent the states, some space dependent coefficients and some constitutive relationships. From (Haghighat and Juanes [2021]) the three main areas covered by PINN are (i) model fitting, (ii) solution of ordinary and partial differential equations and (iii) model inversion (parameter(s) identification).

This report focuses on the different methods used to train PINNs for the wave equation. In section 2 a literature review will present the state of the art research articles regarding the fields covered by PINNs when the flow and the wave equations are solved. Combining multiple fields of research allows to have a better understanding at all the methods used to solve PINNs in order to implement a surrogate model and to inverse parameters. Then, in section 3, the wave equation which will be the main subject of the master thesis will be presented and its parameters explained. In section 4, the methodology used to train and evaluate PINN models will be presented for surrogate modelling and inverse problem alongside some resources available such as Python libraries and GitHub repositories that can be useful for the master thesis. Because the amount of information covering the field of PINNs to solve PDEs is quite immense, a methodology will be presented at the end in section 5 in order to guide the main steps for the master thesis.

2 Literature Review

To solve the PDEs, discretisation methods are used such as finite difference or finite element and iterative time stepping schemes are then implemented to solve them. Surrogate models are usually built for problems that require repetitive but expensive simulations for determinsitsic design, or uncertainty propagation for example. When large 3D solutions have to be computed then up to billions of parameters in the mesh must be computed and the computational time explodes.

As previously presented in the introduction, deep learning models such as deep neural network can be used in PINNs to approximate a physical system modelled with differential equations. Learning an ordinary or a partial differential equation (ODE and PDE respectively) using a constraint based loss functions is not a new domain of research. The earliest related works are more than two decades old as presented in Geneva and Zabaras [2019] and Mo et al. [2019].

In the framework presented by Raissi et al. [2019], deep fully convolutionnal neural networks are used to solve a supervised learning tasks. These networks were first used and already showed promising and impressive performances for surrogate modelling of forward models with high-dimensional input and output fields (Geneva and Zabaras [2019]). Using only a few points for training, the solution can be computed anywhere on the domain. This enables a mesh free solution that can evaluate the PDEs anywhere on the domain while being trained on only a few points. The same kind of networks have been used in Moseley et al. [2020], Sun et al. [2020], Tartakovsky et al. [2020], Shukla et al. [2020], Rao et al. [2020], Karimpouli and Tahmasebi [2020] or even in Python libraries such as Lu et al. [2020] and Haghighat and Juanes [2021]. However for these kind of models, if the initial conditions or the boundary conditions or the properties on the domain change, then the model must be retrained (Geneva and Zabaras [2019]).

To allow the computation anywhere on the domain without retraining the model and better generalisation capabilities, an other approach is to treat the inputs and outputs as images and use convolutional neural networks instead of fully connected neural networks. In Geneva and Zabaras [2019], the model performs an image-to-image regression task in order to predict from an initial field, multiple output fields. Surrogate models are built in Mo et al. [2019] and in Zhu et al. [2019] using also deep convolutional encoder-decoder networks to model the uncertainty of dynamic multi-phase flows and predict the Darcy's flow respectively.

In the work of Geneva and Zabaras [2019], the authors introduce a deep auto-regressive dense encoder-decoder to predict transient PDEs and they extend their model to a Bayesian framework which allow uncertainty quantification (UQ). Regarding the UQ, one wants to compute the distribution of parameters of great importance.Zhu et al. [2019] uses Gaussian processes for parameter inversion and the Kullback-Leibler divergence between the model predictive density and the reference conditional density is minimised. In Sun and Wang [2020], the flow field is reconstructed with a Bayesian approach using the Physics-Constrained Stein Variational Gradient Descent.

To my best knowledge, very few papers dealt with PINNs in 3D and only Mo et al. [2020] implemented a convolutional adversarial auto-encoder to find the parameters of a non-Gaussian conductivity fields in 2D and 3D.

More specifically for the wave equation, Moseley et al. [2020] implemented a fully connected deep neural network to solve the wave equation. Especially for the wave equation, the authors noticed that deep neural networks typically rely entirely on their training data and therefore perform poorly outside of it. However this was not the case for Raissi et al. [2019] as their deep learning framework for surrogate modelling and inverse problems showed good generalisation capabilities outside the training data.

In an other article written by Shukla et al. [2020], the wave equation was used to detect cracks as the wave velocity is a good proxy for the defects. The authors allowed the wave velocity to be space dependent on the domain and when the velocity had a brutal change the crack could be detected on the domain. Karimpouli and Tahmasebi [2020] uses the Gaussian process for parameters inversion to inverse the wave velocity parameter first using the acoustic 1D wave equation and then for the 2D elastic wave equation, the authors were able to invert also the compressional and shearing waves velocity (V_p and V_s) as well as the density ρ of the medium.

Smith et al. [2020] solved the Eikonal equation which characterises the first time of arrival in 3D heterogeneous medium given a source-receiver pair to predict the velocity in different soil layers. In Rao et al. [2020], the authors implemented a PINN to model elastodynamics outputs and more particularly the elastic wave equation in fully confined domain, infinite domain and semi-infinite domain with "hard" enforced boundary conditions where most of the papers enforced the boundary conditions in a "soft" way. The displacement and the stress fields are computed by the PINN.

3 Wave equation

The wave equation is used to model a lot of different physical, natural and technological problems. Using waves, one can transport information along different scales. In geophysics, one can use waves to simulate earthquake, estimate subsurface structure, characterise the interior of the Earth and other planets or carry out non destructive testing (Moseley et al. [2020]).

The wave equation is an hyperbolic PDEs and those PDEs (also 1D Burger's equation) in particular can exhibit sharp discontinuities. they can be difficult to solve for most traditional numerical methods (Sun et al. [2020]). To simulate wave propagation, FEM is traditionally used. The wave equation is discretised and iterative time stepping schemes are used to solve it. FEM has been developed decades ago and benefits from highly sophisticated algorithms to model large range of physics and anisotropy.

They are two popular approaches to simulate wave propagation : the acoustic and the elastic wave equation. The acoustic wave equation is different than the elastic. In the first one a scalar is used for the pressure's magnitude where as in the second one a vector is used which gives the pressure and the direction.

The wave equations as solved in Moseley et al. [2020] and Karimpouli and Tahmasebi [2020] are not be confused with the Eikonal equation presented by Smith et al. [2020] which give the first time of arrival of a wave compared to the variation of pressure or velocities for the first.

3.1 Acoustic wave propagation

The acoustic wave propagation governs the propagation of pressure waves in liquid or gases. Thus shear waves or longitudinal waves motion is not possible. The acoustic wave equation is given by the following formula :

$$\rho \nabla \cdot \left(\frac{1}{\rho} \nabla p \right) - \frac{1}{\nu^2} \frac{\partial^2 p}{\partial t^2} = -\rho \frac{\partial^2 f}{\partial t^2} \tag{1}$$

in this setting $p(t, x)$ describes the pressure response in the acoustic medium known as the wavefield. The $f(t, x)$ term is the point source term for the source injection. ν is the wave velocity in the medium and ρ its density. When the density of the medium is homogeneous and the source term negligible, Equation 1 can be simplified to obtain the canonical form of the wave equation :

$$\nabla^2 p - \frac{1}{\nu^2} \frac{\partial^2 p}{\partial t^2} = 0 \tag{2}$$

According to Moseley et al. [2020], the wave equation is difficult to solve despite its linearity. Indeed one can observe complex phenomena when waves propagate in complex media. Refraction, reflection and transmission can occur at boundaries between layers. A large number of waves can interact in different velocity regions with different range of amplitude and frequencies as presented in Fig. 2.

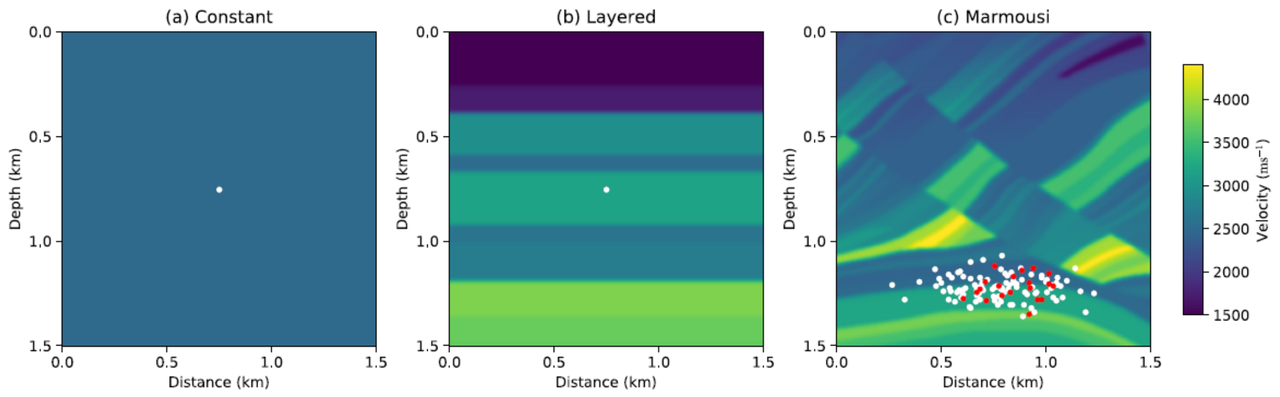

Figure 1: Velocity models, (a) homogeneous, (b) layered and (c) Earth-Realistic - Source : Moseley et al. [2020]

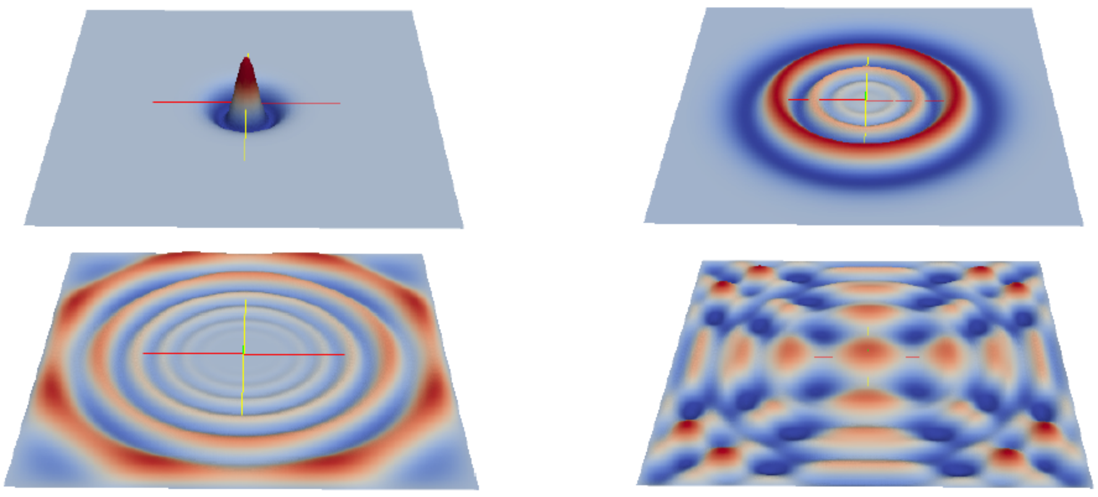

Figure 2: Wave propagation in homogeneous isotropic elastic medium when a Gaussian Pulse is used as a source at the centre of the mesh - Source : Goyal [2017]

3.2 Elastic wave equation

The vector form of the wave equation means that waves will have not only a scalar magnitude as for the pressure wave solution, but also a direction. This is used for wave types that exhibit compression and transverse particle motion. The wave propagation in the earth can be described by the elastic wave equation which is given by the following equation :

$$\rho \frac{\partial^2 u_i}{\partial t^2} - \frac{\partial \sigma_{ij}}{\partial x_j} = 0 \tag{3}$$

where σ is the stress given by :

$$\sigma_{ij} = \lambda u_{k,k} \delta ij + \mu (u_{i,j} + u_{j,i}) \tag{4}$$

λ and μ represent Lame's parameters. Specifically for seismic waves, μ is the propagation materials shear modulus. Compression and shearing waves can also occurs for example.

When one desire to model the wave equation in FEM the weak form of the PDE has to be given to the numerical solver. Also, the Courant-Friedrichs-Lewy (CFL) condition has to be respected if one wants to reach convergence while solving hyperbolic PDEs systems. For the wave equations it establishes the following relationship :

$$c\Delta t \leq \Delta x \tag{5}$$

with c the velocity, Δt is the time step and Δx is the mesh resolution. It means that the distance the wave travels in a single time step cannot exceed the size of a single element in the mesh. To generate waves in FEM, the Fenics library is used for the moment (Alnæs et al. [2015]). It is library available in Python and allows to simulate the propagation of waves. However one has to compute the weak form of the equation. I was not able to do it and I refer the reader to read the conversion given in Goyal [2017]. When a standard simple homogeneous medium is used as a domain (see Fig. 1) then the resulting wave propagation can be observed in Fig. 2.

4 Physics Informed Neural Network

This section will present some characteristics on how physics are implemented in neural network to create a PINN. A PINN uses deep learning methods which belongs to the machine learning area. In machine learning one can find methods such as decision tress, logistic regression, support vector machine, clustering methods and deep learning with all the artificial neural networks (ANNs). The domain of ANNs consists of fully connected neural networks, convolutional networks, recurrent neural networks, residual networks and reinforcement learning just to cite a few.

When neural networks are trained the task can be supervised or unsupervised. In the framework presented by Raissi et al. [2019], the neural networks are trained to solve a supervised learning task which consists in predicting an output $\hat{y}$ given a certain input x. The predicted output is compared to the true output y. If there is a mismatch between the two, the model will recursively update the weights via automatic differentiation and back-propagation algorithm during the training phase. A PINN uses the automatic differentiation to compute the partial derivative of the PDE.

From Haghighat and Juanes [2021], Mo et al. [2019] and Goodfellow et al. [2016], a single-layer forward neural network with inputs $\mathbf{x} \in \mathbb{R}^m$, outputs $\mathbf{y} \in \mathbb{R}^n$ and d hidden units is constructed as :

$$\mathbf{y} = \mathbf{W}^1 \sigma(\mathbf{W}^0 \mathbf{x} + \mathbf{b}^0) + \mathbf{b}^1 \tag{6}$$

with $\mathbf{W}^0 \in \mathbb{R}^{d\times m}, \mathbf{W}^1 \in \mathbb{R}^{n\times d}$ are the weights and $\mathbf{b}^0 \in \mathbb{R}^d, \mathbf{b}^1 \in \mathbb{R}^n$ are the biases. σ is the activation. A single forward network can approximate any measurable function independently of the size of input features m or the activation function. This is a useful property when one wants to solve PDEs. We can construct a general L-layers fully connected neural network as composition of Σ^i functions :

$$\mathbf{y} = \Sigma^L \circ \Sigma^{L-1} \circ \cdots \circ \Sigma^0(\mathbf{x}) \tag{7}$$

with σ^i being an activation functions. This allows to create nonlinear transformations. Some common activation functions are:

- ReLu : $\hat{x} \rightarrow \hat{x}^+$
- sigmoid : $\hat{x} \rightarrow 1/(1 + e^{\hat{x}})$
- tanh : $\hat{x} \rightarrow (e^{\hat{x}} - e^{-\hat{x}})/(e^{\hat{x}} + e^{-\hat{x}})$

Fully connected neural networks may lead to an extremely large number of network parameters. convolutional neural networks are commonly used to greatly reduce the number of parameters being learnt since they lead to sparse connectivity and parameter sharing. They are particularly suited for image processing because they exploit the spatially-local correlation of the data by enforcing a local connectivity pattern between neurons of adjacent layers (Goodfellow et al. [2016]). A convolutional layer is composed of a series of convolution kernels which compute the feature maps. For a 2D image the feature map h containing trainable parameters is computed as follow :

$$h^q_{u,v}(x_{u,v}) = \sigma\left(\sum_{i=1}^{k'_i}\sum_{j=i}^{k'_j} \mathbf{W}^q_{i,j}\mathbf{x}_{u+i,v+j}\right) \tag{8}$$

The output from a convolutional layer h consists of N_{out} feature maps. There are two important parameters for the convolutional layer : the padding which pad the border of the images with zeros to preserve the size and the stride which delimits the distance between two successive locations of the filter.

Using convolutional layers in a PINN means that the network can perform an image-to-image regression task by employing an encoder-decoder network structure (see Fig. 4). Such structure consists of encoding and decoding layers (see Goodfellow et al. [2016]) and dense blocks (see Fig 3).

In the dense block (Zhu and Zabaras [2018]), the outputs of each layer are connected with all successor layers. It means that the outputs from the $z(l)$ layer will receive the outputs from all previous layers. Hence the number of feature maps will grow at a rate K which is the number of feature maps in each layer.

The image in the encoder-decoder architecture goes into a serial of coarse-refine process. It means that at the beginning high-level coarse features are extracted from the input using an encoder and then refined to output images through a decoder. When the image/input has reached the high-level coarse feature maps, a latent representation from the image is learnt. Then the decoder will reduce the number of feature maps and increase the size of the image until the output is reached. This type of architecture enhances information propagation through the network and reduces network parameters compared to fully connected neural networks. Thus one should be able to construct accurate surrogate models with limited training data.

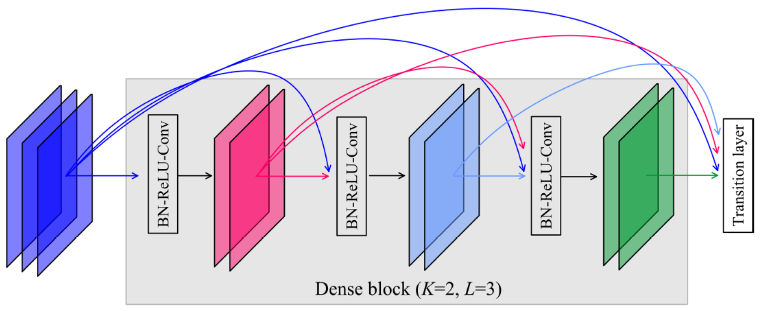

Figure 3: Dense block with $L = 3$ layers and growth rate $K = 2$ - Source : Zhu and Zabaras [2018]

A number of specific challenges appears when the wave equation is solved with PINNs. Indeed it is uncertain if neural networks have enough capabilities to represent the full range of dynamics (transmitted and reflected wave for example). Also as the wave equation is a second

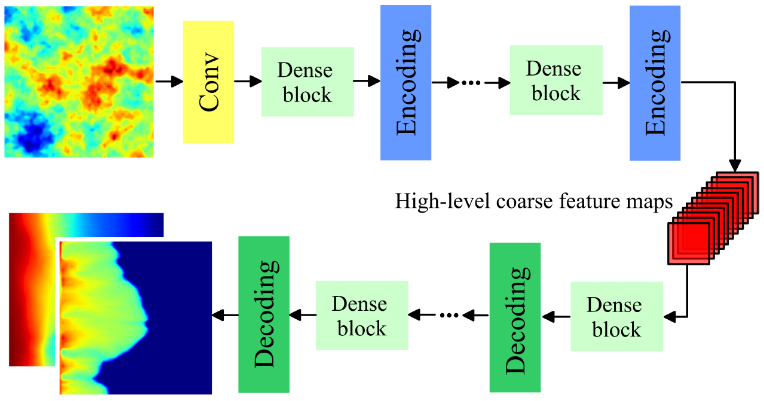

Figure 4: Illustration of the encoder/decoder architecture - Source : Zhu and Zabaras [2018]

order partial differential equation, it requires strict boundary conditions on both the initial wavefield and its derivatives for its solution to be unique. To implement the partial differential equations in the network, one has to ensure that the inputs are space or time variables and that the output is physically meaningful. In Karimpouli and Tahmasebi [2020], the wave equation is solved. Also, due to automatic differentiation in neural networks, partial differentiation is naturally achieved at machine precision. This allows to include differential equations and physics laws in the loss function to penalise the network learning procedure. The loss during the training is expected to decrease and to achieve the best results (lowest loss). The outputs of the network must then meet the PINN loss function.

Consequently, the mean square error of the loss function can be written based on the network output (MSE_u), differential equation (MSE_v) and initial (MSE_0) and boundary (MSE_b) conditions (Karimpouli and Tahmasebi [2020]).

$$MSE = MSE_u + MSE_v + MSE_0 + MSE_b \tag{9}$$

with :

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u}(u - u^*)^2 \tag{10}$$

$$MSE_v = \frac{1}{N_v}\sum_{i=1}^{N_v}(v(x) - L_x^a u(x)^*)^2 \tag{11}$$

$$MSE_0 = \frac{1}{N_0}\sum_{i=1}^{N_u}(u(x_0) - u_0^*)^2 \tag{12}$$

$$MSE_0 = \frac{1}{N_b}\sum_{i=1}^{N_u}(u(x_b) - u_b^*)^2 \tag{13}$$

in the equations, $u(x_0)$, $u(x_b)$, u_0^* and u_b^* are the initial and boundary values of the function and the network respectively.

For the one 1D seismic wave equation (Karimpouli and Tahmasebi [2020]), the only output is $u(x,t)$ and the following loss function is built with the first and second derivatives of the network output with respect to x and t :

$$L = \frac{1}{N_u}\sum_{i=1} N_u(u(x,t) - u^*(x,t))^2 + \frac{1}{N_v}\sum_{i=1}^{N_v}\left(\frac{\partial^2 u^*(x,t)}{\partial t^2} - V^2 \frac{\partial^2 u^*(x,t)}{\partial x^2}\right)^2 \tag{14}$$

where $u^*(x,t)$ is the output of the network. The velocity value is introduced as an unknown parameter to the network. While the network is training, its value will be optimised and the velocity inversion and the displacement prediction will be conducted simultaneously.

4.1 Resources

From the literature, there exist a lot of GitHub repositories available dealing with PINNs for surrogate modelling and parameters inversion. However none of them explicitly covers the wave equation. I am currently in contact with some authors to obtain their code.

4.1.1 Libraries

The list of the few libraries that implement PINN or FEM in Python is presented below :

1. DeepXDE : PINN solver for surrogate modeling (Lu et al. [2020] : `https://github.com/lululxvi/deepxde`)
2. SciANN : PINN solver for surroagte modeling and inverse problem (Haghighat and Juanes [2021] : `https://github.com/sciann/sciann-applications`)
3. Fenics : FEM solver (Alnæs et al. [2015])
4. esys-escript : FEM solver (Schaa et al. [2016])

Regarding the FEM analytical solver, I have been working with Fenics. There exists a lot of examples and tutorials covering Poisson's equation and heat equation. However the documentation for the wave equation is almost non-existing except from few questions on the Fenics Forum. Geneva and Zabaras [2019] use Fenics to implement their FEM results for the Kuramoto-Sivashinsky equation and for the Burger's 1D and 2D equation. The Esys-escript library is a package specifically designed to model the wave equation. It is working on Python and there are a lot of tutorials available even-though I have still not tried this library yet.

For the PINN, I tried SciANN (see the example below in Fig. 5). Physics are entered in the strong form of the equation. One has to specify the activation functions and the number of layers as well as the number of units per layer. Then, the physics are enforced by sampling collocation points on the domain and on the boundary/initial conditions. The loss is minimised with the residual. Below one can find an example where the 1D Burger's equation is solved. The strong form of the problem is the following :

$$u_{,t} + uu_{,x} - \left(\frac{0.01}{\pi}\right)u_{,xx} = 0 \quad t \in [0,1], \quad x \in [-1,1] \tag{15}$$

The mesh size for this problem implemented in SciANN is 200x200 and the time-step for the FEM is $d_t = 0.005[s]$. The initial and the boundary conditions are :

$$u(t=0,x) = -\sin(\pi x), \quad u(t, x = \pm 1) = 0$$

The outputs for this problem are found in Figure 5.

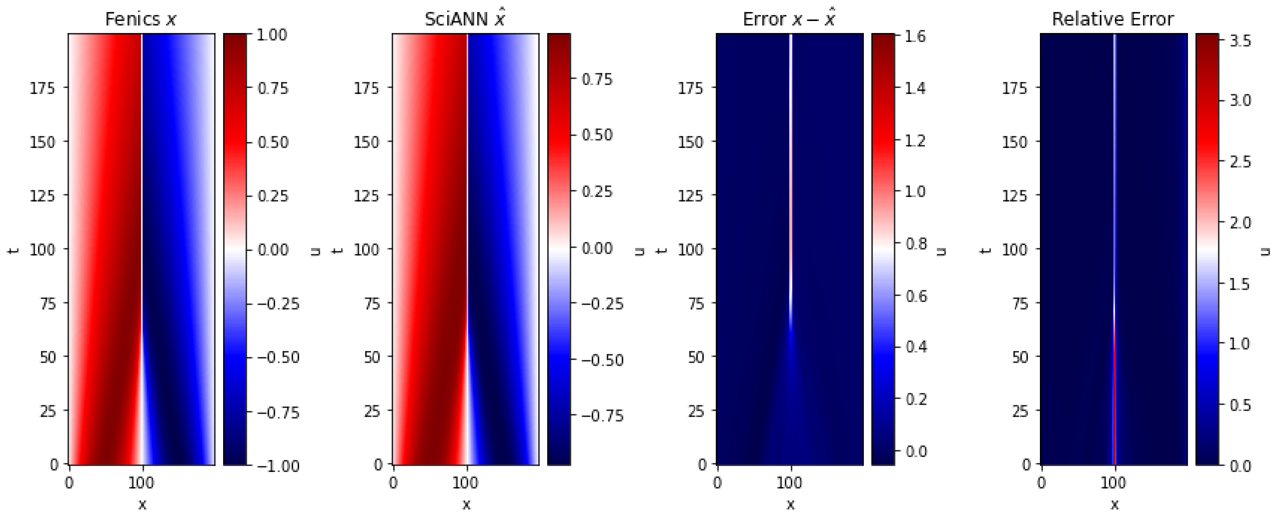

Figure 5: Results from the 1D Burger's equation modelled with FEM and solved with SciANN

Overall one quickly notices the small error and relative error on the whole domain. The PINN can approximate almost perfectly the FEM simulation for this PDE. However there is a region separated with an edge in the middle of the horizontal axis where the network fails to approximate the FEM results (5 first from the left). One has to distinguish between the PDE and the FEM computation as the FEM is an approximation of the real function so the output displayed by the PINN (5 second from the left) could be more accurate than the FEM. To improve the FEM output, the mesh has to be refined around the region with the edge. One has to bear in mind this kind of behaviour while comparing the results during the master project.

4.1.2 Github repositories

1. Sun et al. [2020] : `https://github.com/Jianxun-Wang/LabelFree-DNN-Surrogate`
2. Raissi et al. [2019] : `https://github.com/maziarraissi/PINNs`
3. Zhu and Zabaras [2018] : `https://github.com/cics-nd/cnn-surrogate`
4. Zhu et al. [2019] : `https://github.com/cics-nd/pde-surrogate`
5. Sun and Wang [2020] : `https://github.com/Jianxun-Wang/Physicsconstrained-Bayesian-deep-learning`
6. Geneva and Zabaras [2019] : `https://github.com/cics-nd/ar-pde-cnn`
7. Mo et al. [2019] : `https://github.com/cics-nd/dcedn-gcs`
8. Mo et al. [2020] : `https://github.com/cics-nd/CAAE-DRDCN-inverse`
9. Smith et al. [2020] : `https://github.com/Ulvetanna/EikoNet`

and mine where I will be sharing all the work : `https://github.com/nfholsen/PDM_PINN`

5 Methodology and Objectives

5.1 Methodology

For the moment, I'm not able to generate the desired plots in FEM which will serve later as training data for the PINN and also as true values. However as seen in the previous section one has to be careful with mesh refinement. Once the FEM pipeline is correctly implemented, I will be able to generate 2D acoustic wave propagation for different soil media. The FEM outputs will be stored as images hence I could implement models with either fully connected or convolutional layers. Because the time will be descritised as well, for each time step an image will be saved.

Then using the available libraries (DeepXDE and SciANN), PINNs will be implemented for the wave equation and compared with the results obtained with the FEM instead of the Burgers equation which is currently implemented.

From the literature, Github repositories are available but unfortunately none cover the wave equations. I will implement myself surrogate models based on the same architecture using PyTorch. This will allow me to fully understand all the different steps such as :

- How the models are trained
- How the losses are implemented
- How the boundary and initial conditions are enforced
- The effect of the collocation points sampled on the domain used during the training
- The impact of regularization coefficient in the loss

To visualise the learning rate and to assess that the models are learning some logs could be printed. Choosing to implement the same models as in the literature first with the libraries and then myself and comparing the results would allow me to assess that the model converge to the same results. However the libraries only implements fully convolutional neural network in their models. Hence once the surrogates are built with FCNN, I will try and built models using convolutional layers and the encoder and decoder architecture as presented in Geneva and Zabaras [2019].

In Geneva and Zabaras [2019], the 2D transient Burger's equation is solved with a Encoder-Decoder network and implementing the same model with PyTorch could be a good starting point to assess that my models are working before switching to the wave equation. Once this is done, the results from multiple approaches could be compared. The approaches are the FEM, the data-driven approach (no physics are used), the libraries, the FCNNs and the Encoder-Decoder. Robustness for every model could also be tested. Indeed it could be interested to see what happens for every model if we change the boundary or the initial conditions or if we change the medium (from layered to homogeneous model) or the time-span in which we want predict the solution. The latest could be really interesting because one could train the model with data sampled from $t \in [0, 2)$ and then predict for larger time $t > 2[s]$ and see until when the model predicts reliable solutions.

It is already possible to invert some parameters with the SciANN library (Haghighat and Juanes [2021]). So the first step will be to implement some models with FCNNs from scratch in PyTorch and see to which parameters they converge. The results could be compared with the ones obtained with SciANN. When the models are coded from scratch the loss must change (Gaussian Proccess, Kullback-Leibler Divergence or Stein Variational Gradient Descent) since we want to fit a distribution of parameters. One can also compare the uncertainty propagation and compare the results over many realisations with a Monte-Carlo sampling strategy.

Once the inversion pipeline is implemented for FCNN, the next step would be to switch to the encoder-decoder/convolutional layer models and see how they perform against FEMs and a data driven inversion. To detect the cracks and the different layers, the velocity must not be fixed but space dependent as mentioned in Shukla et al. [2020].

In general I expect the models using convolutional layers to perform better than the FCNNs as the have the property to be equivariance to translation Goodfellow et al. [2016]. It means than if the boundary between to layers changes then the output will be the same wherever the boundary is located in the image.

5.2 List of Objectives

1. Implement the transient acoustic and elastic wave propagation in a FEM solver (Mathematica, Fenics or Esys) for homogeneous, layered and realistic earth model and save the outputs.

2. Use the available libraries (DeepXDE and SciANN) to implement FCNN surrogate models for solving the wave equation and code the same models from scratch using Pytorch.

3. Implement the surrogate models built with convolutional layers models presented in Geneva and Zabaras [2019], Zhu et al. [2019] and Zhu and Zabaras [2018] from scratch using PyTorch

4. Compare the surrogate models outputs between FEM, libraries and PyTorch models and test their robustness by changing the domain (different soil layers) and the time interval (predictions outside their training range)

5. Test the inversion with the SciANN library and implement models from scratch with the same architecture as the ones presented in the papers.

6. Compare the inversion results between the FEM, the FCNNs and convolutional models

7. Test the best performing models on real data instead of synthetic data generated with FEM.

References

M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015. doi: 10.11588/ans.2015.100.20553.

N. Geneva and N. Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks, 2019.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

C. Goyal. *Uncertainty Quantification in Non-linear Seismic Wave Propagation.* PhD thesis, 01 2017.

A. Gulli and S. Pal. *Deep learning with Keras.* Packt Publishing Ltd, 2017.

E. Haghighat and R. Juanes. Sciann: A keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *Computer Methods in Applied Mechanics and Engineering*, 373:113552, Jan 2021. ISSN 0045-7825. doi: 10.1016/j.cma.2020.113552. URL `http://dx.doi.org/10.1016/j.cma.2020.113552`.

S. Karimpouli and P. Tahmasebi. Physics informed machine learning: Seismic wave equation. *Geoscience Frontiers*, 11(6):1993 – 2001, 2020. ISSN 1674-9871. doi: https://doi.org/10.1016/j.gsf.2020.07.007. URL `http://www.sciencedirect.com/science/article/pii/S1674987120301717`.

L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. Deepxde: A deep learning library for solving differential equations, 2020.

S. Mo, Y. Zhu, N. Zabaras, X. Shi, and J. Wu. Deep convolutional encoder-decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media. *Water Resources Research*, 55(1):703–728, 2019. doi: https://doi.org/10.1029/2018WR023528. URL `https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018WR023528`.

S. Mo, N. Zabaras, X. Shi, and J. Wu. Integration of adversarial autoencoders with residual dense convolutional networks for estimation of non-gaussian hydraulic conductivities. *Water Resources Research*, 56(2), Feb 2020. ISSN 1944-7973. doi: 10.1029/2019wr026082. URL `http://dx.doi.org/10.1029/2019WR026082`.

B. Moseley, A. Markham, and T. Nissen-Meyer. Solving the wave equation with physics-informed deep learning, 2020.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686 – 707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL `http://www.sciencedirect.com/science/article/pii/S0021999118307125`.

C. Rao, H. Sun, and Y. Liu. Physics informed deep learning for computational elastodynamics without labeled data, 2020.

Z. E. Ross, D. T. Trugman, E. Hauksson, and P. M. Shearer. Searching for hidden earthquakes in southern california. *Science*, 364(6442):767–771, 2019. ISSN 0036-8075. doi: 10.1126/science.aaw6888. URL `https://science.sciencemag.org/content/364/6442/767`.

R. Schaa, L. Gross, and J. Du Plessis. Pde-based geophysical modelling using finite elements: examples from 3d resistivity and 2d magnetotellurics. *Journal of Geophysics and Engineering*, 13:S59–S73, 2016. doi: doi:10.1088/1742-2132/13/2/S59.

K. Shukla, P. C. D. Leoni, J. Blackshire, D. Sparkman, and G. E. Karniadakis. Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks, 2020.

J. D. Smith, K. Azizzadenesheli, and Z. E. Ross. Eikonet: Solving the eikonal equation with deep neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, pages 1–12, 2020. ISSN 1558-0644. doi: 10.1109/TGRS.2020.3039165.

L. Sun and J.-X. Wang. Physics-constrained bayesian neural network for fluid flow reconstruction with sparse and noisy data. *Theoretical and Applied Mechanics Letters*, 10(3): 161 – 169, 2020. ISSN 2095-0349. doi: https://doi.org/10.1016/j.taml.2020.01.031. URL `http://www.sciencedirect.com/science/article/pii/S2095034920300295`.

L. Sun, H. Gao, S. Pan, and J.-X. Wang. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2019.112732. URL `http://www.sciencedirect.com/science/article/pii/S004578251930622X`.

A. Tartakovsky, C. Marrero, P. Perdikaris, G. Tartakovsky, and D. Barajas-Solano. Physics-informed deep neural networks for learning parameters and constitutive relationships in subsurface flow problems. *Water Resources Research*, 56:e2019WR026731, 04 2020. doi: 10.1029/2019WR026731.

Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415 – 447, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.04.018. URL `http://www.sciencedirect.com/science/article/pii/S0021999118302341`.

Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56 – 81, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2019.05.024. URL `http://www.sciencedirect.com/science/article/pii/S0021999119303559`.