DLSC Project: Part B

ETH zürich

Deep Learning in
Scientific Computing
Due date: July 15th, 2023

Training and testing data can be found on the Moodle page: `https://moodle-app2.let.ethz.ch/course/view.php?id=19789`

IMPORTANT INFORMATION

To get ECTS credits for the course Deep Learning in Scientific Computing you need to submit and obtain a passing grade on the project. Part B is an open project based on reproducing/extending results of current research papers. Please note;

1. You should choose any one topic from the list below and work on it with a team of maximum 4 people.
2. **The final submission consists of PDF report of 2 to a maximum of 4 pages that clearly explains and illustrates the motivation, methods, and results, as well as the code. Save them in the folder *yourfirstname_yoursecondname_yourleginumber.zip*. The report and the code should be the same for all the team members. In the PDF include a little section describing the individual team members contribution.**
3. We do **not** expect you to reproduce all experiments/numerical results in the current research papers referenced. Only the key aspects as described in the problem descriptions below are required to be reproduced/extended.
4. The tasks are designed to be open-ended; you have some flexibility in how you reproduce aspects of the paper and the possible extensions you could carry out.
5. You are **not** allowed to use existing code repositories if they contain code which reproduces the results of the paper. You may use any other existing code repositories/libraries to help you. We will check code submissions to ensure this is the case.

1. **Eigen Values Problem with PINNs.**

   Overview:

   Consider the boundary value problem (BVP):

$$\begin{aligned} u_{xx} + \lambda^2 u = 0,\ x \in [0, L] \\ u(0) = 0 \\ u(L) = 0 \end{aligned} \tag{1}$$

   The general solution of the partial differential equation is

$$u(x) = c_1 \cos(\lambda x) + c_2 \sin(\lambda x) \tag{2}$$

   The boundary conditions give

$$c_1 = 0, \quad \lambda_n = \frac{n\pi}{L} \tag{3}$$

   for any positive integer n. Then,

$$u_n = c_2 \sin\left(\frac{n\pi}{L}\right) \tag{4}$$

   is solution of the BVP (1). u_n and λ_n are the *eigenfunctions* and *eigenvalues* of the BVP (1), respectively.

   Note that u_n solves the BVP $\forall n \in \mathbb{N}$ and $\forall c_2 \in \mathbb{R}$. This means that the solution is not unique. The constant c_2 can be determined by imposing an additional condition on u. It's common to impose

$$||u||_{L^2} = \int_0^L u^2 dx = 1 \tag{5}$$

   Baseline task:

   The objective of the project is to solve the BVP problem, by finding the first 4 solutions solutions (λ_n, u_n), $n = 1, \ldots, 4$, with *physics informed neural networks* (Pinns). In order to do so, the procedure outlined in [4] can be followed.

   Extensions:

   Extension of the project include, for instance, solving the one dimensional *Schrödinger equation* (which the prototypical example of eigenvalue problem) for the single finite well potential (see Section IV, A in [4]) and other forms of the potential.

2. **Radiative Transfer with PINNs**.

   Overview:

   The fundamental equation describing radiative transfer is a *linear partial integro-differential equation*, termed as the *radiative transfer equation*. Under the assumption of a static underlying medium, and in steady conditions, it has the following form,

$$\omega \cdot \nabla_x u + ku + \sigma\left(u - \frac{1}{s_d}\int_S \Phi(\omega, \omega')u(x, \omega')d\omega'\right) = f, \tag{6}$$

with space variable $x \in D \subset \mathbb{R}^d$ (and $D_T = [0,T] \times D$), *angle* $\omega \in S = \mathbb{S}^{d-1}$ i.e. the d-dimensional sphere. The constant s_d in (6) is the surface area of the d-dimensional unit sphere. The unknown of interest in (6) is the so-called *radiative intensity* $u : D_T \times S \mapsto \mathbb{R}$, while $k = k(x) : D \mapsto \mathbb{R}_+$ is the *absorption coefficient* and $\sigma = \sigma(x) : D \mapsto \mathbb{R}_+$ is the *scattering coefficient*. The integral term in (6) involves the so-called *scattering kernel*

$$\Phi : S \times S \mapsto \mathbb{R},$$

which is normalized as

$$\int_S \Phi(\cdot,\ \omega',\ \cdot,\ \nu')d\omega' = 1,$$

in order to account for the conservation of photons during scattering. The dynamics of radiative transfer are driven by a source (emission) term $f = f(x) : D \mapsto \mathbb{R}$.

Since the radiative transfer equation (6) is a *transport equation*, the boundary conditions are imposed on the so-called *inflow boundary* given by,

$$\Gamma_- = \{(t,x,\omega) \in [0,T] \times \partial D \times S : \omega \cdot n(x) < 0\} \tag{7}$$

with $n(x)$ denoting the unit outward normal at any point $x \in \partial D$ (the boundary of the spatial domain D). We specific the following boundary condition,

$$u(t,x,\omega,\nu) = u_b(t,x,\omega,\nu), \quad (t,x,\omega,\nu) \in \Gamma_-, \tag{8}$$

for some boundary datum $u_b : \Gamma_- \mapsto \mathbb{R}$.

The radiative transfer equation is an example of high-dimensional PDE, given that the radiative intensity u is a function of $2d$-variables. Therefore standard numerical methods entails very high computational cost.

Baseline task: The task of this project is to reproduce the results obtained in [5], Section 3.2, where the one-spatial dimensional radiative transfer equation is solved. In this case, the radiative transfer equations (6) simplify to,

$$\mu \frac{\partial}{\partial x} u(x,\mu) + \Big(\sigma(x) + k(x)\Big) u(x,\mu) = \frac{\sigma(x)}{2} \int_{-1}^{1} \Phi(\mu,\mu') u(x,\mu') d\mu', \quad (x,\mu) \in [0,1] \times [-1,1], \tag{9}$$

with

$$\sigma(x) = x, \quad k(x) = 0, \quad \Phi(\mu',\mu) = \sum_{\ell=0}^{L} d_\ell P_\ell(\mu) P_\ell(\mu'), \quad d_0 = 1, \tag{10}$$

and $P_\ell(\mu)$ denoting the Legendre polynomial of order ℓ. We employ the sequence of coefficients $d_\ell = \{1.0, 1.98398, 1.50823, 0.70075, 0.23489, 0.05133, 0.00760, 0.00048\}$.

Extensions: Possible extensions of the project include solving the equation for additional experimental setups reported in [5].

3. **Neural Operators and Operator Networks vs Parametric Approach.**

   Overview:

In this project, we follow tutorial 5 and consider the two-dimensional parametric heat equation:

$$u_t = \Delta u, \quad t \in [0, T], \ (x_1, x_2) \in [-1, 1]^2, \ \mu \in [-1, 1]^d$$

with the initial condition

$$u(0, x_1, x_2, \mu) = u_0(x_1, x_2, \mu) = \frac{1}{d} \sum_{m=1}^{d} \sum_{m=1}^{d} u_0^m(x_1, x_2, \mu_m) = -\frac{1}{d} \sum_{m=1}^{d} \mu_m \sin(\pi m x_1) \sin(\pi m x_2)/m^{0.5},$$

$$u_0^m(x_1, x_2, \mu_m) = -\mu_m \sin(\pi m x_1) \sin(\pi m x_2)/m^{0.5}$$

and zero Dirichlet BC.

Assume that $\mu \sim Unif([-1, 1]^d)$. Every realization $\mu^{(j)}$ of the input parameter uniquely identifies the initial condition $u_0^{(j)}$, which in turn uniquely define the solution of the PDE $u^{(j)}$.

Consider

(a) the function g, $(x_1, x_2, \mu) \mapsto u(T, x_1, x_2, \mu)$,

$$g(x_1, x_2, \mu) = u(T, x_1, x_2, \mu)$$

(b) the operator $\mathcal{G} : u_0(\cdot, \cdot) \mapsto u(T, \cdot, \cdot)$,

$$\mathcal{G}(u_0)(x_1, x_2) = u(T, x_1, x_2)$$

Given the remark and the definition above, we have

$$g(x_1, x_2, \mu^{(j)}) \equiv \mathcal{G}(u_0^{(j)})(x_1, x_2) \tag{11}$$

Therefore, two potential surrogate models can be learned to approximate the underlying solution of the partial differential equation:

(a) a standard feed forward neural network to approximate the function g (*parametric approach*)

(b) an operator network (DeepONet) or Neural Operator to approximate the operator $\mathcal{G}$ (*operator approach*)

Baseline task: In this project we would like to compare the parametric approach and the operator approach (DeepONet, Fourier Neural Operator) as approximations of the problem above.

Extensions:

Possible extensions of the project include:

(a) Consider additional experimental setups, for which an exact solution for the PDE of interest is computable (for instance Wave Equation). If any analytical solution is available, manufactured solution can be also used .

(b) Perform the comparison for different number d of the parameter space.

(c) Include *Convolutional Neural Operator* in the comparison.

4. **Finite Basis PINNs.**

Overview:

In this project, you will reproduce aspects of the paper Finite basis physics-informed neural networks (FBPINNs) [6].

Whilst physics-informed neural networks (FBPINNs) provide a powerful approach for solving problems related to differential equations, their performance (accuracy and computational efficiency) often declines rapidly when solving problems with high-frequency and multi-scale solutions. That is, PINNs struggle to *scale* to more complex problems.

This is typically due to the *spectral bias* of neural networks, which is the well-studied property that neural networks tend to learn higher frequencies much slower than lower frequencies. Compounding this, as the solution complexity grows, deeper neural networks, larger numbers of free parameters and more collocation points are required to model the solution, which leads to an increasingly complex PINN optimisation problem.

FBPINNs try to improve the scalability of PINNs to higher-frequency and multi-scale problems by combining PINNs with *domain decomposition*. Instead of using a single neural network to model the global solution, FBPINNs place many small neural networks in overlapping subdomains, and represent the solution as a summation over all subdomain networks. By taking this "divide-and-conquer" approach, the hope is that the global PINN optimisation problem is decomposed into many smaller, easier-to-solve, local coupled optimisation problems.

Baseline task:

Your task is to reproduce a key element of this paper. In particular, you should first reproduce Figure 7 from this paper. That is, training a FBPINN to solve the multi-scale ODE boundary value problem

$$\begin{aligned} \frac{du}{dx} &= \sum_{i=1}^{n} \omega_i \cos(\omega_i x) \\ u(0) &= 0, \end{aligned} \tag{12}$$

where for the paper, $n = 2$ and $\omega_1 = 1$, $\omega_2 = 15$. Similar to Figure 7, the performance of the FBPINN (in terms of solution accuracy and computational cost of training) should be compared to the performance of a PINN.

Extensions:

After reproducing Figure 7, there are many possible extensions you could carry out. Some ideas could be:

(a) Test the scalability of FBPINNs and PINNs for the problem above (eq. (12)) by increasing the value of n and adding more multi-scale components to the solution. For example, one could set $n = 5$, $\omega_i = 2^i$ and increase the number of subdomains in the FBPINN (or number of layers in the PINN) appropriately. How does the accuracy and computational cost of the FBPINN change as n is increased?

(b) Reproduce any of the other experiments in the paper, for example the 2D sinusoidal problem (section 5.3) or 2D Burgers' equation (section 5.4).

(c) Extend FBPINNs however you think interesting. For example, testing the effect of different neural network architectures, using FBPINNs to solve an inverse problem, or using irregular domain decompositions instead of uniform rectangular decompositions.

5. **Neural ordinary differential equations (Neural ODEs).**

Overview:

In this project, you will reproduce aspects of the paper Neural ordinary differential equations (Neural ODEs) [1].

Neural ODEs provide a fascinating link between differential equations and the design of neural network architectures. In particular, they show that traditional differential equation solvers can be used to define certain neural network architectures.

For example, consider the ODE

$$\frac{d\mathbf{h}(t)}{dt} = \mathbf{f}(\mathbf{h}(t), \boldsymbol{\theta}(t)).$$

A simple way of solving this ODE is to discretise the solution in time and take Euler steps, i.e.

$$\mathbf{h}_{t+1} = \mathbf{h}_t + \Delta t \cdot \mathbf{f}(\mathbf{h}_t, \boldsymbol{\theta}_t), \tag{13}$$

where Δt is some small timestep. But note, eq. (13) also exactly defines the layer function for a single layer in a residual network (ResNet) [2], where $\mathbf{h}_t$ is the input to the layer, $\mathbf{h}_{t+1}$ is the output of the layer and $\mathbf{r}(\mathbf{h}_t, \boldsymbol{\theta}_t) = \Delta t \cdot \mathbf{f}(\mathbf{h}_t, \boldsymbol{\theta}_t)$ is the learnable residual component of the layer. Thus, from a numerical point of view, ResNets can be thought of as simple Euler ODE solvers.

This opens up the possibility of using insights from traditional numerical methods to inform the design and training of neural networks. For example, one extension is to use a higher-order method (e.g. higher-order Runge–Kutta methods) to solve the ODE, which defines a different type of neural network architecture.

Baseline task:

Your task is to reproduce a key element of this paper. In particular, you first should reproduce Table 1 from this paper. That is, training a ResNet to classify digits based on the MNIST dataset, and comparing its performance to a Neural ODE trained with a Runge-Kutta solver.

Extensions:

After reproducing Table 1, there are many possible extensions you could carry out. Some ideas could be:

(a) Test the performance of the Neural ODE (in terms of accuracy and computational cost of training) when using other ODE solvers, for example higher-order Runge-Kutta methods, or the implicit solvers used in the paper.

(b) Run similar comparisons of Neural ODEs vs ResNets on harder ML tasks, for example image segmentation, image denoising or harder classification tasks.

(c) Reproduce other aspects of the paper, for example training a continuous normalizing flow to model probability distributions using Neural ODEs (Section 4 of the paper).

6. **Enforcing temporal causality in PINN training.**

Overview:

The purpose of this project is to emphasize the importance of *causality* across the time dimension for certain PDEs. It is easy to show (and also qualitatively obvious), that a PINN solution $u_\theta(\cdot, t)$ will stray further and further away from the true solution for $t' > t$, if the error at time t is already uncontrolled.

As a way of mitigating potential violations of causality, the authors of [7] propose a new loss formulation for the residual error which reads

$$\mathcal{L}(\theta) = \frac{1}{N_t} \sum_{i=1}^{N_t} w_i \mathcal{L}(t_i, \theta) \tag{14}$$

where $\mathcal{L}(t_i, \theta)$ denotes the residual error at time t_i and w_i denotes the weight assigned to the i'th residual. The trick resides in defining w_i in such a way that causality is enforced, i.e.

$$w_i := \exp\Big(-\epsilon \cdot \sum_{j=1}^{i-1} \mathcal{L}(t_j, \theta) \Big) \tag{15}$$

As can be seen, large errors accumulating at times $j < i$ for some given i will force the loss defined at 15 to not be minimized, hence ensuring that we only start optimizing for further timesteps $k \leq i$ iff causality is indeed respected.

Baseline task:

Based on the work done in [7], the task at hand is to train PINNs for several time-dependent equations (namely *Allen-Cahn*, *Burger's* equation and the *KdV* equation) both in the usual fashion (as a baseline) and with the causal loss term.

Report as in Figure 1 and Figure 3 in [7] the results you get from the two modes of training (regular PINN training vs causal), for the aforementioned equations (use the Allen-Cahn initial / boundary data from the paper, but for the other two equations you are free to choose your own settings).

You are **not** expected to fully reproduce the numbers in the paper, nor the full range of experiments outside of the equations mentioned in the description of the task.

Extensions:

Aim to reproduce the results for the *Kuramoto–Sivashinsky* equation (regular, not chaotic version) as in the experimental setup from the paper.

7. **Meta-Learning PINNs for parametric PDEs.**

Overview:

One is often faced with PDE systems where small variations in the boundary / initial conditions have to be taken into account, ultimately resulting in having to solve the system repeatedly for a finite number of perturbations and consider an ensemble of solutions as a final solution.

This is rather inconvenient, and fully disregards the common structure of the perturbed systems. One way to approach this problem, as presented in *Meta-Auto-Decoder for Solving Parametric Partial Differential Equations* [3] is to employ meta-learning for learning to solve PDE systems while accounting for different small variations in the initial / boundary data.

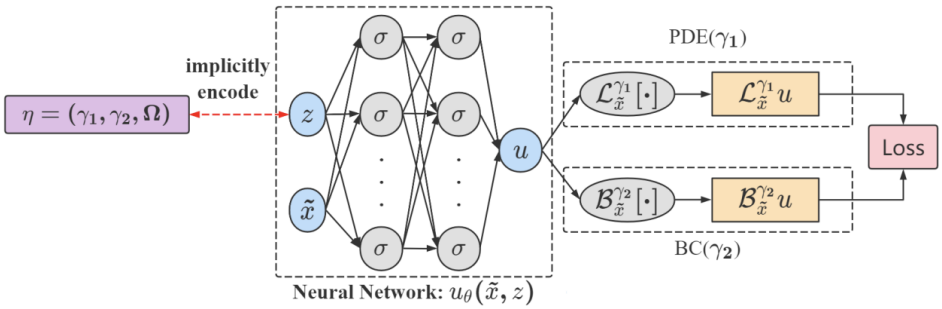

Figure 1: Meta-learning pipeline

The authors proposed encoding the different boundary / initial data into an extra input vector η, which then gets a latent code z, that makes up the augmented ansatz $u_\theta(x, z)$ which now takes into account the variations in the initial / boundary data, see Fig. 1.

For example, given the Laplace equation $\Delta u = 0 \in \Omega, u|_{\partial\Omega} = g$ the author use $\eta := (\hat{\Omega}, \hat{g})$ i.e. a tuple containing a discretization of the domain, along with a discretized version of the boundary data g. This allows one to account for different shapes of Ω and different perturbations of the boundary data g both at traing and inference time.

The varying domains Ω are simple polygons with small variations in their vertices, while the functions g can be simply drawn directly from a Gaussian process with a reasonably nice covariance function.

Baseline task:

Aim to reproduce the results presented in the table from Fig. 4 (plots (a), and (c)) in [3] but only for the methods MAD, MAD-L, FROM-SCRATCH and only for *Burger's* equation and the *Laplace* equation, using the experimental specifications presented in sections 3.1 and 3.3 of the paper.

However, you **may** use different encoding strategies for η and do not need to conform with using the architectures presented in the paper.

You are **not** expected to fully reproduce the numbers in the paper, nor the full range of experiments outside of the equations mentioned in the description of the task.

Extensions:

Implement other baseline algorithms from the paper i.e. MAML, or add *Maxwell's* equations (with the setup from the paper) to your set of solved equations.

References

[1] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural Ordinary Differential Equations. *NIPS*, 109(NeurIPS):31–60, jun 2018.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778. IEEE Computer Society, dec 2016.

[3] Xiang Huang, Zhanhong Ye, Hongsheng Liu, Shi Ji, Zidong Wang, Kang Yang, Yang Li, Min Wang, Haotian Chu, Fan Yu, et al. Meta-auto-decoder for solving parametric partial differential equations. *Advances in Neural Information Processing Systems*, 35:23426–23438, 2022.

[4] Henry Jin, Marios Mattheakis, and Pavlos Protopapas. Physics-informed neural networks for quantum eigenvalue problems. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2022.

[5] Siddhartha Mishra and Roberto Molinaro. Physics informed neural networks for simulating radiative transfer. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 270:107705, 2021.

[6] Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *arXiv*, jul 2021.

[7] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404*, 2022.