Machine Learning per il Calcolo Scientifico

Fourth laboratory exercises

General guide

For each laboratory, an incomplete Python notebook will be provided with exercises (steps) that must be completed in the given order (some of the exercises will be needed in future laboratories). In step zero, all the Python packages that are needed in order to complete the notebook are listed. This PDF includes the text for the exercises and the expected outcomes for each step. While following the notebook is recommended, you are also welcome to attempt the exercises without using it.

Step Zero

Here are the required Python (`https://www.python.org/`) packages for this laboratory:

- PyTorch (`https://pytorch.org/`)
- Numpy (`https://numpy.org/`)
- Matplotlib (`https://matplotlib.org/`)
- Scipy (`https://scipy.org/install/#pip-install`)

Step one: Create the dataset

In this laboratory we consider the one-dimensional Poisson equation

$$\begin{cases} -\frac{d^2u}{dx^2}(x) = f(x) & x \in [0, L], \\ u(0) = u(L) = 0, \end{cases}$$

with random r.h.s. $f(x) = a_0 \sin(0.1\pi x) + a_1 \cos(0.5\pi x) + a_2 \sin(\pi x)$.

a. Generate a dataset of 250 examples with 100 collocation points per sample using the function 'generate_diffusion_data' provided in the utilities.py file. The dataset is composed of the collocation points 'x' the input data (in this case the r.h.s. $f(x)$) and the solution 'u' of the Poisson equation.

b. Plot the dimensions of each tensor to ensure that the dataset has been created correctly and take a look plotting some input-output pairs using the 'plot_data' function provided in the utilities.py file.

Step two: Define the DeepONet class

a. Using DeepNet (DNN.py) define the class DeepONet (at the moment ignore the input called 'bc').

b. Define a DeepONet with tanh as activation function and with trunk and branch composed of 2 hidden layer of 50 neurons each and an output layer of 120 neurons, the input layer have to be chosen properly to respect the input sizes. Be careful that trunk network needs the activation function even in the last layer while the branch network does not.

c. Print the model and check that corresponds to the figure provided.

Step three: Train the DeepONet

a. Define a function to train the DeepONet using the Adam optimizer with a learning rate of 10^{-4} for 2500 epochs and halving the learning rate every 2500 epochs and use the Mean Squared Error (MSE) as loss function. During the training process save the value of the loss function.

b. Make a figure of the loss function and check that it decreases.

c. Make a function to evaluate the model on the dataset (without training) and plot the results. You can use the function 'plot_results' provided in the utilities.py file.

d. You can run the training process multiple times (without restarting the notebook) to augment the number of epochs of training and check both about the convergence of the loss function and the resulting approximation plots.

Step four: Impose hard boundary conditions

a. Modify the DeepONet class to include the include the possibility to impose hard Dirichlet boundary conditions at the solution.

b. Train the model with the new implementation of the boundary conditions and check the new results obtained.

c. Try to change the loss function with the relative L^2 loss function and check the results using 'LprelLoss' provided in utilities.py.