DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

WI4635 LINEAR ALGEBRA AND OPTIMIZATION FOR MACHINE LEARNING – PROJECT 2

Instructions

- The ideal solution (grade 10) includes both the implementation as well as a discussion of the training, validation, and test results. The implementation has to be commented/documented as well as algorithmic choices explained and justified.
- The implementation should only use standard packages, such as the Python packages NumPy and SciPy, but no deep learning libraries that directly provide an implementation of any of the methods/models described below.
- Suboptimal performance or unexpected behavior of the model/implementation should be discussed.
- A submission should contain a PDF file of the report as well as a zipped folder containing the code, data, as well as instructions on how to execute the code. The code may be provided in form of a Python script or a Jupyter notebook. If during the grading, it is not clear how to run the code, we may assume that the code does not work.
- **The deadline is December 20, 2024, 23:59 CET. Please submit one project per group through Brightspace.**

Motivation

Consider the boundary value problem

$$\begin{aligned} -\Delta u(x,y) &= 2a\pi^2 \sin(\pi x)\sin(\pi y) && \text{in } \Omega, \\ u(x,y) &= b && \text{on } \partial\Omega, \end{aligned} \tag{Poisson 2D}$$

for $\Omega = [0,1]^2$. The analytical solution is given by

$$u(x,y) = a\sin(\pi x)\sin(\pi y) + b;$$

as an example the solution for $a = 1$ and $b = 0$, $u(x,y) = \sin(\pi x)\sin(\pi y)$, is shown in Figure 1 (left).

In this project, you will approximate $u(x,y)$ using neural networks and compare the approximation properties of neural networks with the performance of a numerical finite difference solver on a uniform grid; cf. Figure 1 (right). The problem (Poisson 2D) can be solved with a central finite difference scheme using the class `PoissonSolver2D` as shown in the following code.

```
# Define the boundary condition function
def boundary_func(x, y):
        return 0.0

# Define the source term function
def source_func(x, y):
        return 2.0 * np.pi**2 * np.sin(np.pi * x) * np.sin(np.pi * y)

# Create an instance of the PoissonSolver2D class
solver = PoissonSolver2D(nx=100, ny=100, lx=1.0, ly=1.0)
```

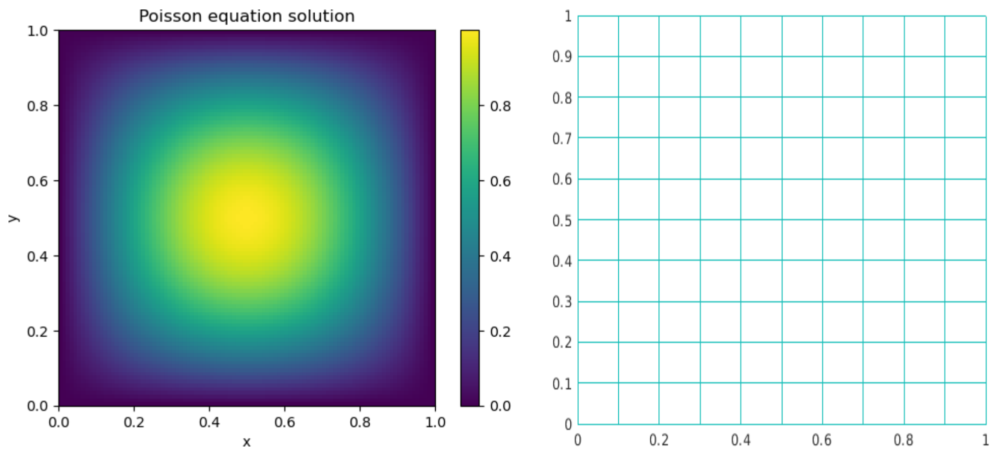

Figure 1: **Left:** Solution of eq. (Poisson 2D) with $a = 1$ and $a_2 = b = c = d = e = b_5 = 0$, that is, $u(x, y) = \sin(\pi x)\sin(\pi y)$. **Right:** Uniform 10×10 grid.

```
# Set the boundary conditions
solver.set_boundary_conditions(boundary_func)

# Set the source term
solver.set_source_term(source_func)

# Setup the linear system
solver.setup_linear_system()

# Solve the Poisson equation
solver.solve()
```

In this example, the solution is approximated on a 100×100 grid. This code and the code for the class `PoissonSolver2D` can also be found in the Jupyter notebook `fdm.ipynb` in the assignment on Brightspace. You do not have to study the details of this method or the implementation.

Part 1 – Feedforward Neural Network (FNN)

In the first part of the project, implement a feedforward neural network model

$$\begin{aligned} \mathrm{NN} &: \mathbb{R}^d \to \mathbb{R}^p, \\ \mathrm{NN}(x,y) &= \mathbf{A} \cdot F_L \circ \ldots \circ F_1(x,y) \end{aligned} \tag{FNN}$$

with

$$\begin{aligned} F_i &: \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}, \\ \mathbf{x}_i &= \sigma\left(\mathbf{W}_i \mathbf{x}_{i-1} + \mathbf{b}_i\right). \end{aligned}$$

for $i = 1, \ldots, L$. Here, $\mathbf{W}_i \in \mathbb{R}^{n_i \times n_{i-1}}$ ($n_0 := d$), $\mathbf{b}_i \in \mathbb{R}^{n_i}$, $\mathbf{A} \in \mathbb{R}^{p \times n_L}$, and $\sigma(x)$ is the activation function, which is applied element-wise.

(a) Implement an FNN for approximating the solutions of problem (Poisson 2D) with **(3 pt.)**

$$\begin{aligned} -\Delta u(x,y) &= 2\pi^2 \sin(\pi x)\sin(\pi y) && \text{in } \Omega, \\ u(x,y) &= 0 && \text{on } \partial\Omega, \end{aligned}$$

for $\Omega = [0,1]^2$ using data. You can generate the training data yourself by choosing points (x_i, y_i) in Ω as input and evaluating the analytical solution $u(x_i, y_i)$ in these points as output. You may decide how to choose the points yourself $u(x_i, y_i)$. It is advised to normalize the data, for instance, by transforming them to the interval $[0,1]$ (min-max scaling) before training the neural network to fit the data.

For your implementation implement the missing functions in the template for the class `FeedforwardNeuralNetwork` to be found in the Jupyter notebook `fnn.ipynb` in the assignment on Brightspace. Most importantly, this includes

- the activation function,
- the loss function,
- the forward propagation,
- the backward propagation,
- the gradient descent algorithm.

As the loss function and performance measure employ the mean squared error (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n} \|\mathrm{NN}(x_i, y_i) - u(x_i, y_i)\|^2, \tag{MSE}$$

where u is the analytical solution of (Poisson 2D).

You may employ any techniques and variants of algorithms discussed in the lecture to improve performance, given that you implement them yourself. Justify your algorithmic choices.

(b) Perform a grid search with cross validation to optimize hyper parameters (e.g., neural network architecture, activation function, and learning rate). Therefore, define a reasonable search space yourself. **(1.5 pt.)**

(c) Compare the accuracy of approximating the solution u using your feedforward neural network `FeedforwardNeuralNetwork` for varying numbers of training data points N against the accuracy of the finite element solver `PoissonSolver2D`. **(1 pt.)**

As test points to compare against the finite difference solver `PoissonSolver2D` use the grid points of `PoissonSolver2D`, which can be obtained using the function `get_meshgrid()`. You can change the refinement of the grid in x and y direction via the parameter `nx` and `ny`, respectively.

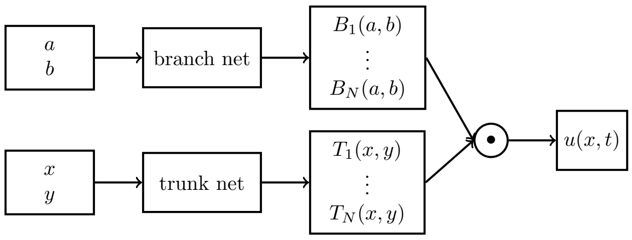

Figure 2: Deep operator network architecture inspired by [3].

Part 2 – Deep Operator Network Architecture

Next, we extend the neural network to approximate "all" solutions of eq. (Poisson 2D), employ the deep operator network (DeepONet) architecture from [3]. In particular, we train the neural network for different values of the parameters (a, b) at the same time. The DeepONet architecture is given by

$$\mathrm{DeepONet}(a, b, x, y) = \sum_{i=1}^{N} B_i(a, b)\, T_i(x, y), \tag{DeepONet}$$

as shown in Figure 2; here N is a hyper parameter, which is related to the complexity of the model. The *"branch net"* $b(a, b)$ and *"trunk net"* $t(x, y)$ are a FNNs with the following input and output dimensions

$$B(a,b) = \begin{bmatrix} B_1(a,b) \\ \vdots \\ B_N(a,b) \end{bmatrix}, \quad T(x,y) = \begin{bmatrix} T_1(x,y) \\ \vdots \\ T_N(x,y) \end{bmatrix},$$

of the form eq. (FNN). Simply speaking, this architecture splits the input into the parameters (a, b) and the spatial coordinates (x, y). It employs a separate FNN on each part and then combines them via a dot product.

A template for the implementation can be found in the notebook `deeponet.ipynb`.

Note: *If you are interested in more details on this architecture, please see the original paper [3]. The DeepONet is an example of a neural operator; see also [2, 1]. However, it is not necessary to read those references (in detail) to complete the project assignment.*

(a) Implement the DeepONet architecture using your implementation `FeedforwardNeuralNetwork` for the branch and trunk networks. Some preliminary testing, fix a and b and test whether you can obtain similar results as in Part 1. **(2.5 pt.)**

(b) Now, test this approach on (Poisson 2D) with fixed b, that is, for the boundary value problem **(1 pt.)**

$$\begin{aligned} -\Delta u(x,y) &= 2a\pi^2 \sin(\pi x)\sin(\pi y) && \text{in } \Omega, \\ u(x,y) &= 0 && \text{on } \partial\Omega, \end{aligned} \tag{Poisson 2D}$$

for $\Omega = [0,1]^2$ and with $a \in [-1, 1]$. The training data can be generated as follows: As input data, choose tuples of the form (a_i, x_i, y_i). The output data can be obtained evaluating the analytical solution $u(x_i, y_i)$ for these values of a_i, $b_i = 0$, and point (x_i, y_i).

Perform a hyper parameter optimization with cross-validation to find a good set of hyper parameters (including the network architecture).

Then, test your model for "unseen" values of a (values not used during training).

(c) Finally, repeat the procedure from the previous subquestion when varying both parameters a and b in the interval $[-1, 1]$. **(1 pt.)**

References

[1] Nicolas Boullé and Alex Townsend. Chapter 3 - A mathematical guide to operator learning. In Siddhartha Mishra and Alex Townsend, editors, *Handbook of Numerical Analysis*, volume 25 of *Numerical Analysis Meets Machine Learning*, pages 83–125. Elsevier, January 2024.

[2] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural Operator: Learning Maps Between Function Spaces, October 2022. arXiv:2108.08481 [cs, math].

[3] Lu Lu, Pengzhan Jin, and George Em Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. arXiv:1910.03193 [cs, stat].