DLSC

Matthias Sanicanin

15.06.2023

Task 2: PDE-Constrained Inverse Problem

In task 2 we wanted to infer the solid temperature from the fluid temperature, by training two neural nets: One to approximate the fluid and another to use this approximation to get solution for the solid, by enforcing the PDE that governs the fluid and solid temperatures,

$$\frac{\partial T_f}{\partial t} + U_f \frac{\partial T_f}{\partial x} = \alpha_f \frac{\partial^2 T_f}{\partial x^2} - h_f(T_f - T_s).$$

The template from tutorial 5 contained most of the basic structure needed to complete this task, so I will only go over the points where my solution differs from the template.

The loss function has two key components, the function loss and the PDE loss.
The function loss is comprised of the loss of the fluid approximation at the temporal boundary, the spatial boundary and at the measurement points (from Datasolution.txt). We use it to train the first NN to approximate the fluid temperature accurately. The PDE loss is the loss arising from the PDE given for the problem, it allows us to infer the solid temperature from the fluid temperature.
The **compute_temporal_boundary_residual** function, in my solution, simply enforces the initial condition on the fluid temperature

$$T_f(x, t = 0) = T_0, \quad x \in [0, 1].$$

The **compute_spatial_boundary_residual** function is used to compute the boundary residuals arising from,

$$\begin{aligned}
\textbf{Charging:} \quad & T_f(0,t) = T_{hot}, \quad \frac{\partial T_f}{\partial x}|_{x=1} = 0, \quad t \in [0, 8] \\
\textbf{Discharging:} \quad & \frac{\partial T_f}{\partial x}|_{x=0} = 0, \quad T_f(1,t) = T_{cold}, \quad t \in [0, 8] \\
\textbf{Idle:} \quad & \frac{\partial T_f}{\partial x}|_{x=0} = 0, \quad \frac{\partial T_f}{\partial x}|_{x=1} = 0, \quad t \in [0, 8]
\end{aligned}$$

which can be done by simply taking the the values of $\frac{\partial T_f}{\partial x}$ at all the spatial boundary points and overwriting them at at the proper charging and discharging points with the difference $T_{hot} - T_{pred}$, $T_{cold} - T_{pred}$, respectively.
The **compute_spatial_boundary_residual** function simply calculates the deviation of our two approximate solutions from the PDE governing the fluid and solid temperatures. My code, again, relies on the use of masks to account for the fact that U_f changes between phases and therefore in the residual calculation of our interior points.

With the core architecture in place, I spent a considerable amount of time tuning hyperparameters of the networks. As in task 1, once I found a sufficient capacity for the networks I tried optimising the weighting of the function and PDE losses. In short, I found that weighting the function loss heavily allowed the networks to focus, in a first step, on approximating the fluid accurately. Once that was done, i.e. the function loss was very small, the training started to infer the solid temperature by minimising the PDE loss, once it was roughly on par with the heavily weighted function loss. I ended up having to do multiple runs of LBFGS, always reloading the previous weights from **approximate_solution_fluid.state_dict()** and changing the weighting in between runs to get a satisfactory result.

It is worth noting that a considerable amount of time was spent pondering large measurement and temporal boundary losses, this was due, in part, to faulty values in the DataSolution text file. The very first point for instance, contradicts the initial condition, making it impossible to reconcile the behaviour at the temporal boundary with that of the measurement values.