Project 1: Thermal Storage Design with PINNs

A. Umeri

April 12, 2024

Abstract

The main objective of the project is to apply *Physics Informed Neural Networks* (PINNs) to solve tasks related to the preliminary design of a *thermal energy storage*.

Task 1: PINNs for solving PDEs

We consider the following system of *reaction-convection-diffusion* equations:

$$\begin{aligned} \frac{\partial \bar{T}_f}{\partial t} + U_f \frac{\partial \bar{T}_f}{\partial x} &= \alpha_f \frac{\partial^2 \bar{T}_f}{\partial x^2} - h_f \left(\bar{T}_f - \bar{T}_s\right) \quad x \in [0,1],\ t \in [0,1] \\ \frac{\partial \bar{T}_s}{\partial t} &= \alpha_s \frac{\partial^2 \bar{T}_s}{\partial x^2} + h_s \left(\bar{T}_f - \bar{T}_s\right) \quad x \in [0,1],\ t \in [0,1] \end{aligned} \tag{7}$$

with the following initial and boundary conditions:

$$\begin{aligned} &\bar{T}_f(x, t=0) = \bar{T}_s(x, t=0) = T_0, && x \in [0,1] \\ &\left.\frac{\partial \bar{T}_s}{\partial x}\right|_{x=0} = \left.\frac{\partial \bar{T}_s}{\partial x}\right|_{x=1} = \left.\frac{\partial \bar{T}_f}{\partial x}\right|_{x=1} = 0, && t \in [0,1] \\ &\bar{T}_f(x=0, t) = \frac{T_{hot} - T_0}{1 + \exp(-200(t - 0.25))} + T_0, && t \in [0,1] \end{aligned} \tag{8}$$

and with the following constants:

$$\begin{aligned} &\alpha_f = 0.05, \quad h_f = 5, \quad T_{hot} = 4, \quad U_f = 1, \\ &\alpha_s = 0.08, \quad h_s =, 6 \quad\ \ T_0 = 1. \end{aligned} \tag{9}$$

We approximate the solution of the system of PDEs with a *physics informed neural network*. To this end we used a two-outputs neural network

$$NN_\theta \colon (t, x) \mapsto \left(\bar{T}_f^\theta, \bar{T}_s^\theta\right)$$

with trainable parameters θ. For performance reasons we did not consider to implement two separate neural networks. We use the *PINNs tutorial* as foundation for our implementation. The following procedures have been suitably adapted to perform this task:

1. *initial_condition*
2. *add_temporal_boundary_points*
3. *add_spatial_boundary_points*
4. *add_interior_points*
5. *apply_boundary_conditions*
6. *compute_pde_residual*
7. *compute_loss*

The loss function takes into account the squared approximation error (L_2-norm of the residuals) at the temporal and spatial boundaries and interior points.

Here are the approximations $(\bar{T}_f^\theta, \bar{T}_s^\theta)$ given Sobol points as input:

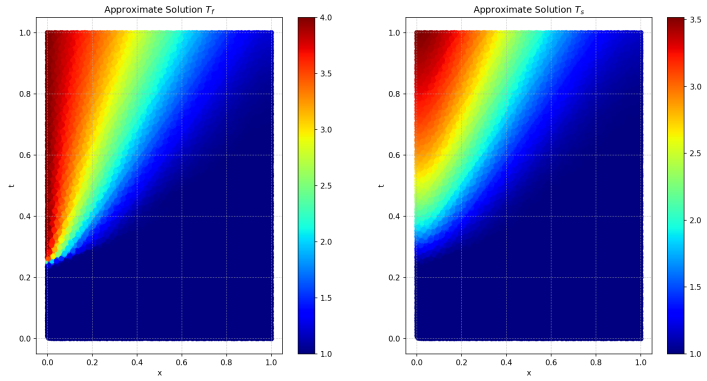

Figure 1: PINN-approximations $(\bar{T}_f^\theta, \bar{T}_s^\theta)$ given Sobol points as input data.

Predictions have been made based on test data as well:

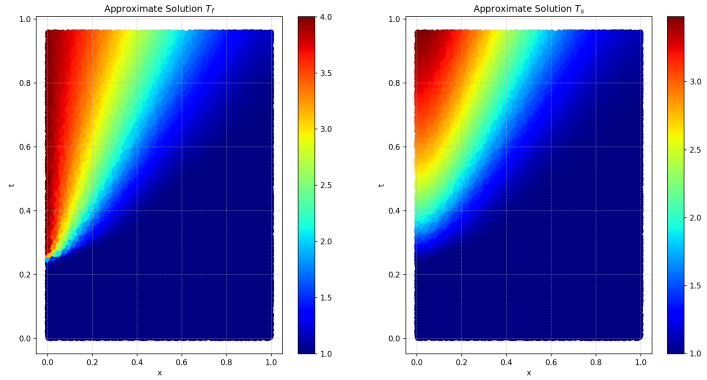

Figure 2: Predictions based on the trained PINN and test data as input.

Task 2: PDE-Constrained Inverse Problem

Consider now the equation governing the fluid temperature:

$$\frac{\partial \bar{T}_f}{\partial t}(x,t) + U_f(t)\frac{\partial \bar{T}_f}{\partial x}(x,t) = \alpha_f \frac{\partial^2 \bar{T}_f}{\partial x^2} - h_f\left(\bar{T}_f(x,t) - \bar{T}_s(x,t)\right) \quad x \in [0,1], t \in [0,8].$$

The goal of the task is to infer the values for the solid temperature $\bar{T}_s = \bar{T}_s(x,t)$, given input data of exact values of the fluid temperature $\bar{T}_f$ in the space-time domain using inverse algorithms. The task has been solved using a PINN-algorithm using 2 neural networks $\bar{T}_f^\theta$, $\bar{T}_s^\theta$ trained based on Sobol-points in the space-time domain. We used the *PINNs for Inverse Problems* as template for our implementation. The following procedures in the template have been modified:

1. *add_temporal_boundary_points*
2. *add_spatial_boundary_points*
3. *add_interior_points*
4. *get_measurement_data*
5. *assemble_datasets*
6. *apply_boundary_conditions*
7. *compute_pde_residual*
8. *compute_loss*

In this inverse problem the time horizon consists of 2 cycles of lenght $T_c = 4$. Each cycle consists of a *charging phase*, an *idle phase*, a *discharging phase* and again an *idle phase* each of lenght 1. Depending on the phase, the velocity of the fluid U_f takes values in $\{1, 0, -1\}$. This has been implemented in the procedure *get_fluid_velocity*. The boundary conditions for the fluid equation depend on the phase as well. This has been taken into account in the procedures *add_temporal_boundary_points* and *apply_boundary_conditions*. The loss function defined in *compute_pde_residual* forces the neural networks $(\bar{T}_f^\theta, \bar{T}_s^\theta)$ to approximately satisfy the equation above for the fluid temperature in the interior points which are again given by Sobol points as training data. This then allows us to infer values for the solid temperature $\bar{T}_s$ after training the networks. The loss function on *compute_loss* contains the usual loss for the fluid temperature $\bar{T}_f$ at the temporal and spatial boundary points.

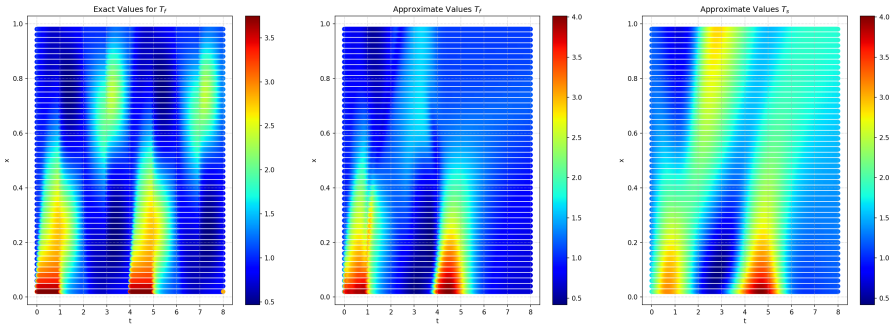

Figure 3: Exact values for T_f beased on measurement data and predictions based on the trained PINN for T_f, T_s.

Task 3: Applied Regression

Neural networks for regression tasks have been implemented in PyTorch on the dataset *California Housing Data (1990)*. The dataset contains various features related to sociodemographic statistics such as *medianIncome*, *totalBedrooms*, *population*. The goal is to predict the target variable *median_house_value* based on the feature variables as input.

Various data transformations (feature scalings) have been examined on the input data to accomplish this task:

1. Scaling down feature variable
2. Standardizing/normalizing input data
3. Log-transforming input data
4. Removing highly correlated input data

To allow to examine the accuracy of our prediction on new data, the dataset has been split into training and testing data. The distribution of the numerical features have been studied. To remove skewed data (which might reduce the effect of outliers on the regression), it has been attempted as well to log-transform the input data, which however led to poor prediction results. Correlations of the numerical features has been studied as well. To remove the effect of dimensionality and avoid overfitting, a naive approach has been used by removing highly correlated feature variables.

The targer variable has been scaled down by a factor of 100000, which has been advantageous in terms of learning the neural networks.

For the log-transformed input data, a fully connected neural network with tunable parameters (amount of hidden layers, neurons per layer) with a single output-layer for the target variable and with a *tanh*-activation for the hidden layers has been implemented. However the prediction based on the log-transformed input data led to rather poor prediciton results on the test data.

For the standardized input data, a fully connected neural network has been implemented as well. However here we used a $ReLU$-activation for the hidden layers and for the final output-layer. The $ReLU$-activation on the output-layer prevents negative predictions for the target variable.

Various parameters for the neural network have been tried out. It is found that many hidden layers and neurons lead to rather poor prediciton results on the test data, which is due to overfitting on the training data. A satisfying neural network was given by a very simple architecture: A single hidden layer with 2 neurons. The calculated *MSE* on test data and on the scaled target variable has been 0.5894 which is comparable (but slightly above) other reference results on the same dataset.

Task 4: Robustness of PINNs and Transferability

Consider the linear elliptic PDE given as differential operator:

$$L_f u(x) \coloneqq \nabla(A(x)\nabla u(x)) - f(x) \text{ for } x \in B_1 \text{ and } u \in C^2\left(B^1\right).$$

The matrix A satisfies the usual ellipticity condition and $f \in L^\infty(B_1)$.

Set $f_\delta \coloneqq f + \delta\varphi$ for $\delta > 0$ and $\varphi \in C^\infty_{\text{Comp}}$ such that $\varphi \leq 1$. Assume $u, u_\delta \in C^2(B^1)$ are solutions of L_f, L_{f_δ} respectively (I.e. $L_f u = 0$, $L_{f_\delta} u_\delta = 0$). By Linearity of the PDE, $u - u_\delta \in C^2(B^1)$ is a solution of L_{f-f_δ}. Hence applying the Schauder estimate and noticing that $\varphi \leq 1$ and $f - f_\delta = \delta\varphi$ we get the following inequality:

$$\|u - u_\delta\|_{C^1\left(B_{1/2}\right)} \leq C\left(\|u - u_\delta\|_{L^\infty(B_1)} + \delta\right)$$

Assume we have a PINN-approximation $u_\theta \coloneqq NN_\theta(u)$ such that $\|u_\theta - u\|_{C^1(B_1)} \leq \epsilon$ for some $\epsilon > 0$.

Hence applying the triangle inequality and using the inequality above, we get the following estimate:

$$\|u_\theta - u_\delta\|_{C^1\left(B_{1/2}\right)} \leq \epsilon + C\left(\|u - u_\delta\|_{L^\infty(B_1)} + \delta\right)$$

Notice that the differential operator is continuous (it is enough to prove it for L_0, for which it follows from linearity and boundedness). The continuity and the estimate above have the following consequences when we let $\delta > 0$ approach 0:

- The gradient of the PINN-approximation is bounded, which is an advantage in terms of learning and for minimizing loss functions.
- By continuity, we can choose $\delta > 0$ small enough such that e.g. $\|u_\theta - u_\delta\|_{C^1\left(B_{1/2}\right)} \leq 2\epsilon$. Hence the PINN-approximation is robust w.r.t. perturbations.

The previous consideration leads us to propose the following *transfer learning algoritm*:

1. Train a PINN for $u - u_\delta$.
2. Set $\delta > 0$ as small as possible or equal to 0.

Training a PINN for $u - u_\delta$ is beneficial since this way we replace the possibly complicated function $f \in L^\infty(B_1)$ with a much better behaved, compactly supported function, which might speed up optimization. By setting $\delta > 0$ as small as possible the trained PINN-approximation is an approximation for u as well, which follows from the robustness property discussed above.

References

[1] C. Bishop and H. Bishop, *Deep Learning: Foundations and Concepts*, Springer, Switzerland, 2024.

[2] *California Housing Data (1990)*, available at `https://www.kaggle.com/datasets/harrywang/housing`.

[3] *Regression with Neural Networks using PyTorch*, available at `https://www.kaggle.com`.