Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

AI in the Sciences and Engineering **Spring Semster 2024**

Student: Carla Judith López Zurita

Project 1

Due date: Friday 12 April 2024, 23:59

The main objective of the project is to apply the concepts learned in class by implementing our own machine learning algorithms. The tasks are related to solving differential equations using a Physics Informed Neural Network (PINN), as well as solving an inverse problem. The project also includes a regression problem and an optional task to test the robustness of a given PINN. My attempt at solving each of the mentioned tasks will be described in detail below.

1. PINNs for solving PDEs

The first task consists in solving the system of equations given by the heat equation for a fluid and a solid in a thermal storage. The equations are as follows:

$$\frac{\partial T_f}{\partial t} + U_f \frac{\partial T_f}{\partial x} = \alpha_f \frac{\partial^2 T_f}{\partial x^2} - h_f(T_f - T_s) \quad \text{for} \quad x \in [0,1], \quad t \in [0,1], \tag{1}$$

$$\frac{\partial T_s}{\partial t} = \alpha_s \frac{\partial^2 T_s}{\partial x^2} + h_s(T_f - T_s) \quad \text{for} \quad x \in [0,1], \quad t \in [0,1], \tag{2}$$

with appropriate boundary conditions and constant values defined in the project description. To solve this problem, a two-outputs neural network $(t, x) \rightarrow (T_f^\theta, T_s^\theta)$, with tunable parameters θ, was trained. The implementation was based on the Tutorial 2 presented in class. It consists on a very straight-forward implementation of a feed-forward neural network using pytorch. The neural network uses SiLU and Linear activation functions and has with adjustable number of layers and neurons, for which I chose 2 and 100 respectively. The numbers reveal a dense but not very deep network. The neural network was trained using the LBFGS optimizer with 10,000 iterations done over a single epoch. The optimizer parameters include a learning rate of 0.5, a history size of 150, with a strong Wolfe condition. The loss function is a custom function that consists of the sum of the mean squared error for the PDEs and the boundary conditions. Both Dirichlet and Von Neumann boundary conditions were used, with the latter requiring a derivative. This and all other derivatives required by the system of equations were calculated using the autograd function. The results of the loss function when training are shown in Figure 1. The plot shows the loss function decreasing steadily and smoothly, reaching a minimum value of $1 \times 10^{-4.48}$. Training took around 6 minutes to complete on an M1 processor. Figure 2 shows the visualization of the results of the PINN for the heat equation. The plot shows the approximate solution for the fluid T_f and solid phases T_s at the end of the simulation. The results show a smooth and continuous solution that seems to be a good approximation of the real system.

2. PDE-Constrained Inverse Problem

In the second task, the goal is to solve an inverse problem for the following equation,

$$\frac{\partial T_f}{\partial t}(x,t) + U_f(t) \frac{\partial T_f}{\partial x}(x,t) = \alpha \frac{\partial^2 T_f}{\partial x^2}(x,t) - h_f(T_f(x,t) - T_s(x,t)) \quad \text{for} \quad x \in [0,1], \quad t \in [0,8], \tag{3}$$

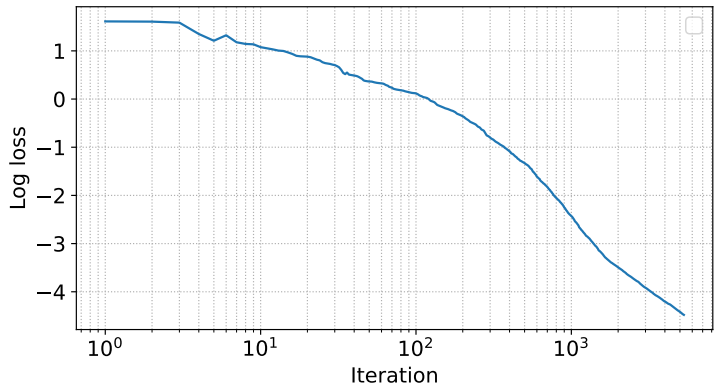

Figure 1: Loss function during training.

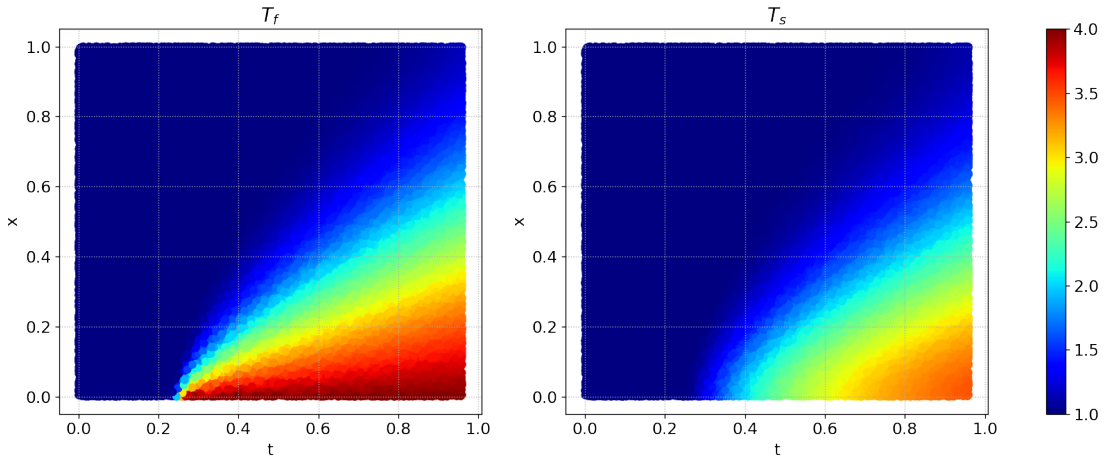

Figure 2: Approximate solution for the fluid T_f and solid temperatures T_s.

where the goal is to find the solid temperature T_s through all 8 phases of the simualtion. The equation is accompanied by a series of initial and boundary conditions (Dirichlet nd Von Neumann), as well as values for the time dependent parameter $U_f(t)$. These correspond to the different phases of the system, namely charging, idle and discharging. The problem is solved by training two different neural networks with the same architecture, each one with a different output corresponding to the fluid and solid temperatures. The neural networks follow a similar architecture as the one in Task 1, but has a Tanh activation function instead and uses Xavier initialization. The neural network were trained using the LBFGS optimizer with 10,000 iterations over two epochs, with a learning rate of 0.3. The loss function is again a custom function that depends on the output of both neural networks, the PDEs and the initial and boundary conditions. The plot regarding the loss function during training is shown in Figure 3. The loss function decreasing steadily and smoothly, and reaches a minimum value of $1 \times 10^{-0.827}$. The training took around 30 minutes to complete on an M1 processor. The results of the inverse problem are shown in Figure 4. The plot show a good approximation of the solid temperature T_s through all phases of the simulation. We can see the "discovered" physics in the plot, as the solid temperature T_s shows an interesting pattern that seems to be related to the fluid temperature T_f.

3. Applied Regression

This task consists in solving a regression problem using a neural network. The problem is based on the California Housing dataset [3]. The goal is to predict the median value of a house in California,

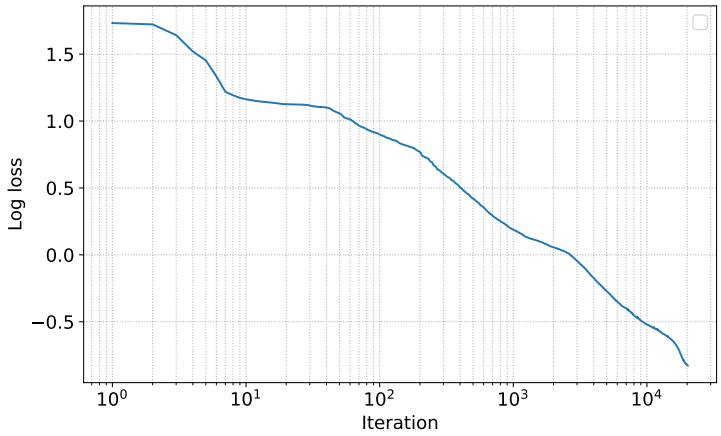

Figure 3: Loss function during training.

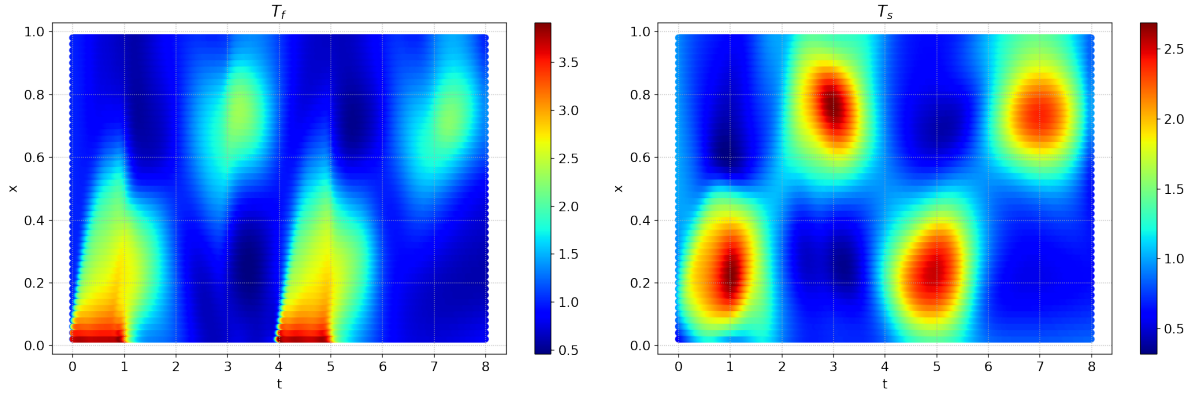

Figure 4: Results of the inverse problem for the system of equations, showing the fluid T_f and solid temperature T_s.

given a set of features. For our purposes, the dataset was augmented following the same procedure as the one presented in the tutorial by Larchenko [2]. All the numerical features except for the target variable are normalized before training using the StandardScaler function from the scikit-learn library. The target variable is also normalized using the MinMaxScaler, but this is done by the trainer since we need to recover the original values for the evaluation metrics.

The dataset is split into 75% training and 25% testing. The network has one output, uses ReLU activation, and is composed of three hidden layers with double the features' neurons each. It includes a Sigmoid layer for output normalization (0 to 1). The Adam optimizer was used with an adaptive learning rate starting at 1×10^{-2}, decreasing every 15 epochs, and weight decay of 1×10^{-6}. The loss function used is the mean squared error.The datasets were assembled using the DataLoader class, with a batch size of 150, and pin memory activated. The training is done over 200 epochs, and takes less than five minutes to complete on an M1 processor. Loss function results are shown in Figure 5. The evaluation metrics for the regression problem with both the normalized and unnormalized values are shown in Table 1.

Metric	**Normalized**	**Unnormalized**
Mean Squared Error (MSE)	0.0115	2,698,856,223.25
Root Mean Squared Error (RMSE)	0.1071	51,950.52
Mean Absolute Error (MAE)	0.0697	33,781.57
R-squared (R^2)	0.7949	0.7949

Table 1: Comparison of Normalized and Unnormalized Performance Metrics

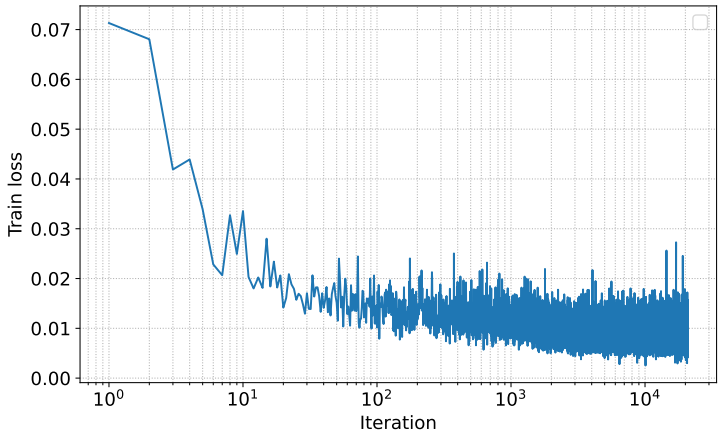

Figure 5: Loss function during training using normalized target variable.

Despite achieving a minimum MSE around 0.01, the model's evaluation metrics, including mean absolute error and R-squared, indicate good performance but not superior to a gradient boosting model presented by Larchenko. The neural network's RMSE is approximately 52,000, slightly worse than the gradient boosting model's 45,000. This suggests the neural network may have reached a minimum and didn't converge to a better solution.

4. Robustness of PINNs and Transferability

Taking the elliptic PDE,

$$\nabla \cdot (A(x)\nabla u(x)) = f(x) \quad \text{in}, \quad B_1, \tag{4}$$

the following version of the Schauder estimate can be used to establish a bound on the C^1 norm of the solution u on the subset $B_{1/2}$ of the domain B_1,

$$\|u\|_{C^1(B_{1/2})} \leq C_\epsilon(\|u\|_{L^\infty(B_1)} + \|f\|_{L^\infty(B_1)}). \tag{5}$$

We want to establish some bound on $\|u_\theta - u_\delta\|_{C^1(B_{1/2})}$, where u_θ and u_δ are the solutions of the PDE with source terms f_θ and f_δ respectively. By the triangle inequality,

$$\|u_\theta - u_\delta\|_{C^1(B_1)} \leq \|u_\delta - u\|_{C^1(B_1)} + \|u_\theta - u\|_{C^1(B_1)}. \tag{6}$$

Using the Schauder estimate, we can say that,

$$\|u - u_\delta\|_{C^1(B_{1/2})} \leq C_\epsilon(\|u - u_\delta\|_{L^\infty(B_1)} + \|f - f_\delta\|_{L^\infty(B_1)}). \tag{7}$$

And since we know that $f_\delta := f + \delta \cdot \phi$, with $\phi \leq 1$, the L^∞ norm of the difference of the source terms is bounded by δ,

$$\|u - u_\delta\|_{C^1(B_{1/2})} \leq C_\epsilon(\|u - u_\delta\|_{L^\infty(B_1)} + \delta). \tag{8}$$

We also know that $\|u_\theta - u\|_{C^1(B_1)} < \epsilon$ for some ϵ. And so, putting it all together,

$$\|u_\theta - u_\delta\|_{C^1(B_{1/2})} \leq C_\epsilon(\|u - u_\delta\|_{L^\infty(B_1)} + \delta) + \epsilon. \tag{9}$$

The established bound can help us analyze the robustness of the PINN solutions to perturbations are present. It states that the difference between a PINN solution and a perturbed solution is limited by the difference in perturbation, along with δ and the ϵ. This means that the PINN solution is robust to perturbations in the source terms and initial data, as long as these are small. In relation to transfer learning strategies, we can use the bound to establish a strategy to use the PINN solution

for a different problem. Some general stretegies include using a petrained model with the exception of the last layer, or some specific neurons, as discussed by Desai et al. [1]

The fact that the differential operator is a linear, local operator also affects the robustness of the PINN solution. This lets us know that the solution of the PINN is only affected by the local behavior of the source terms and initial data. In other words, the solution is robust as long as the perturbations are small and local. It can also be improved by using higher-order Sobolev spaces to determine the extent of the perturbations, as well as to make sure they are smooth and continuous. This will allow us to determine how much they will affect higher-order derivative terms, should there be any in the PDE.

References

[1] Chandrajit Bajaj, Luke McLennan, Timothy Andeen, and Avik Roy. Recipes for when physics fails: Recovering robust learning of physics informed neural networks. *arXiv preprint arXiv:2110.13330*, 2021.

[2] Ilialar Larchenko. California housing analysis and preciction. `https://www.kaggle.com/code/ilialar/california-housing-analysis-and-preciction`, 2024. Accessed: April 08, 2024.

[3] Harry Wang. California housing data (1990): California housing price prediction. `https://www.kaggle.com/datasets/harrywang/housing`, 2024. Accessed: April 08, 2024.