Deep Learning for Scientific Computing Project B, Task 1

Chihiro Okuyama, Karin Yu, Luca Sacchi, Nico Graf

July 2023

1 Introduction

We chose the first task, Eigen Value Problems with PINNs, in our group. The task is based on the paper Physics-Informed Neural Networks for Quantum Eigenvalue Problems [JMP22]. In the aforementioned paper a method is presented ot find eigenvalues and eigenfunctions for the stationary Schrödinger equation with physical informed neural networks (PINNs).

Since the group members have a diverse background, this task was chosen, as the eigenvalue problem is a problem which appears in various fields, e.g., physics, or structural dynamics. Therefore, everyone of us could apply this to our own research fields.

The baseline task consists of applying the methods described in the paper to the one dimensional boundary value problem (BVP)

$$\begin{aligned} &u_{tt} + \lambda^2 u = 0 \qquad t \in [0, L] \\ &u(0) = 0, \qquad u(L) = 0. \end{aligned} \tag{1}$$

For this problem we can derive the analytical solutions, which are given by

$$u_n(t) = c_2 \sin\left(\frac{n\pi}{L} t\right) =: c_2 \sin\left(\lambda_n t\right), \tag{2}$$

As the exact task is to find the first four eigenvalues with their respective eigenfunctions, it leaves the freedom to choose L and c_2. Or rather it is necessary to enforce one more condition to fix c_2 as described in the next sections.

For the extension of the baseline task, we attempt to solve the one dimensional Schrödinger equation given by Equation 3 for the single finite well potential $V(x)$ in Equation 4.

$$\left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + V(x)\right]\psi(x) = E\psi(x), \tag{3}$$

$$V(x) = \begin{cases} 0 & -\ell \leq x \leq \ell \\ V_0 & \text{otherwise} \end{cases}, \tag{4}$$

where $\hbar$ and m are the reduced Planck constant and mass, which are set to $\hbar = m$, $\psi(x)$ and E the eigenfunction and eigenvalue, and $\ell = 1$ and $V_0 = 20$ the depth of the quantum well. The eigenfunctions must decay to infinity outside of the walls, however, for numerical reasons they are set to $\psi(x_L = -6\ell) = 0$ and $\psi(x_R = 6\ell) = 0$.

2 Methodology

2.1 Network

We use almost the same neural net as in the paper [JMP22]. This network consists of one single layer $In_0 : I \to \lambda$, with a constant input and a constant output, and learns λ. And two fully connected hidden layers as well as a final dense layer, the output layer. Moreover, as in the paper we chose the activation function $\sin(x)$.

The difference to the network described in the paper [JMP22], is that we concatenate the inputs to every layer of the network with λ. This is also what the authors of the [JMP22] do in the attached code and it seemed to give better results.

2.2 Loss Function

To enforce the zero boundary conditions, we make use of the parametric approach, that is

$$\tilde{f}(t,\lambda) = N(t,\lambda) \cdot \left(1 - e^{-(x-x_0)}\right)\left(1 - e^{-(x_f - x)}\right). \tag{5}$$

We point out that despite the notation $\tilde{f}(t,\lambda)$, the neural net only takes t as an input because λ is learned by a constant input, the first neural network.

During training of the neural network, we use a loss function that consists in principle of three components, namely a PDE loss, a normalization loss and an orthogonality loss. In the following, we denote the batch size by M and the batch samples by $(t_i)_{i\in\{1,\dots,M\}}$. Moreover, as above $\tilde{f}(t,\lambda)$ is the approximation of the eigenfunction by the neural network, where λ is as well computed by the neural network but independent of the input sample t_i.

For the PDE loss we use the mean squared error of the given BVP

$$L_{pde} = \frac{1}{M}\sum_{i=1}^{M}(\mathcal{L}\tilde{f}(t_i,\lambda) - \lambda^2 \tilde{f}(t_i,\lambda))^2, \tag{6}$$

where $\mathcal{L}$ is the operator defined by the BVP and in the baseline task for example we have $\mathcal{L} := \frac{\partial^2}{\partial t^2}$.

The second loss used is the normalization loss

$$L_{norm} = \left(\sqrt{\sum_{i=1}^{M}\tilde{f}(t_i,\lambda)\cdot\tilde{f}(t_i,\lambda)} - \frac{M}{L}\right)^2, \tag{7}$$

The normalization with respect to $\frac{M}{L}$ is chosen over 1 (as suggested in the task) because with 1 the network becomes more sensitive which leads to exploding losses and training instabilities.

The last loss is the orthogonality loss which is used to find new orthogonal solutions once a solution or multiple solution have been discovered, since the eigenfunctions are orthogonal to each other.

$$L_{orth} = \sqrt{\left(\sum_{i=1}^{M}\psi_{eigen}(t_i)\cdot\tilde{f}(t_i,\lambda)\right)^2}, \tag{8}$$

in which ψ_{eigen} describes the sum of the previously discovered eigenfunctions. Of course, this loss is zero if the network is training to find the first eigenfunction, because $\psi_{eigen} \equiv 0$ in this case.

The total loss is then computed by a weighted sum of the three losses with

$$L_{tot} = w_{pde}L_{pde} + w_{norm}L_{norm} + w_{orth}L_{orth}. \tag{9}$$

2.3 Stopping

Another important part of the approach is the way we decide, what we accept as a solution, i.e. to what accuracy we aim to train. We use the two criteria described in the paper.

The first is based on a moving average of the last losses. Here we consider the last W computed L_{tot} and take the average. If this average is increasing we consider the first criteria as fulfilled because an increasing moving average indicates no improvement in training.

For the baseline task, we use instead of the moving average only the criteria that we stop if the last three iterations had increasing losses.

The second criteria is the loss of the current iteration itself and this criteria is fulfilled if $L_{tot} < th_{oc}$ where the threshold th_{oc} depends on the number of eigenfunctions we previously found, which we indicate by the index oc.

Once both criteria are fulfilled, we break the current training loop and continue searching the next solution in a new epoch.

3 Experiments

3.1 Baseline Task

As described in the introduction, Section 1, the task consists of finding eigenvalues to the BVP in (1). Using the methodology as described above in Section 2, we simply give the hyper-parameters used to attain the results below.

First of all, we used the L-BFGS optimizer with a maximum of 5000 iteration internally, and variable learning rates lr_{oc} to find the different orthogonal solutions. The exact learning rates are described below. Note that the maximum of $5,000$ iteration was never reached, as we always terminate due to stopping criteria (Subsection 2.3) being reached.

Then we chose $L = \pi$, as we then expect to get $\lambda_n \in \mathbb{N}$. c_2 is fixed implicitly by the loss L_{norm}. Then for the network, we chose the layers to have 100 neurons each. And for the samples $(t_i)_{i\in\{1,\dots,M\}}$, we chose $M = 1200$ which we pick only once before training with a Sobol engine.

The results were obtained and cross-checked on two different devices (MacBook M1 Pro with OS 13.0.1 and Macbook Pro M1 max with OS Ventura 13.3.1 (a)) using Python versions 3.9.1 and 3.10.1, respectively. We would like to point out the limitation of the proposed algorithm in the paper [JMP22] which required us to invest a lot of time in parameter tuning. Therefore, the proposed algorithm is not very robust and required very careful hyper-parameter tuning, as it is stated in the following paragraph.

First, we set the stopping thresholds to $th_0 = 0.01, th_1 = 0.06, th_2 = 1.4, th_3 = 0.45$. Then we chose the learning rates $lr_0 = 0.05, lr_1 = 0.04, lr_2 = 0.05, lr_3 = 0.04$. The weighing of the loss function turns out to be the most complicated. For all runs, after $L_{pde} < 1000$ for the first time, we use $w_{pde} = 1$. For the first eigenfunction we have $w_{norm} = 1, w_{orth} = 0.04, w_{pde} = 1$. The second eigenfunction is found with $w_{norm} = 1, w_{orth} = 0.02, w_{pde} = 3$ in the beginning and then we change w_{norm} to 0.8 after the normal loss dropped below 0.01. And as soon as $L_{orth} < 0.001$ we set $w_{orth} = 0.001$. Then for the third eigenfunction we start with $w_{norm} = 1, w_{orth} = 5, w_{pde} = 10$. And after $L_{orth} < 0.02$ we change $w_{orth} = 0.001$. For the last eigenfunction we then go with $w_{norm} = 1, w_{orth} = 0.1, w_{pde} = 1$.

As results, we found all four eigenfunctions and the corresponding eigenvalues with relatively small errors as it can be seen in Figure 1. In particular, the mean square errors with respect to the true solution are $\mathcal{L}_1 = 6.6974e-06$, $\mathcal{L}_2 = 4.9508e-06$, $\mathcal{L}_3 = 7.3538e-05$ and, $\mathcal{L}_4 = 0.0003$.

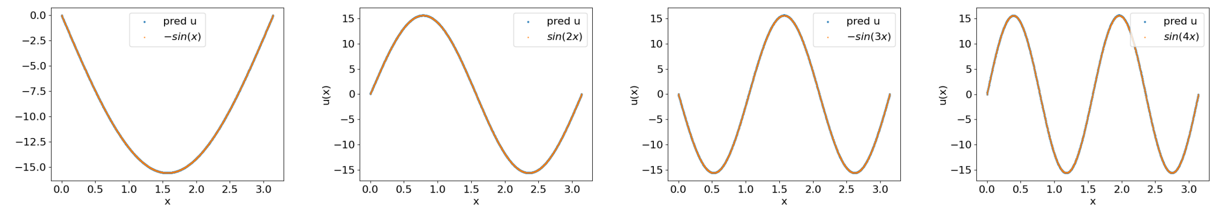

Figure 1: Results from baseline task compared with the analytical solution with the eigenvalues $\lambda = 1, 2, 3,$ and 4.

3.2 Extension Task: Single Finite Well

Similarly to the baseline task, the problem has been presented in Section 1 and the methodology in Section 2. For the optimizer, L-BFGS optimizer, and variable learning rates between 0.2 and 0.1, depending on the number of orthogonal solutions. For the losses we use the exact same as in the baseline task, with the weights equal to 1 for the pde- and normal-loss and either 0.01 or 0.02 for the orthogonal loss. For stability reasons, we set weight of the orthogonal loss to 0 if the orthogonal loss falls below 0.0001.

We have noticed significant inconsistencies in the paper [JMP22], which made the implementation much harder. For instance, the finite well potential function defined by the author, where the boundaries are set to $0 \leq x \leq \ell$ does not coincide with the resulting plots, which are symmetric around $x = 0$. Furthermore, they state that they chose $\ell = 1$, however, in the plots the eigenfunctions should decay at these boundaries to 0 but they do not. Therefore, we have decided to adapt the functions in order to fit the plots shown in the paper [JMP22] the best, which also explains the difference in the obtained eigenvalues.

For our solutions, we chose $\ell = 1.7$ which fit the plots the best. As network architecture we use the exact same as in the baseline task but we chose 50 neurons per hidden layer. The stopping are implemented exactly as described in the paper.

With this approach we fund the results in figure 2. For this task we were able to get the result on a x86_64 GNU/Linux based system with python 3.10.12.

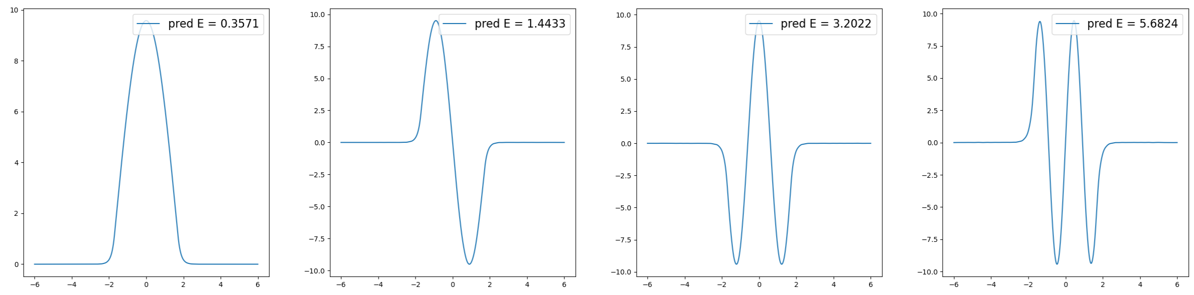

Figure 2: Predictions for single finite well problem

4 Contributions

Chihiro Okuyama Developed successfully the working baseline code employing the LBFGS optimizer, and accomplished the hyperparameter tuning, ensuring its functionality on multiple operating systems: MacOS with Python 3.10.1 and Windows with Python 3.9.6.

Karin Yu Unsuccessful trial to implement the baseline task with both Adam optimizer and L-BFGS optimizer using PINNs in PyTorch. Cross-checking the final implementations. Writing some parts of the report (mainly on experiments and extension task) and proof-reading the rest of the report.

Luca Sacchi Testing the code for the baseline task and hyper-parameter tuning. Adapted the code of the baseline task for the extension task, test and run of the code and hyper-parameter tuning for the extension task.

Nico Graf Testing and debugging the code for baseline and extension task, hyper-parameter tuning for extension task. Moreover, testing if it works better with Adam without success. Writing the framework and parts of the report (Methodology and baseline task).

References

[JMP22] Henry Jin, Marios Mattheakis, and Pavlos Protopapas. *Physics-Informed Neural Networks for Quantum Eigenvalue Problems.* 2022. arXiv: `2203.00451 [cs.LG]`.