ETH Quantum Hackathon 2024

3rd to 5th of May, ETH Zürich, Campus Hönggerberg

Pasqal Assignment

Topic: Differentiable Quantum Circuits

Introduction

Differential equations (DE) represent the mathematical model of many phenomena in various fields of natural science, economics, engineering. Machine learning ranks among modern approaches to solve differential equations: thanks to their character of universal function approximators, deep neural networks can be trained to represent the solution or the differential equation itself. The method is known as physics-informed machine learning.

Recently, physics-informed neural networks (PINN) have found extension to the realm of quantum computing in the form of Differentiable Quantum Circuits (DQC) [1]. DQC combine the power of classical data embedding via quantum feature maps with the potential of trainable quantum circuits. The exact differentiability of DQCs with respect to the variables of the equation, enabled by the latent space embedding of the quantum feature map, is key to obtain a precise solution that satisfies the DE.

Qadence is a high-level programming interface for building complex digital-analog quantum programs developed at Pasqal. Qadence offers all the elements needed to define a variational quantum circuit: flexible feature maps, parameterised single and multi-qubit gates, custom operators as quantum observables, and integration with PyTorch for automatic differentiation. In this Assignment, you will tackle the task of coding an algorithm that trains differentiable quantum circuits to solve ordinary and partial differential equations in one and two dimensions using Qadence.

To get started, you will find some useful information below:

- Qadence github: https://github.com/pasqal-io/qadence
- Installation: https://pasqal-io.github.io/qadence/latest/getting_started/installation/
- Getting started with blocks: https://pasqal-io.github.io/qadence/latest/content/block_system/
- Useful QML constructors: https://pasqal-io.github.io/qadence/latest/content/qml_constructors/
- 1D ODE example: https://pasqal-io.github.io/qadence/latest/tutorials/qml/dqc_1d/

Part I: Solving the 2D Laplace equation

Warm-up round: a 1D ODE

To illustrate Qadence's ability to define and train DQC models, you can follow to tutorial linked above to implement a QNN to solve a non-linear ordinary differential equation (ODE). Here, you are tasked with solving a similar equation:

$$\frac{df}{dx} = 4x^3 + x^2 - 2x - \frac{1}{2} \tag{1}$$

with a Dirichlet boundary condition $f(0) = 10$, and a known closed-form solution: $f(x) = x^4 + \frac{1}{3}x^3 - x^2 - \frac{1}{2}x + 10$. Note the slightly different equation and boundary condition compared to the tutorial. Can you find the needed changes to solve this variation? The exact solution and the approximation thereof obtained by training a DQC is shown in Figure 1.

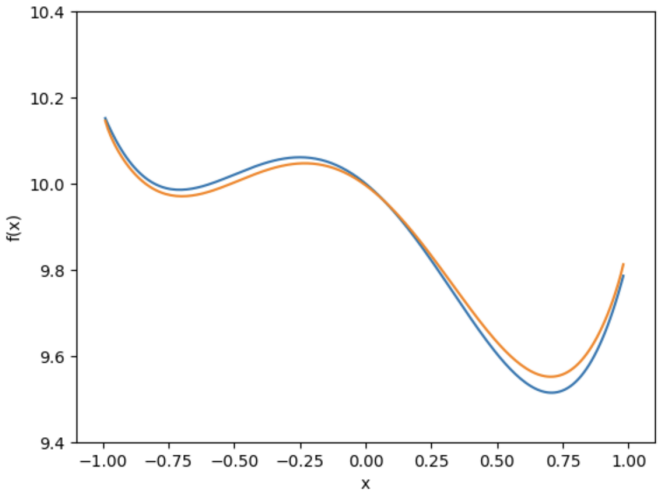

Figure 1: Solution of the simple non-linear ODE presented in the text Eq. (1) using DQC. The solution is computed within the $(-1, 1)$ domain for x and the results are compared to the known closed-form solution after gradient-based training with a random uniform sample of 20 collocation points.

Step up the game: PDEs in 2D

Many industrial and scientific problems are described as partial differential equations (PDEs), and finding their solutions is the focus of numerous classical numerical and physics-informed deep learning methods. In this part of the assignment, you will solve the Laplace equation in two spatial dimensions:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{2}$$

Eq. (2) has a known exact solution $u(x, y) = e^{-\pi x} \sin(\pi y)$ when solved with the Dirichlet boundary conditions in the domain $x, y \in [0, 1]$:

$$\begin{aligned} u(0, y) &= \sin(\pi y), \\ u(x, 0) &= 0, \\ u(X, y) &= e^{-\pi} \sin(\pi y), \\ u(x, Y) &= 0. \end{aligned} \tag{3}$$

The analytical solution to the 2D Laplace equation is illustrated in the left panel of Figure 2. The right-hand panel shows an example of the expected result, obtained by training a minimal DQC instance with Qadence.

Task breakdown

1. *DQC model.* The model approximating the solution is a Quantum Neural Network (QNN). Use Qadence's `QNN` class to define a function approximator model.

2. *Problem definition.* The problem consists of 5 terms: 1 PDE and 4 boundary conditions. Each term contributes with a residual to the global loss function. Using the QNN model and its second-order partial derivatives, define the 5 terms of the problem.

3. *Equation domains.* Each term of the equation is defined on a different domain: the Laplace function domain and its boundaries along the x and y axis. Define the equation domains and populate them with discrete and finite batches of collocation points.

4. *DQC architecture.* Decide on the feature map, the variational ansatz and the cost function observable. Assemble these elements to build the DQC model. You see how the QNN was built in the 1D ODE tutorial, and also what variations you can make in the QML constructors tutorial.

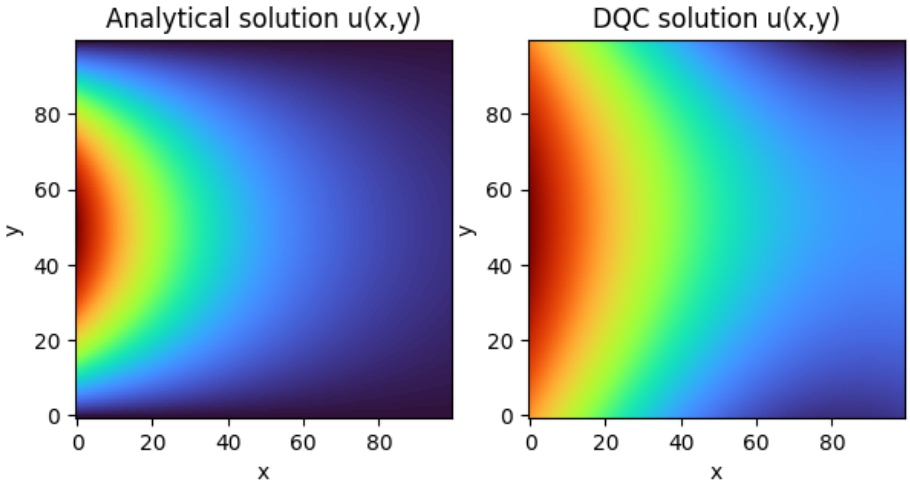

Figure 2: 2D Laplace equation, heat maps of the analytical solution (left) and one example of the DQC-model solution approximator (right).

5. *Loss function* Combine the equation terms and the DQC model to specify the physics-informed loss function and sum the differential equation's and boundaries' losses.
6. *Optimization*. Specify the classical deep learning optimisation routine.

Part II: Harmonic QNNs (Bonus)

As seen in the previous section, the convergence of the algorithm is not optimal. To obtain good QNN convergence it is essential that the model architecture is compatible with the mathematical properties of the problem. This was the reason for the development of Harmonic Quantum Neural Networks [2].
In the referenced article, the features are encoded through an imaginary time-evolution operator,

$$\text{IQFM}(x, y) = e^{-(x+iy)\mathcal{H}}, \tag{4}$$

for some Hamiltonian $\mathcal{H}$. The output, following some parameterized ansatz, is measured as the overlap with the all-zero state,

$$\text{Re}[\langle \emptyset | \psi_{\theta,x,y} \rangle]. \tag{5}$$

As described in the article, with this structure, the output of the QNN is harmonic over the inputs x and y, as expected for functions that solve the Laplace equation.

Can you use Harmonic QNNs to improve your solution?

Evaluation

You will be evaluated in three areas:

- How well your solution to the 2D Laplace equation converged.
- The scientific quality of your approach in terms of the QML concepts used. For example, if you only implemented simpler QNNs, if you implemented the Harmonic QNN, or even went creative beyond that.
- Your final presentation.

To quantify your convergence score, you should:

- Before training your model, fix your random seed using `torch.manual_seed(i)`. If you use any other packages in your code, make sure to also set their respective seeds in the same way.
- Train your model a total of 10 times, each time changing the seed from $i = 1$ to $i = 10$.

- At the end of each training run, compute the Mean Squared Error between your model solution and the exact solution over a grid of 100x100 points for $x, y \in (0, 1)$.
- Report your average MSE score over the 10 runs. The lower, the better.

The code snippet below shows how you might compute the evaluation score. Note that it assumes your model takes the input as a flat torch tensor of 2D points, which may or may not be true depending on how you code it!

Code Sample 1: Evaluation example code

```
n_points_1d = 100 # Use a grid of 100x100

# Create a length = 10.000 tensor of the (x, y) coordinates between 0 and 1
domain_1d = torch.linspace(0, 1.0, steps = n_points_1d)
domain = torch.tensor(list(product(domain_1d, domain_1d)))

# The exact solution for the Laplace equation
def exact_laplace(domain: torch.Tensor):
    exp_x = torch.exp(-torch.pi * domain[:, 0])
    sin_y = torch.sin(torch.pi * domain[:, 1])
    return (exp_x * sin_y)

# Getting the exact solution and the DQC solution
exact_sol = exact_laplace(domain).reshape(n_points_1d, n_points_1d).T
dqc_sol = model(domain).reshape(n_points_1d, n_points_1d).T.detach()

# Mean Squared Error as the comparison criterion
criterion = torch.nn.MSELoss()

# Final score, the lower the better
score = criterion(dqc_sol, exact_sol)
```

References

[1] Oleksandr Kyriienko, Annie E. Paine, and Vincent E. Elfving. Solving nonlinear differential equations with differentiable quantum circuits. *Physical Review A*, 103(5):052416, 2021.

[2] Atiyo Ghosh, Antonio A Gentile, Mario Dagrada, Chul Lee, Seong-Hyok Kim, Hyukgeun Cha, Yunjun Choi, Brad Kim, Jeong-Il Kye, and Vincent E Elfving. Harmonic (quantum) neural networks. *arXiv preprint arXiv:2212.07462*, 2022.