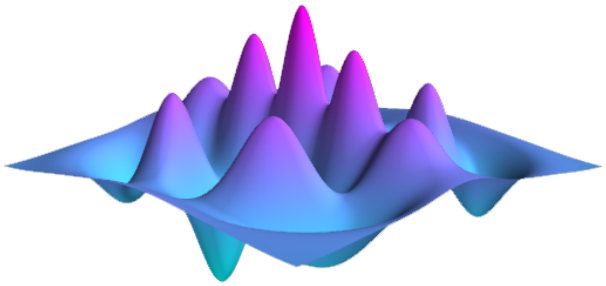

QuTiP - Quantum ToolBox in Python

Google Summer of Code Proposal - 2020

Machine Learning with QuTiP

Mentors:
Shahnawaz Ahmed & Nathan Shammah

Proposal made by:
Asad Raza

NUMFOCUS

OPEN CODE = BETTER SCIENCE

Google Summer of Code

Contents

1 Introduction and Student Information

Name: Asad Raza
Email: araza6-c@my.cityu.edu.hk; asadraza1999@hotmail.com
GitHub: https://github.com/araza6
Twitter: https://twitter.com/hermitian$_a$*sad*
Blog: https://araza6.github.io/
University: City University of Hong Kong
Major: Computing Mathematics
Minors: Physics & Computer and Science
Project Name: Machine Learning with QuTiP

2 Abstract

In recent years, there has been a rise of interesting applications of Machine Learning to solve problems in quantum physics [7]. Also, there has been a whole new inter-disciplinary field of Quantum Machine Learning [8] that aims to harvest ideas from quantum physics to make computers learn better.
This project aims to open-source tools to perform experiments/simulations that would help the users discern how machine learning can be used to solve certain problems in quantum physics.
The idea is to inculcate machine learning tools *natively* into QuTiP and integrate them with the existing QuTiP modules in order to leverage machine learning prowess in simulation open (which is QuTiP's forte) and closed quantum systems.

3 Technical Details

This part of the proposal aims to explain in detail the extension to the existing functionalities that this proposed GSoC project could bring about in QuTiP.

3.1 Making QuTiP Auto-differentiable

Currently in QuTiP, one can initialize Hamiltonians, exponentiate them for evolution – with and without noise, among other things. One limitation, however, is that one cannot natively differentiate these exponents of Hamiltonians. In order to solve this, auto differentiation libraries like JAX and/or Autograd can be used. The reason these libraries would be a good choice to integrate with QuTiP for the said purpose is that they already provide robust ways to efficiently compute the derivatives of python and Numpy functions, which is what QuTiP is largely built on.

3.1.1 Benefits of Auto-Differentiable QuTiP

With auto-differentiable QuTiP, one can differentiate the desired unitary and non-unitary operators and optimize the initial variational parameters of, say, an initial wavefunction or density matrix using gradient descent or other optimization methods.

A possible extension, if time allows, would be to allow users to natively optimize the the parameters of their initial unitary using different optimization backends. That is to say, to give users the freedom to chose beyond the simple gradient descent optimization, and experiment with for example Adagrad, Adamax, etc. Widely used machine learning libraries Pytorch and Tensorflow come pre-packaged with torch.optim and tf.keras.optimizers that provide many of those different optimizers to choose from. Integration of these library(-ies) with QuTiP, like PennyLane does with TensorFLow and Pytorch (among others) shall allow users to perform aforementioned optimization of desired unitaries with minimal friction in the workflow.

3.2 Calculating Loss with Density Matrices

In many machine learning tasks, the aim is to minimize some sort of cost function (which may or may not be convex) with respect to the given parameters. This cost function, although has some canonical representation in classical machine learning literature (one of the famous ones being the mean-squared error loss), formulation is still somewhat application dependent. The main idea however remains the same – to minimize the difference between what we *want* to predict and what are we actually predicting.

This cost function formulation can take slightly different forms when we apply machine learning to quantum physics, depending on the application. If we want to predict the target wavefunction by optimizing the initial ansatze, one way to do it is what Netket (a machine learning library for quantum many-body physics) does. The idea is to capture the similarity between the target state, ψ_{tar} and the α-parametrized variational state, $\psi_{NN}(\alpha)$ as implemented in [10].

$$\mathcal{L}(\alpha) = -log\frac{\langle\psi_{tar}|\psi_{NN}(\alpha)\rangle}{\langle\psi_{tar}|\psi_{tar}\rangle}\frac{\langle\psi_{NN}(\alpha)|\psi_{tar}\rangle}{\langle\psi_{NN}(\alpha)|\psi_{NN}(\alpha)\rangle} \tag{1}$$

Loosely speaking, one can say that 1 does capture fidelity between ψ_{tar} and $\psi_{NN}(\alpha)$ if normalization and absolute value-squared is ignored. This notion of loss calculation, however, would need further development if the idea is to be generalized to *mixed* states. Since mixed states are characterized by density matrices, we would be needing a tool that (intuitively speaking) measures the difference (or similarity) between two density matrices of the target state, ψ_{tar} and of the parametrized predicted state (parametrized by, say, α), $\psi_{pred}(\alpha)$.

3.3 Unitary Learning

Present Quantum Machine Learning libraries like PennyLane [11] and the recent TensorFlow Quantum [9] focus on circuit based learning. QuTiP's Quantum Machine Learning module has a potential to stand out if it provides tools to understand Hamiltonian Learning. A simple example of unitary learning is given in the following code-snippet using QuTiP where we start with an initial (mixed state) 4×4 density matrix and act random initial 4×4 unitary matrix with the intention to get it closer to the target density matrix (which we randomly choose and fix).

```
from qutip import *

def dm_fidel(dm_init, dm_tar, unit_init): #function to calculate the fidelity between initial and
    evolved  density matrices

 dm_new = unit_init*dm_init # State after the the unitary acts on intial density matrix
 return fidelity(dm_new,dm_tar) # QuTiP's fidelity function that calculates fidelity btw two density
    matrices

dm_init = rand_dm(4,0.5) # Initial density matrix
dm_tar = rand_dm(4,0.5) # Target/Desired density matrix
unit_init = rand_unitary(4) #Initial Unitary to act on the density matrix
print(dm_fidel(dm_init, dm_tar,unit_init))
```

The fidelity (roughly) using the above code ranges mostly from 0.4 to 0.6. This is not too surprising given that we have random matrices each time. However, the task is to improve the entries in the unitary matrix such that the fidelity improves for a given target state. This can be done by reducing the error, which for the case of pure states is formulated to be the following by [1]

$$E = 1 - (\frac{1}{M}) \sum_{l} \langle \psi_l | U^{\dagger} U(\vec{t}, \vec{\tau}) | \psi_l \rangle \tag{2}$$

where U is the target unitary and $U(\vec{t}, \vec{\tau})$ is the parameterized unitary that we aim to steer towards U, while M is the number of training examples of input and output pairs $|\psi_l\rangle$ and $U|\psi_l\rangle$ respectively.

The error can then be minimized using gradient descent [2] or similar optimization routines.
One way the aforementioned idea can be generalized to mixed states is the following:

$$E = 1 - (\frac{1}{M}) \sum_{l} tr(\rho_l U^{\dagger} U(\vec{t}, \vec{\tau})) \tag{3}$$

where ρ_l is the target density matrix instead of the target ket.

Therefore, the goal here is to open-source tools such that the target unitary U can be learnt efficiently for both pure and mixed states. Some useful QutiP modules here shall be the method `q.tr()` of class `Qobj`, random quantum states and operators like `rand_ket`, `rand_dm`, and `rand_unitary` and the fidelity calculation method `fidelity()`.

3.4 Dissipative dynamics prediction for non-unitary evolution

A general extension to the aforementioned unitary learning idea is to learn the dynamics of non-unitary open quantum systems – which is QuTiP's forte. The idea here is somewhat similar. We have a target density matrix and the system's Hamiltonian. The goal is to optimize for the free parameters in the Lindblad terms such that the target density matrix can be reached efficiently starting the evolution from an initial density matrix. Such evolution of open (and also closed) quantum systems can be studied using QuTiP's `mesolve` module. A callback function can be passed as argument to `e_ops` in `mesolve` to calculate the fidelity/cost function at each time step in `tlist` in order to optimize the Lindblad free parameters at each discreet time step.

One way to optimize the free parameters in the Lindblad terms would be to run, say, gradient descent to find the free parameters such that fidelity of the evolved density matrix and initial density matrix is improved – which can be cast as a cost function for this problem.
There has been recent interest in this area [4] [5] [6] and inspiration can be taken from these and similar works along the way.

4 Schedule of Deliverables

4.1 Community Bonding Period: May 2, 2020 - June 2, 2020

During the community bonding period, I will start looking into the relevant literature that I think might be useful for the project and to brush up on relevant physics-based concepts. I will spend this time understanding the code base and producing minimal working code to motivate the module under development. Lastly, I will make try as much as possible to get to know the QuTiP developer community, interact with them and head start a great journey.

4.2 Week 1 and 2

Work on integrating auto-diff libraries, most likely JAX, into QuTiP. I will write wrappers in this time that can convert QuTiP's `Qobj` into JAX compatible functions to be easily differentiated.

4.3 Week 2

Complete the method to calculate the gradient of the fidelity function with respect to any parameters of interest. This might just be an addition to the list of fidelity methods already provided by QuTiP. Possibly use the delivered items in the first two weeks for this task.

4.4 Week 3

Make a class for optimizers and implement canonical optimization methods in it – Stochastic Gradient Descent, Adagrad, etc.

4.5 Week 4

Write a method for unitary learning that takes initial unitary and density matrices and target density matrix, along with one of the optimizer methods built in Week 3 to perform unitary learning. This method would return the optimized unitary.

Phase 1 Evaluation

4.6 Week 5

Collect data where input is the initial state (pure or mixed) and output is the final state after the action of the unitary and reproduce [2].

4.7 Week 6

Write `pytests` for the code written in Weeks 1 - 5 and write helper functions, where need be.

4.8 Week 7 and 8

Extend the module (method) for unitary learning to open quantum systems such that passing in the collapse operators would treat the system as open (like `mesolve`) and would optimize the method for free parameters of the dissipative terms. An extension in the open quantum system version would be to also have an argument that lets the user to print the optimized free parameters and not just the final density matrix.

Phase 2 Evaluation

4.9 Week 9

Explore the performance of different cost functions. [1] does it as shown in **??**. One possible way to do for density matrices is, in my opinion, 3, but other optimal cost functions can be formulated to plot graphs of accuracy against time (for training) in order to benchmark all those cost functions and to ship them. A class of cost functions, similar to that of optimizers can be designed to ship all the cost functions that might be of interest to the users, depending on the use-cases and trade-offs.

4.10 Week 10 and 11

Complete documentation of the newly built modules, fix bugs (if any) and write additional unit tests.

4.11 Week 12

Complete Jupyter Notebook tutorials for Auto-diff, unitary and non-unitary learning.

Final Evaluation

5 Other Information

5.1 Why QuTiP?

QuTiP is the one of the most used, or I should say the most used library, for quantum physics simulations. Recently, QuTiP has been extending its focus to quantum information. GSoC 2019 summer project on `qutip.qip.noise` module is a testimony to that.
As a Math major (and physics and computer science minor), I believe my background and interest in the fields of machine learning and quantum computing can complement very well with this GSoC summer project. Lastly, as an advocate for open-source, it would indeed be an honor to make tools for open science.

5.2 Relevant Background

5.2.1 Past experience

I claim a reasonably good background in machine learning and quantum information. I did some work [3] at the intersection of Reinforcement Learning and Quantum Control, after which I have been looking into ways to marry Quantum Information and Machine Learning.
In the summer of 2019, I was at Entropica Labs (a quantum machine learning startup in Singapore) on variational algorithms. In particular, applying Deep Learning machinery on the parameters (betas and gammas) in QAOA to discern if we can learn something useful about the optimal ansatz. We got some interesting results.

5.2.2 Knowledge of Libraries and/or Technology stacks

As for the familiarity with technology stacks, I have extensively used Rigetti's Forest framework for projects related to quantum computing. Pytorch and sci-kit-learn are my go-to machine learning libraries and Pennylane for quantum machine learning. Numpy is the library that I have used most in my projects. It is a library that I often use for school work as well (almost every week). Since all the above mentioned libraries are written in Python (and QuTiP as well), I claim to be fluent with Python programming in general.
In the two internships I have done in the past, I had to use git day in and day out. For any collaborative school projects (and even for hosting my personal website), I have leveraged git for version control.

There are libraries, however, that I have not tinkered around with that can be potentially useful for the project. TensorFlow Quantum [9] and NetKet[10] are the ones. Apart from the technology stack, I think I will have to put in extra effort during the project to learn about open quantum systems since the project will certainly require a strong knowledge base in that area as well, whereas I have only studied closed quantum systems in the past.

5.3 Past interaction with mentors

I have been in contact with the mentors via email shortly after QuTiP was announced as one of the participating organizations. Since I was interested in a project at the intersection of quantum physics and machine learning, I emailed the mentors about my idea to integrate machine learning in some QuTiP modules and that was well-taken.
I had a Zoom chat with Nathan and Shahnawaz, and we further polished the initial ideas and drafted new ideas about how to go about this project. Based on back and forth communication with the mentors, I was able to finally draft this proposal.

5.4 Other Commitments

I have no other commitments during the summer. For project catch-up, I will be available via Skype/-Zoom in Hong Kong Time Zone, GMT +8h.

References

[1] Seth Lloyd and Reevu Maity. *Efficient implementation of unitary transformations* https://arxiv.org/abs/1901.03431

[2] Bobak Toussi Kiani, Seth Lloyd, and Reevu Maity. *Learning Unitaries by Gradient Descent* https://arxiv.org/abs/2001.11897

[3] Xiao-Ming Zhang, Zezhu Wei, Raza Asad, Xu-Chen Yang and Xin Wang. *When does reinforcement learning stand out in quantum control? A comparative study on state preparation* https://www.nature.com/articles/s41534-019-0201-8

[4] Maria Schuld, Ilya Sinayskiy and Francesco Petruccione. *Viewpoint: Neural Networks Take on Open Quantum Systems* https://physics.aps.org/articles/v12/74

[5] Alexandra Nagy and Vincenzo Savona. *Variational Quantum Monte Carlo Method with a Neural-Network Ansatz for Open Quantum Systems* https://physics.aps.org/featured-article-pdf/10.1103/PhysRevLett.122.250501

[6] Michael J. Hartmann and Giuseppe Carleo. *Neural-Network Approach to Dissipative Quantum Many-Body Dynamics* https://physics.aps.org/featured-article-pdf/10.1103/PhysRevLett.122.25050

[7] Matthew J. S. Beach, Isaac De Vlugt, Anna Golubeva, Patrick Huembeli, Bohdan Kulchytskyy, Xiuzhe Luo, Roger G. Melko, Ejaaz Merali, Giacomo Torlai. *QuCumber: wavefunction reconstruction with neural networks* https://arxiv.org/abs/1812.09329

[8] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, Seth Lloyd. *Quantum Machine Learning* https://arxiv.org/abs/1611.09347

[9] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J. Martinez, Jae Hyeon Yoo, Sergei V. Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, Martin Leib, Andrea Skolik, Michael Streif, David Von Dollen, Jarrod R. McClean, Sergio Boixo, Dave Bacon, Alan K. Ho, Hartmut Neven, Masoud Mohseni. *TensorFlow Quantum: A Software Framework for Quantum Machine Learning* https://arxiv.org/abs/2003.02989

[10] Giuseppe Carleo, Kenny Choo, Damian Hofmann, James E. T. Smith, Tom Westerhout, Fabien Alet, Emily J. Davis, Stavros Efthymiou, Ivan Glasser, Sheng-Hsuan Lin, Marta Mauri, Guglielmo Mazzola, Christian B. Mendl, Evert van Nieuwenburg, Ossian O'Reilly, Hugo Théveniaut, Giacomo Torlai, Alexander Wietek. *NetKet: A Machine Learning Toolkit for Many-Body Quantum Systems* https://arxiv.org/abs/1904.00031

[11] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M. Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, Keri McKiernan, Johannes Jakob Meyer, Zeyue Niu, Antal Száva, Nathan Killoran. *PennyLane: Automatic differentiation of hybrid quantum-classical computations* https://arxiv.org/abs/1811.04968