Shot-adaptative optimizers for Quantum Machine Learning

By `my_favourite_team`: Charles Moussa and Adrien Suau

Motivation for shot-adaptative optimizers

- Variational Quantum Algorithms (VQAs) are heavily present for near-term quantum applications,
- Among applications, we find machine learning, especially with quantum data,
- Giving rise to the field of Quantum Machine Learning (QML),
- However, required resources in QML can be very high, especially when considering shot or measurement count,
- Hence, we require tailored optimizers for running VQAs in NISQ era.

Resource frugal optimizer

During QHack 2023, we implemented a resource-frugal optimizer named Refoqus designed for QML applications*. We demonstrate how to apply it on 3 examples of applications.

* "Resource frugal optimizer for quantum machine learning", 2022, arXiv:2211.04965

Quantum Machine Learning Applications

General setting

Given:

- a training dataset composed of quantum states defined by density matrices $\mathcal{S} = \{\rho_i\}_{i=1}^N$,
- a corresponding probability distribution $\mathcal{P} = \{p_i\}_{i=1}^N$
- a parameterized quantum model $\mathcal{M}_{\boldsymbol{\theta}}$ for some set of parameters $\boldsymbol{\theta}$

one trains the model to optimize a loss function of interest $\mathcal{L}(\boldsymbol{\theta})$.

Examples of applications

- **Variational quantum error correction**,
- **Quantum autoencoder**,
- Fidelity for Quantum Neural Networks,
- Classification, Regression cases,
- **Variational quantum state eigensolver** and diagonalization, etc.

In bold, examples can be found in the notebooks available in our GitHub repo. Our implementation use Pennylane, particularly with the VQE data of molecular datasets.

Loss functions in QML

General framework

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{\boldsymbol{i}} p_{\boldsymbol{i}} \ell(E_{\boldsymbol{i}}(\boldsymbol{\theta}))$$

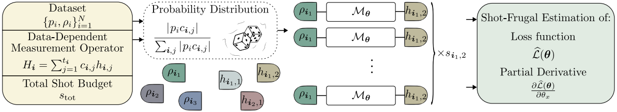
ℓ is an application-dependent function whose input is a measurable expectation value $E_{\boldsymbol{i}}(\boldsymbol{\theta})$

$$E_{\boldsymbol{i}}(\boldsymbol{\theta}) = \mathrm{Tr}[\mathcal{M}_{\boldsymbol{\theta}}(\rho_{\boldsymbol{i}}) H_{\boldsymbol{i}}]$$

Many possible forms for ℓ, linear form is mainly present in literature.
$H_{\boldsymbol{i}}$ can have many terms itself: $H_i = \sum_{j=1}^{t_i} c_{\boldsymbol{i},j} h_{\boldsymbol{i},j}$

Evaluating these many terms on a quantum computer can explode the shot count if set arbitrarily (like done in simple gradient descent).

Shot allocation in adaptative optimizer



Source: arXiv:2211.04965

To unlock shot-frugality (case of ℓ linear)

$\mathcal{L}(\boldsymbol{\theta}) = \sum_{\boldsymbol{i},j} q_{\boldsymbol{i},j} \langle h_{\boldsymbol{i},j}(\boldsymbol{\theta}) \rangle$ where $q_{\boldsymbol{i},j} = p_{\boldsymbol{i}} c_{\boldsymbol{i},j}$ and $\mathrm{Tr}[\mathcal{M}_{\boldsymbol{\theta}}(\rho_{\boldsymbol{i}}) h_{\boldsymbol{i},j}] = \langle h_{\boldsymbol{i},j}(\boldsymbol{\theta}) \rangle$

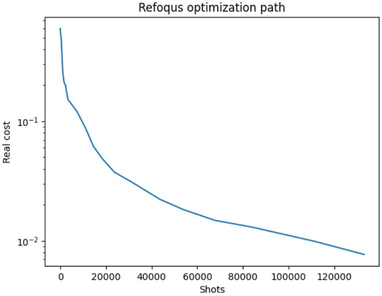
1. We distribute shots based on defining a probability distribution with $q_{\boldsymbol{i}j}$ values.
2. We evaluate each measurable expectation term $\langle h_{\boldsymbol{i},j}(\boldsymbol{\theta}) \rangle$ with the corresponding number of shots.
3. Summing up all evaluated terms, we obtain the evaluation of an unbiased estimator of the loss.
4. We can apply the parameter shift-rule to obtain an evaluation of an unbiased estimator of the gradient.
5. One can then use a shot-adaptative optimizer like gCANS (arXiv:2108.10434) to set automatically the number of shots to distribute.

See Alg.1 of arXiv:2211.04965 for the Refoqus pseudo-code.

Example: quantum autoencoder

Goal: compress quantum data. Here we have 42 states produced from VQE experiments on the H2 molecule in STO-3G basis. From 4 qubits, we compress to 2 ($t_i = 2$ terms per hamiltonian and data).

The closer the cost to 0, the better. We use as variational part 3 Strongly Entangling Layers (36 parameters). Here we have obtained a low score of 7.7×10^{-3} after 20 iterations and 132954 shots only. In the case we were using a simple gradient descent optimizer, setting 100 shots per term, one iteration would already use $42 * 36 * 2 * 2 * 100 = 604800$ shots.



Source: Notebook quantum autoencoder example on GitHub

Other examples and GPU simulations

- We also implemented a variational quantum state eigensolver (VQSE) applied on the same previous VQE data, which is the basis for quantum principal component analysis.
- As well as an example of variational quantum error correction application which can be used for devising new potential error-correction codes for quantum memory.
- Finally, we compared runtimes for VQSE when using the `lightning.gpu` plugin versus `lighthing.qubit` for state-vector simulations, demonstrating a speedup by nearly 40% in runtime with the Nvidia GPU on a laptop. The switch from one backend to another is very simple in Pennylane (see notebook on Github).

Conclusion

- We have implemented the resource frugal optimizer Refoqus in Pennylane,
- We have provided three QML applications to show how to use it,
- We also compared using the lightning.gpu plugin versus lighthing.qubit,
- We were also able to test with quantum data from the datasets made available within Pennylane,
- More applications and data can be tackled with our code.

`https://github.com/chMoussa/adaptative_vqa_optimizers`

Thanks for the hackathon and looking forward to your feedback on our Refoqus implementation!