Full-stack Quantum Machine Learnig

Matteo Robbiati

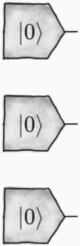
28 September 2023

A snapshot of Quantum Computing

Quantum Computing and circuits notation

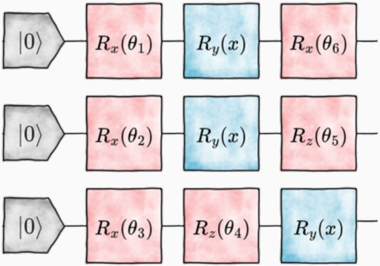
Quantum Computing and circuits notation

- Classical bits are replaced by **qubits**: $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$;



Quantum Computing and circuits notation

- Classical bits are replaced by **qubits**: $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$;
- we modify the qubits state by applying unitaries, which we call **gates**;

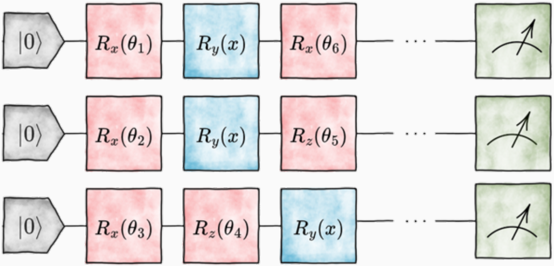


Quantum Computing and circuits notation

- Classical bits are replaced by **qubits**: $|q\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$;
- we modify the qubits state by applying unitaries, which we call **gates**;
- we extract information by calculating expected values:

$$\langle q_i | \mathcal{C}^\dagger(\boldsymbol{\theta}) \hat{O}\, \mathcal{C}(\boldsymbol{\theta}) | q_i \rangle\,,$$

with $\mathcal{C}(\boldsymbol{\theta})$ parametric circuit, $|q_i\rangle$ initial qubit's state and $\hat{O}$ arbitrary observable.



Quantum Machine Learning

Quantum Machine Learning - doing ML using QC

Machine Learning
$\mathcal{M}$: model;
$\mathcal{O}$: optimizer;
$\mathcal{J}$: loss function.
(x, y): data

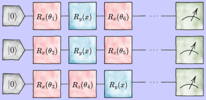
Quantum Computation
$\mathcal{Q}$: qubits;
$\mathcal{S}$: superposition;
$\mathcal{E}$: entanglement.

Quantum Machine Learning - operating on qubits

Machine Learning
$\mathcal{M}$: model;
$\mathcal{O}$: optimizer;
$\mathcal{J}$: loss function.
(x, y): data

Quantum Computation
$\mathcal{Q}$: qubits;
$\mathcal{S}$: superposition;
$\mathcal{E}$: entanglement.

Circuit execution



Quantum Machine Learning - natural randomness

Machine Learning

$\mathcal{M}$: model;

$\mathcal{O}$: optimizer;

$\mathcal{J}$: loss function.

(x, y): data

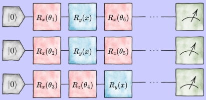
Quantum Computation

$\mathcal{Q}$: qubits;

$\mathcal{S}$: superposition;

$\mathcal{E}$: entanglement.

Circuit execution



Expected values

$y_{est} \equiv \langle q_f | \hat{O} | q_f \rangle$

Quantum Machine Learning - encoding the problem

Machine Learning

$\mathcal{M}$: model;

$\mathcal{O}$: optimizer;

$\mathcal{J}$: loss function.

(x, y): data

Expected values

$y_{est} \equiv \boxed{\langle q_f | \hat{O} | q_f \rangle}$

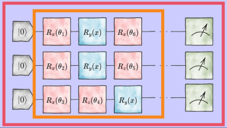
Quantum Computation

$\mathcal{Q}$: qubits;

$\mathcal{S}$: superposition;

$\mathcal{E}$: entanglement.

Circuit execution



Quantum Machine Learning!

Optimizer $\mathcal{O}$
Hybrid-strategy

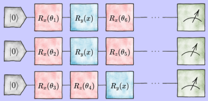
Machine Learning
$\mathcal{M}$: model;
$\mathcal{O}$: optimizer;
$\mathcal{J}$: loss function.
(x, y): data

loss function $\mathcal{J}$
$\mathcal{J}(y_{meas}, y_{est})$

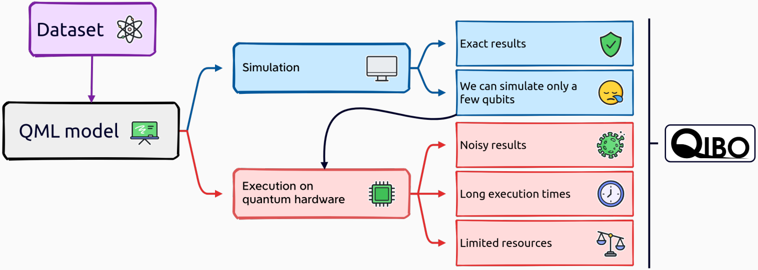
Expected values
$y_{est} \equiv \langle q_f | \hat{O} | q_f \rangle$

Quantum Computation
$\mathcal{Q}$: qubits;
$\mathcal{S}$: superposition;
$\mathcal{E}$: entanglement.

Circuit execution



Full-stack QML



Some results

High level API: Qibo

- </> define prototypes;
- </> implement training loop;
- </> simulate training.

A full-stack PDF fit — arXiv:2308.06313

High level API: Qibo

- define prototypes;
- implement training loop;
- simulate training.

Calibration: Qibocal

- calibrate qubits;
- generate platform configuration;

A full-stack PDF fit

arXiv:2308.06313

High level API: Qibo

- </> define prototypes;
- </> implement training loop;
- </> simulate training.

Calibration: Qibocal

- calibrate qubits;
- generate platform configuration;

Execution: Qibolab

- allocate calibrated platform;
- compile and transpile circuits;
- execute the model and return results.

A full-stack PDF fit   arXiv:2308.06313

High level API: Qibo

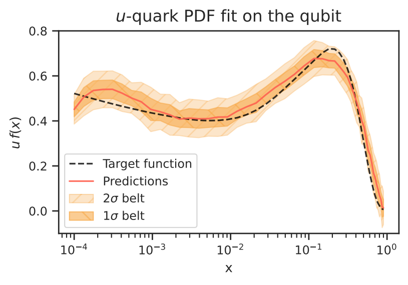
- define prototypes;
- implement training loop;
- simulate training.

Calibration: Qibocal

- calibrate qubits;
- generate platform configuration;

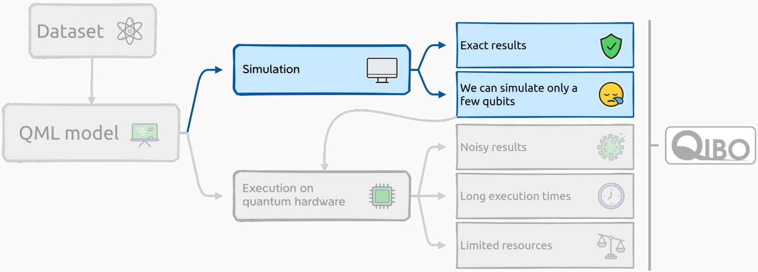
Execution: Qibolab

- allocate calibrated platform;
- compile and transpile circuits;
- execute the model and return results.



Parameter	Value
N_{data}	50
N_{shots}	500
MSE	50
Electronics	Xilinx ZCU216
Training time	2h

Simulation as first approach



- Determining Probability Density Functions (PDF).

- Determining Probability Density Functions (PDF).
- Algorithm's summary:

- Determining Probability Density Functions (PDF).
- Algorithm's summary:

1. we optimize the parameters $\boldsymbol{\theta}$ of the following adiabatic evolution:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = [1 - s(\tau;\boldsymbol{\theta})]\hat{\sigma}_x + s(\tau;\boldsymbol{\theta})\hat{\sigma}_z. \tag{1}$$

   we use the GS of the evolved H_{ad} to approximate the Cumulative Density Function (CDF);

We work on the algorithmic part   arXiv:2303.11346

- Determining Probability Density Functions (PDF).
- Algorithm's summary:

1. we optimize the parameters $\boldsymbol{\theta}$ of the following adiabatic evolution:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = [1 - s(\tau;\boldsymbol{\theta})]\hat{\sigma}_x + s(\tau;\boldsymbol{\theta})\hat{\sigma}_z. \quad (1)$$

   we use the GS of the evolved H_{ad} to approximate the Cumulative Density Function (CDF);
2. we derivate from H_{ad} a circuit $\mathcal{C}(\tau;\boldsymbol{\theta})$ whose action on the GS of $\hat{\sigma}_x$ returns $|\psi(\tau)\rangle$;

We work on the algorithmic part

- Determining Probability Density Functions (PDF).
- Algorithm's summary:

1. we optimize the parameters $\boldsymbol{\theta}$ of the following adiabatic evolution:

$$H_{\mathrm{ad}}(\tau;\boldsymbol{\theta}) = [1 - s(\tau;\boldsymbol{\theta})]\hat{\sigma}_x + s(\tau;\boldsymbol{\theta})\hat{\sigma}_z. \tag{1}$$

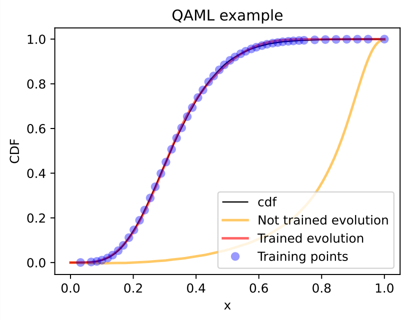
   we use the GS of the evolved H_{ad} to approximate the Cumulative Density Function (CDF);
2. we derivate from H_{ad} a circuit $\mathcal{C}(\tau;\boldsymbol{\theta})$ whose action on the GS of $\hat{\sigma}_x$ returns $|\psi(\tau)\rangle$;
3. we compute the PDF by derivating $\mathcal{C}$ w.r.t. τ using the Parameter Shift Rule (PSR).

- Determining Probability Density Functions (PDF).
- Algorithm's summary:

1. we optimize the parameters $\boldsymbol{\theta}$ of the following adiabatic evolution:

$$H_{\rm ad}(\tau;\boldsymbol{\theta}) = [1 - s(\tau;\boldsymbol{\theta})]\hat{\sigma}_x + s(\tau;\boldsymbol{\theta})\hat{\sigma}_z. \tag{1}$$

   we use the GS of the evolved $H_{\rm ad}$ to approximate the Cumulative Density Function (CDF);
2. we derivate from $H_{\rm ad}$ a circuit $\mathcal{C}(\tau;\boldsymbol{\theta})$ whose action on the GS of $\hat{\sigma}_x$ returns $|\psi(\tau)\rangle$;
3. we compute the PDF by derivating $\mathcal{C}$ w.r.t. τ using the Parameter Shift Rule (PSR).



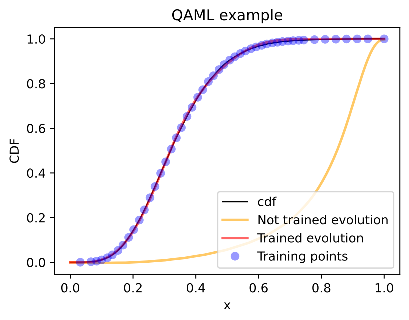
Determining Probability Density Functions (PDF).

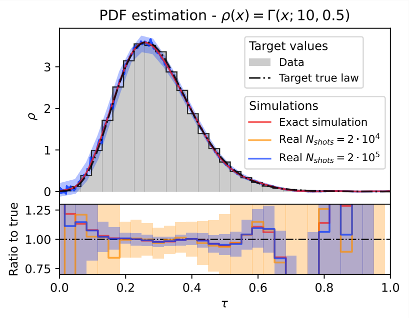
Algorithm's summary:

1. we optimize the parameters $\boldsymbol{\theta}$ of the following adiabatic evolution:

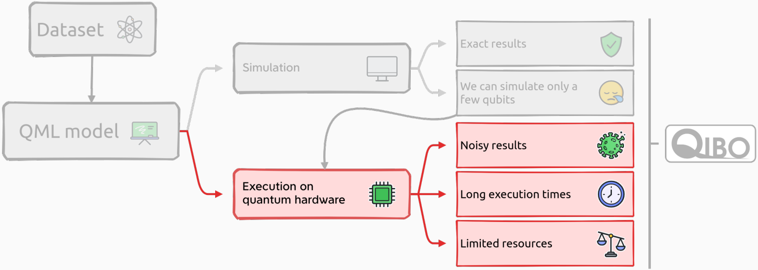
$$H_{\rm ad}(\tau;\boldsymbol{\theta}) = [1 - s(\tau;\boldsymbol{\theta})]\hat{\sigma}_x + s(\tau;\boldsymbol{\theta})\hat{\sigma}_z. \tag{1}$$

   we use the GS of the evolved $H_{\rm ad}$ to approximate the Cumulative Density Function (CDF);
2. we derivate from $H_{\rm ad}$ a circuit $\mathcal{C}(\tau;\boldsymbol{\theta})$ whose action on the GS of $\hat{\sigma}_x$ returns $|\psi(\tau)\rangle$;
3. we compute the PDF by derivating $\mathcal{C}$ w.r.t. τ using the Parameter Shift Rule (PSR).





How does my algorithm perform on a real quantum computer?



- Use Variational Quantum Circuits to calculate multi-dimensional integrals of the form

$$I(\alpha) = \int_{\boldsymbol{x}_a}^{\boldsymbol{x}_b} g(\alpha; \boldsymbol{x}) \mathrm{d}^n \boldsymbol{x}. \tag{2}$$

- Use Variational Quantum Circuits to calculate multi-dimensional integrals of the form

$$I(\alpha) = \int_{\boldsymbol{x}_a}^{\boldsymbol{x}_b} g(\alpha; \boldsymbol{x}) \mathrm{d}^n \boldsymbol{x}. \tag{2}$$

- Algorithm's summary:

Use Variational Quantum Circuits to calculate multi-dimensional integrals of the form

$$I(\alpha) = \int_{\boldsymbol{x}_a}^{\boldsymbol{x}_b} g(\alpha; \boldsymbol{x}) \mathrm{d}^n \boldsymbol{x}. \tag{2}$$

Algorithm's summary:

1. we train the derivative of a VQC w.r.t. the integral variables $\boldsymbol{x}$ to approximate $g(\boldsymbol{x})$;

Multi-dimensional integration on hardware arXiv:2308.05657

- Use Variational Quantum Circuits to calculate multi-dimensional integrals of the form

$$I(\alpha) = \int_{\boldsymbol{x}_a}^{\boldsymbol{x}_b} g(\alpha; \boldsymbol{x}) \mathrm{d}^n \boldsymbol{x}. \tag{2}$$

- Algorithm's summary:

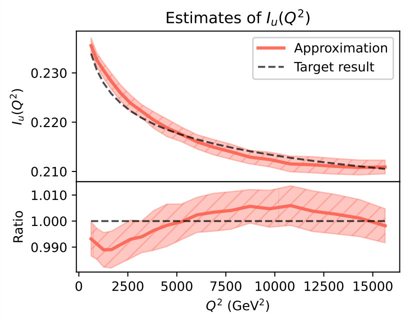
1. we train the derivative of a VQC w.r.t. the integral variables $\boldsymbol{x}$ to approximate $g(\boldsymbol{x})$;
2. we compute the derivatives using the PSR, which allows the same circuit $\mathcal{C}$ to be used for approximating any integrand marginalisation and the primitive! when varying α.

- Use Variational Quantum Circuits to calculate multi-dimensional integrals of the form

$$I(\alpha) = \int_{x_a}^{x_b} g(\alpha; \boldsymbol{x}) \mathrm{d}^n \boldsymbol{x}. \quad (2)$$

- Algorithm's summary:

1. we train the derivative of a VQC w.r.t. the integral variables $\boldsymbol{x}$ to approximate $g(\boldsymbol{x})$;
2. we compute the derivatives using the PSR, which allows the same circuit $\mathcal{C}$ to be used for approximating any integrand marginalisation and the primitive! when varying α.

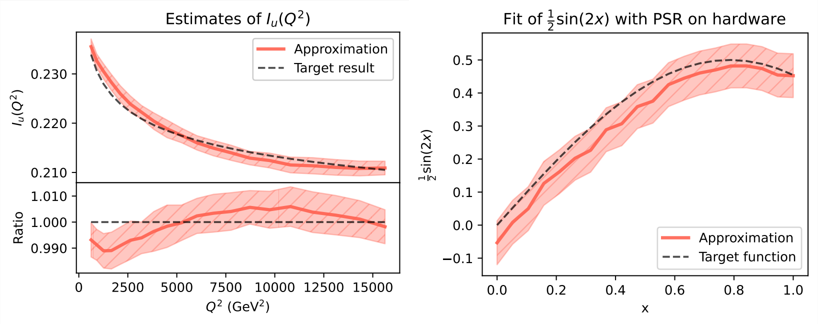


- Use Variational Quantum Circuits to calculate multi-dimensional integrals of the form

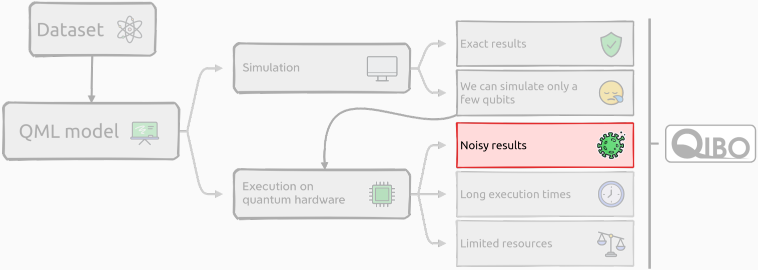
$$I(\alpha) = \int_{\boldsymbol{x}_a}^{\boldsymbol{x}_b} g(\alpha; \boldsymbol{x}) \mathrm{d}^n \boldsymbol{x}. \tag{2}$$

- Algorithm's summary:

1. we train the derivative of a VQC w.r.t. the integral variables $\boldsymbol{x}$ to approximate $g(\boldsymbol{x})$;
2. we compute the derivatives using the PSR, which allows the same circuit $\mathcal{C}$ to be used for approximating any integrand marginalisation and the primitive! when varying α.



How to deal with noise?



Real time error mitigation in QML trainings

Coming soon!

Real time error mitigation in QML trainings | Coming soon!

- Use Error Mitigation techniques to clean up the parameters space during the QML training.

Real time error mitigation in QML trainings — Coming soon!

- Use Error Mitigation techniques to clean up the parameters space during the QML training.
- Algorithm's summary:

Real time error mitigation in QML trainings

Coming soon!

- Use Error Mitigation techniques to clean up the parameters space during the QML training.
- Algorithm's summary:

1. we mitigate all the expected values E through Clifford Data Regression (CDR):

$$E_{\rm mit} = \alpha_{\rm cdr} E_{\rm noisy} + \beta_{\rm cdr}; \tag{3}$$

Real time error mitigation in QML trainings

Coming soon!

- Use Error Mitigation techniques to clean up the parameters space during the QML training.
- Algorithm's summary:

1. we mitigate all the expected values E through Clifford Data Regression (CDR):

$$E_{\rm mit} = \alpha_{\rm cdr} E_{\rm noisy} + \beta_{\rm cdr}; \tag{3}$$

2. we update $(\alpha, \beta)_{\rm cdr}$ periodically during the training in order to track the noise;

Real time error mitigation in QML trainings — Coming soon!

- Use Error Mitigation techniques to clean up the parameters space during the QML training.
- Algorithm's summary:

1. we mitigate all the expected values E through Clifford Data Regression (CDR):
$$E_{\mathrm{mit}} = \alpha_{\mathrm{cdr}} E_{\mathrm{noisy}} + \beta_{\mathrm{cdr}}; \tag{3}$$
2. we update $(\alpha, \beta)_{\mathrm{cdr}}$ periodically during the training in order to track the noise;
3. the mitigation removes the bounds and accelerates the training process.

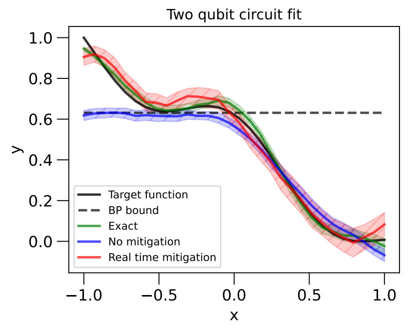
Use Error Mitigation techniques to clean up the parameters space during the QML training.

Algorithm's summary:

1. we mitigate all the expected values E through Clifford Data Regression (CDR):

$$E_{\rm mit} = \alpha_{\rm cdr} E_{\rm noisy} + \beta_{\rm cdr}; \tag{3}$$

2. we update $(\alpha, \beta)_{\rm cdr}$ periodically during the training in order to track the noise;
3. the mitigation removes the bounds and accelerates the training process.



Real time error mitigation in QML trainings

Coming soon!

- Use Error Mitigation techniques to clean up the parameters space during the QML training.
- Algorithm's summary:

1. we mitigate all the expected values E through Clifford Data Regression (CDR):

$$E_{\mathrm{mit}} = \alpha_{\mathrm{cdr}} E_{\mathrm{noisy}} + \beta_{\mathrm{cdr}}; \tag{3}$$

2. we update $(\alpha, \beta)_{\mathrm{cdr}}$ periodically during the training in order to track the noise;
3. the mitigation removes the bounds and accelerates the training process.