

Multi-dimensional integration with quantum circuits

Based on  arXiv:2308.05657

Juan Manuel Cruz-Martinez, Matteo Robbiati, Stefano Carrazza

19 October 2023



Aim and motivation

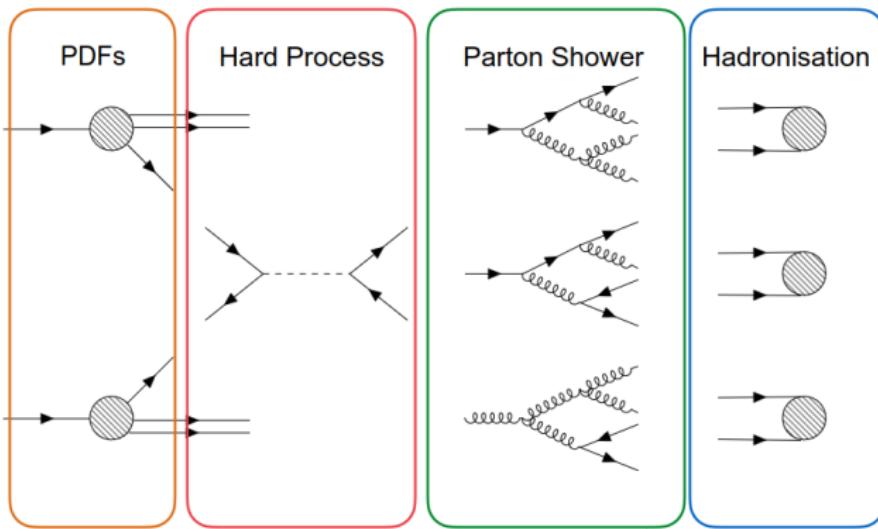
- ❖ Calculate multi-dimensional integrals like:

$$I(\alpha) = \int_{x_a}^{x_b} g(\alpha; x) d^n x. \quad (1)$$

Aim and motivation

- ❖ Calculate multi-dimensional integrals like:

$$I(\alpha) = \int_{x_a}^{x_b} g(\alpha; x) d^n x. \quad (1)$$



Introductory concepts

Machine Learning

Machine Learning helps in solving statistical problems, such as data generation, classification, regression, forecasting, etc.

Machine Learning

Machine Learning helps in solving statistical problems, such as data generation, classification, regression, forecasting, etc.

- ❖ we aim to know some hidden law between two variables: $y = f(x)$;

Machine Learning helps in solving statistical problems, such as data generation, classification, regression, forecasting, etc.

- ❖ we aim to know some hidden law between two variables: $y = f(x)$;
- 📊 we define a parameteric model with returns $y_{\text{est}} = f_{\text{est}}(x; \theta)$;

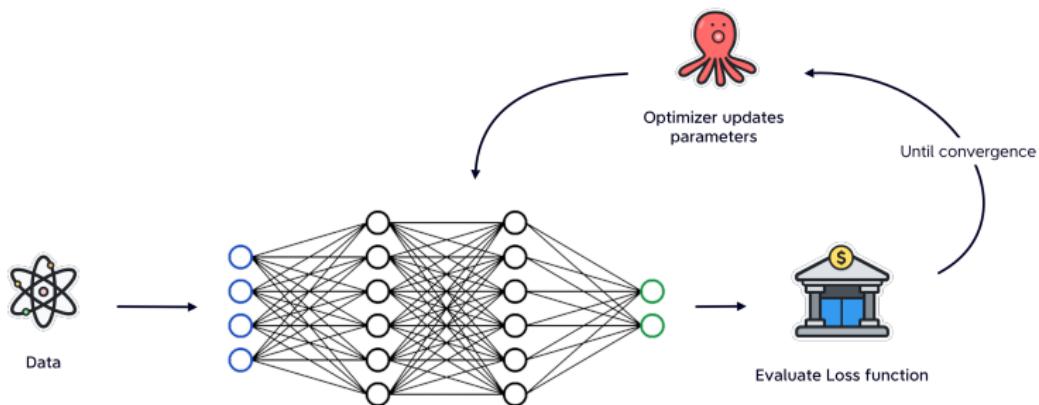
Machine Learning helps in solving statistical problems, such as data generation, classification, regression, forecasting, etc.

- ❖ we aim to know some hidden law between two variables: $\mathbf{y} = f(\mathbf{x})$;
- 📊 we define a parameteric model with returns $\mathbf{y}_{\text{est}} = f_{\text{est}}(\mathbf{x}; \boldsymbol{\theta})$;
- 🔭 we define an optimizer, which task is to compute $\operatorname{argmin}_{\boldsymbol{\theta}} [J(\mathbf{y}_{\text{meas}}, \mathbf{y}_{\text{est}})]$.

Machine Learning

Machine Learning helps in solving statistical problems, such as data generation, classification, regression, forecasting, etc.

- ❖ we aim to know some hidden law between two variables: $y = f(x)$;
- 📊 we define a parameteric model with returns $y_{\text{est}} = f_{\text{est}}(x; \theta)$;
- 🔭 we define an optimizer, which task is to compute $\operatorname{argmin}_{\theta} [J(y_{\text{meas}}, y_{\text{est}})]$.



Parametric Quantum Circuits

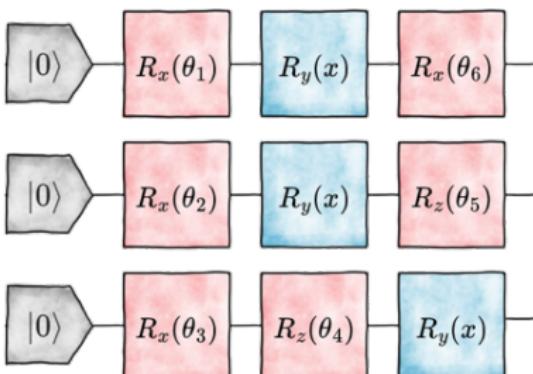
Parametric Quantum Circuits

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



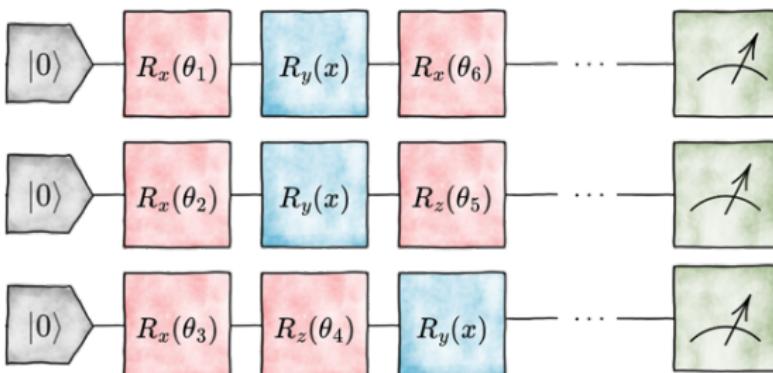
Parametric Quantum Circuits

- ✍ 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**.
Rotational gates $R_j(\theta) = e^{-i\theta\hat{\sigma}_j}$ are used to build parametric circuits $\mathcal{C}(\theta)$;



Parametric Quantum Circuits

- ✍ 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**.
Rotational gates $R_j(\theta) = e^{-i\theta\hat{\sigma}_j}$ are used to build parametric circuits $\mathcal{C}(\theta)$;
- 👁️ information is accessed calculating expected values $E[\hat{O}]$ of target observables \hat{O} on the state obtained executing \mathcal{C} .



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

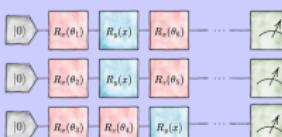
Quantum Computation

\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

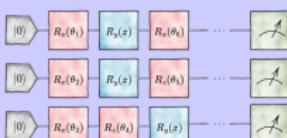
Quantum Computation

\mathcal{Q} : qubits;

\mathcal{S} : superposition;

\mathcal{E} : entanglement.

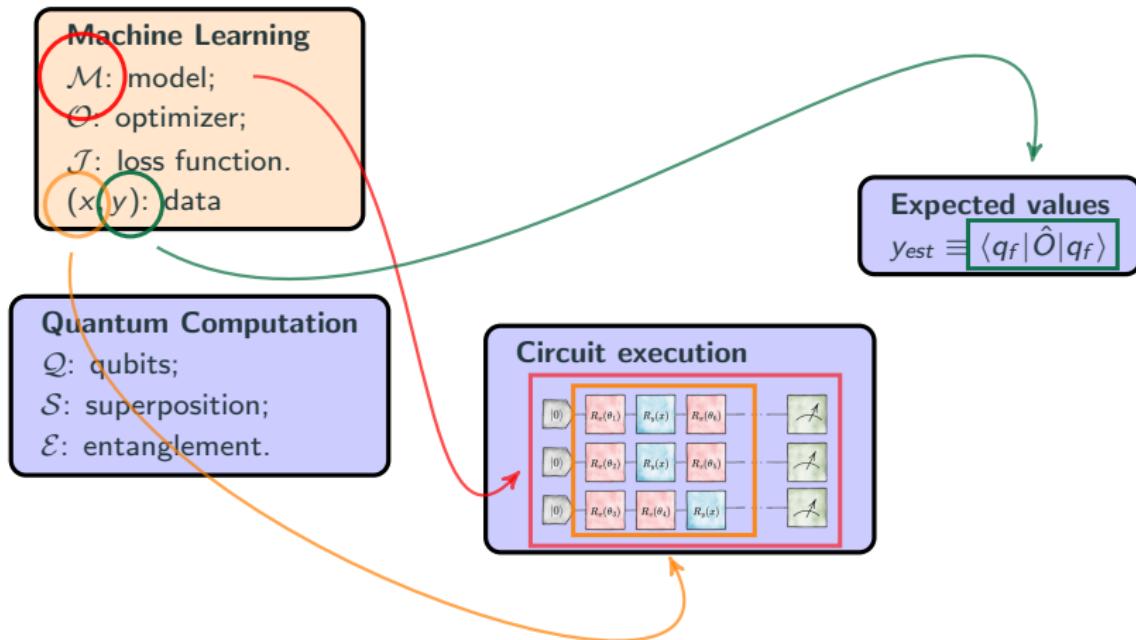
Circuit execution



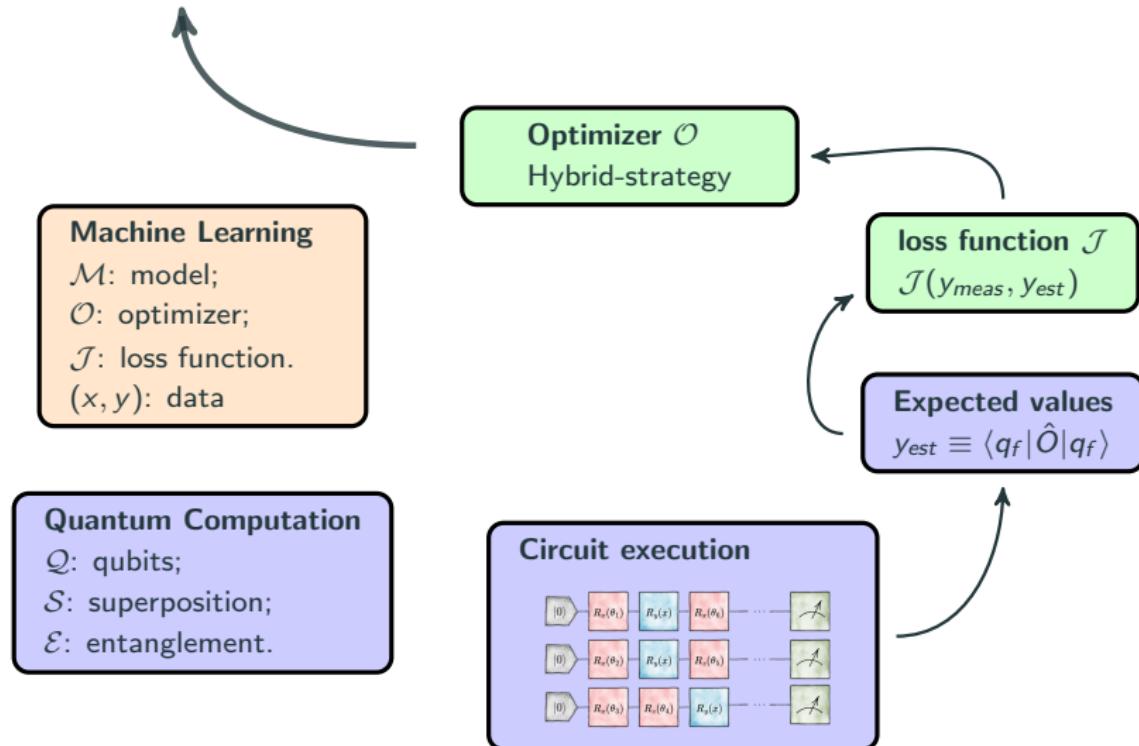
Expected values

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

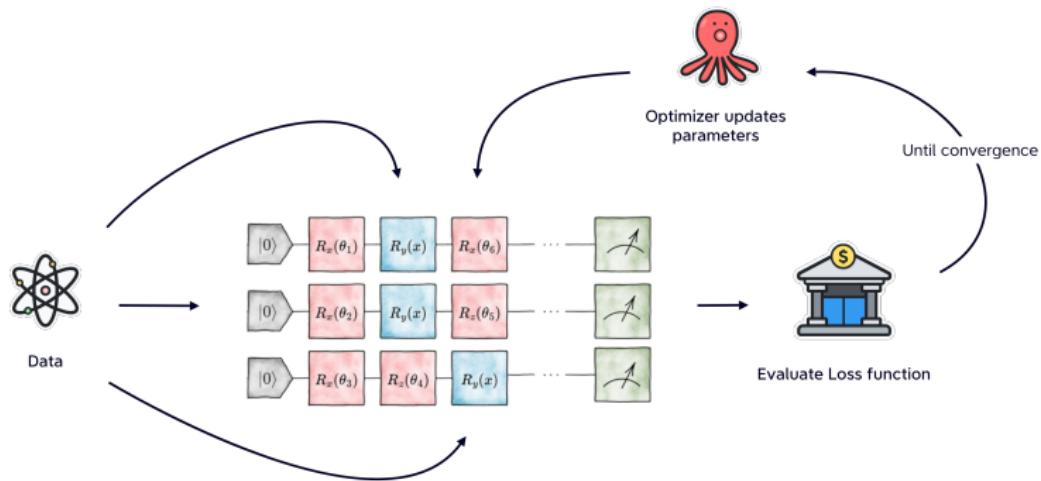
Quantum Machine Learning - encoding the problem



Quantum Machine Learning!

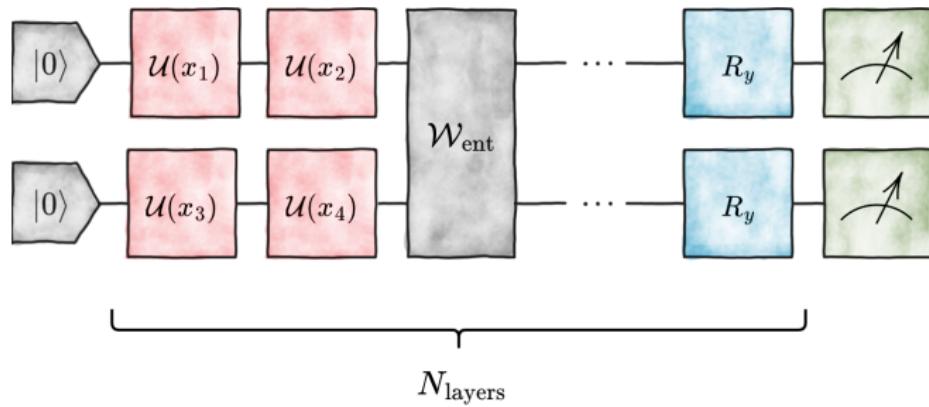


From ML to QML

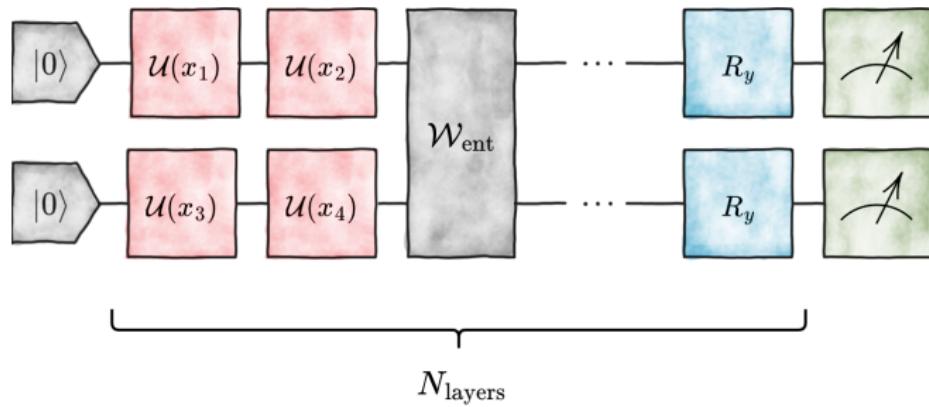


We define an uploading channel $U(x; \theta)$, and we repeat the uploading N times.

We define an uploading channel $U(x; \theta)$, and we repeat the uploading N times.



We define an uploading channel $U(x; \theta)$, and we repeat the uploading N times.

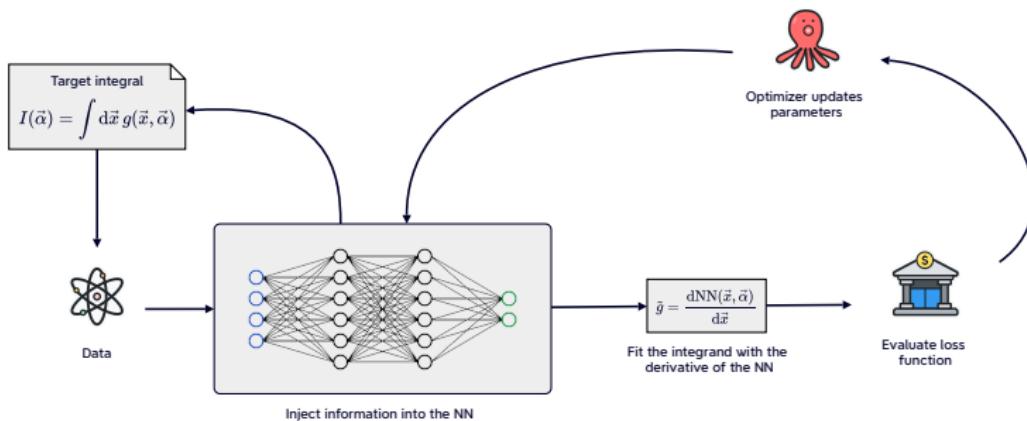


It has been proved this approach is equivalent to approximate a function with an N -term Fourier Series.

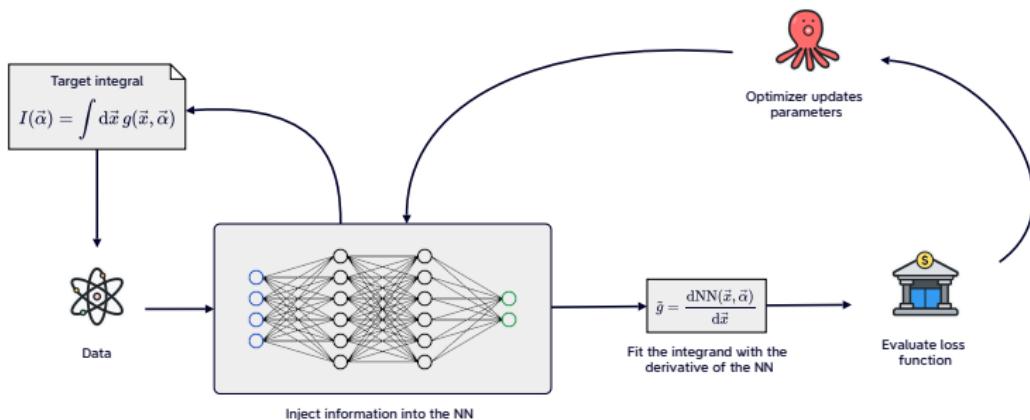
Two inspirations

In 2022, D. Maitre and R. Santos-Mateus presented a method to estimate multi-variable integrals using Neural Networks:

In 2022, D. Maitre and R. Santos-Mateus presented a method to estimate multi-variable integrals using Neural Networks:

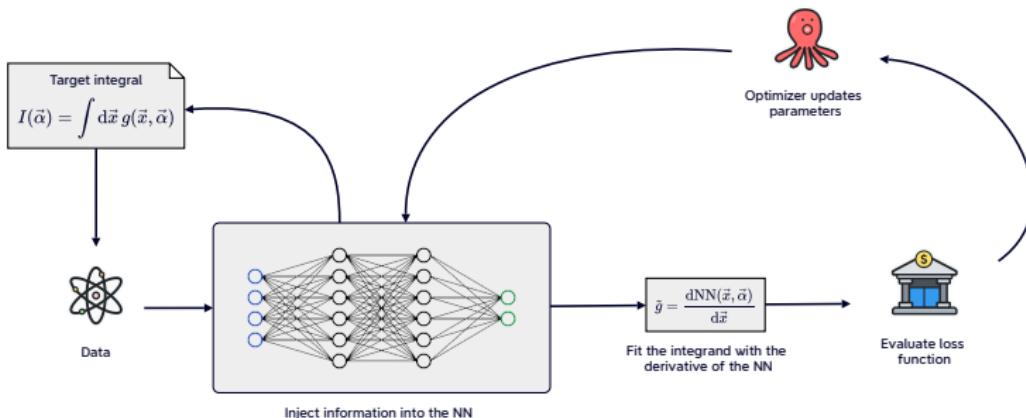


In 2022, D. Maitre and R. Santos-Mateus presented a method to estimate multi-variable integrals using Neural Networks:



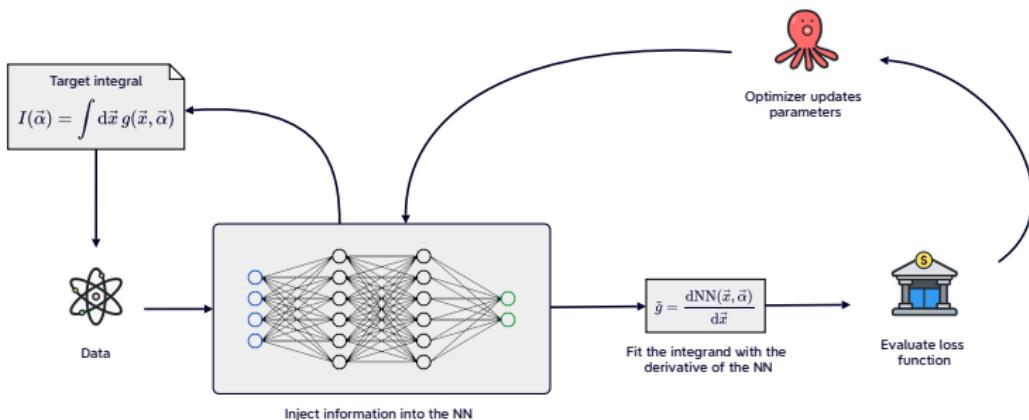
- both NN and dNN are models of the integral and integrand respectively;

In 2022, D. Maitre and R. Santos-Mateus presented a method to estimate multi-variable integrals using Neural Networks:

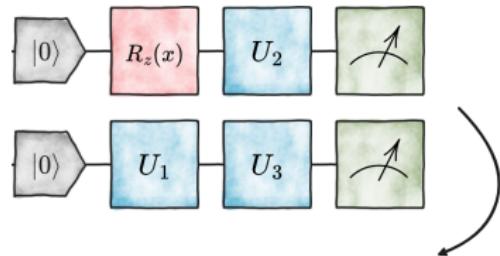


- both NN and dNN are models of the integral and integrand respectively;
- once trained, the NN can be called with any combination of data and parameters. Monte Carlo Integration (MCI), instead, has to be recomputed every time;

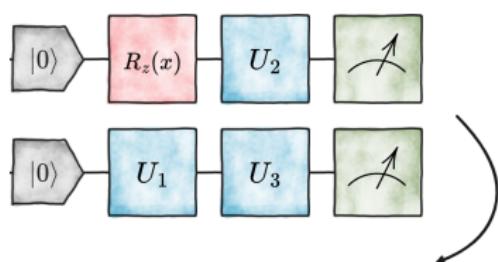
In 2022, D. Maitre and R. Santos-Mateus presented a method to estimate multi-variable integrals using Neural Networks:



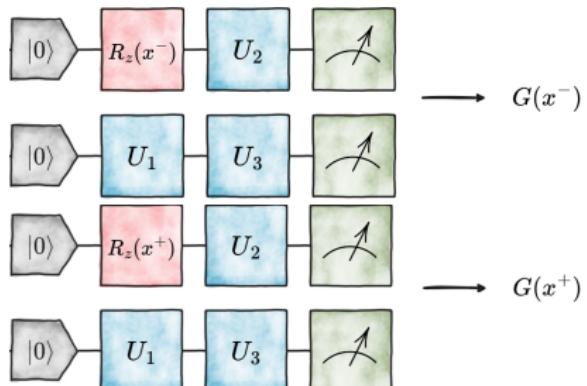
- both NN and dNN are models of the integral and integrand respectively;
- once trained, the NN can be called with any combination of data and parameters. Monte Carlo Integration (MCI), instead, has to be recomputed every time;
- in the INN is the integrand to be approximated, instead of the integral (as in MCI), swaps **variance** for approximation error.



$$G(x) = \langle 0 | \mathcal{C}^\dagger(x) \hat{O} \mathcal{C}(x) | 0 \rangle$$



$$G(x) = \langle 0 | \mathcal{C}^\dagger(x) \hat{O} \mathcal{C}(x) | 0 \rangle$$



Considering the unitary $\mathcal{U}(x) = e^{-ixU}$ affected by one parameter x , if the hermitian generator U has at most two eigenvalues $\pm r$, an exact estimator of $\partial_x G$ is:

$$\partial_x G = r [G(x^+) - G(x^-)].$$

Combining inspirations: qinntegrate

The qinntegrate algorithm

At this point, we know that:

The qinntegrate algorithm

At this point, we know that:

1. **variables** can be **injected** into a quantum circuit as rotational angles;

The qinntegrate algorithm

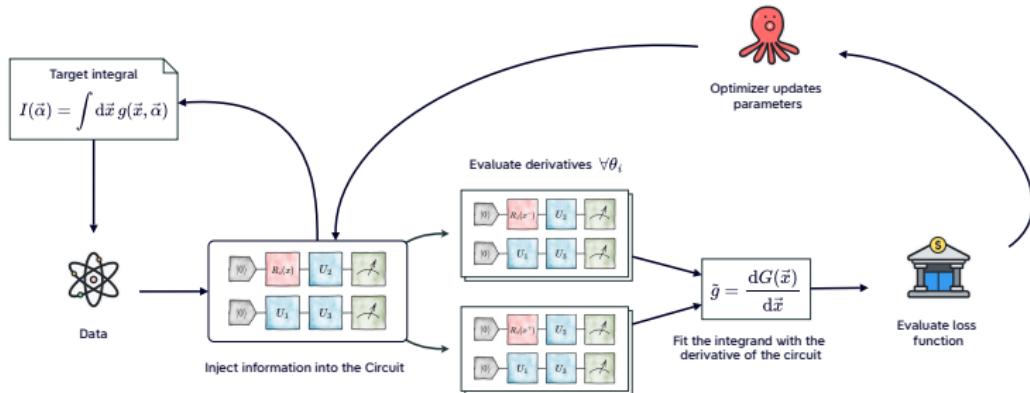
At this point, we know that:

1. **variables** can be **injected** into a quantum circuit as rotational angles;
2. the same circuit architecture \mathcal{C} can be used to compute **both** the estimator and its derivatives.

The qinntegrate algorithm

At this point, we know that:

1. **variables** can be **injected** into a quantum circuit as rotational angles;
2. the same circuit architecture \mathcal{C} can be used to compute **both** the estimator and its derivatives.



If independent variables, $\frac{dG(x)}{dx}$ is obtained by summing all PSR contributions.

Validation examples

Toy model: 3-dimensional trigonometric function

We firstly consider a simple multi-dimensional target:

$$\begin{aligned} I(\alpha, \alpha_0; \mathbf{x}) &= \int g(\alpha, \alpha_0; \mathbf{x}) d\mathbf{x} \\ &= \int \cos(\alpha \cdot \mathbf{x} + \alpha_0) d\mathbf{x}. \end{aligned} \tag{2}$$

And we target a differential distribution $\frac{dI(\alpha, \alpha_0; \mathbf{x})}{dx_i}$ for fixed i but varying α_0 's.

Toy model: 3-dimensional trigonometric function

We firstly consider a simple multi-dimensional target:

$$\begin{aligned} I(\alpha, \alpha_0; \mathbf{x}) &= \int g(\alpha, \alpha_0; \mathbf{x}) d\mathbf{x} \\ &= \int \cos(\alpha \cdot \mathbf{x} + \alpha_0) d\mathbf{x}. \end{aligned} \tag{2}$$

And we target a differential distribution $\frac{dI(\alpha, \alpha_0; \mathbf{x})}{d\alpha_i}$ for fixed i but varying α_0 's.

Parameter	Value
$N_{x,train}$	100
α	$\{1, 2, 0.5\}$
N_{α_0}	10
N_{layers}	2
N_{params}	20
$ I - \tilde{I} $	$4.4 \cdot 10^{-3}$
N_{shots}	Exact simulation
Optimizer	L-BFGS

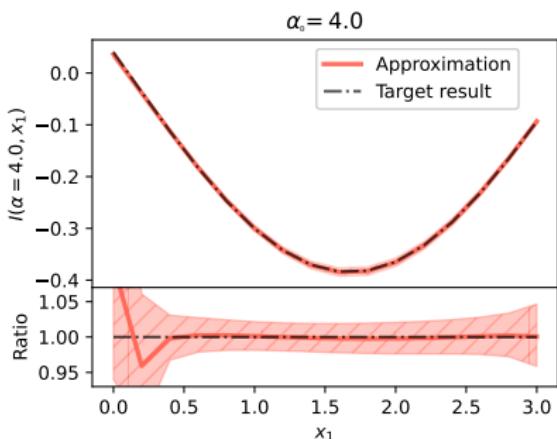
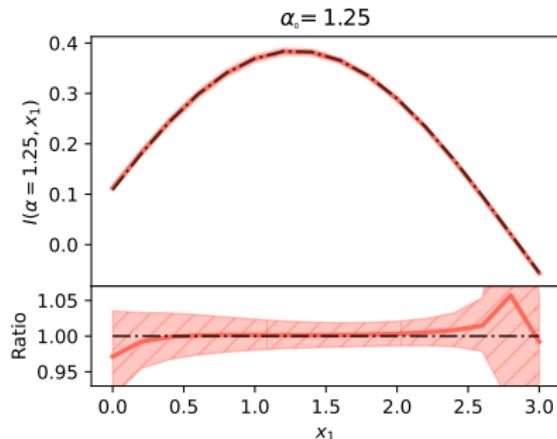
Toy model: 3-dimensional trigonometric function

We firstly consider a simple multi-dimensional target:

$$\begin{aligned} I(\alpha, \alpha_0; \mathbf{x}) &= \int g(\alpha, \alpha_0; \mathbf{x}) d\mathbf{x} \\ &= \int \cos(\alpha \cdot \mathbf{x} + \alpha_0) d\mathbf{x}. \end{aligned} \quad (2)$$

And we target a differential distribution $\frac{dI(\alpha, \alpha_0; \mathbf{x})}{dx_i}$ for fixed i but varying α_0 's.

Parameter	Value
$N_{x, \text{train}}$	100
α	$\{1, 2, 0.5\}$
N_{α_0}	10
N_{layers}	2
N_{params}	20
$ I - \tilde{I} $	$4.4 \cdot 10^{-3}$
N_{shots}	Exact simulation
Optimizer	L-BFGS



The u -quark PDF

We then consider the u -quark PDF:

$$I_u(Q^2) = \int_{10^{-4}}^{0.7} x u(x, Q) dx. \quad (3)$$

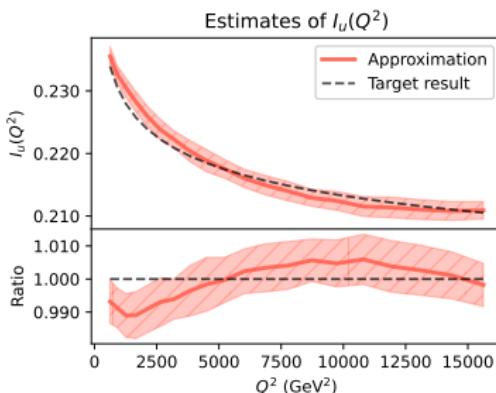
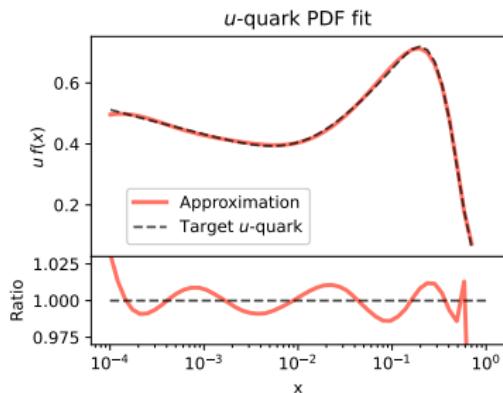
The u -quark PDF

We then consider the u -quark PDF:

$$I_u(Q^2) = \int_{10^{-4}}^{0.7} x u(x, Q) dx. \quad (3)$$

Parameter	Value
$N_{x,\text{train}}$	500
Q	1.67
N_{layers}	4
N_{params}	27
$ I - \tilde{I} $	$1.2 \cdot 10^{-5}$
N_{shots}	Exact simulation
Optimizer	L-BFGS

Parameter	Value
$(N_x, N_Q)_{\text{train}}$	(120, 100)
$N_{Q,\text{est}}$	20
N_{runs}	100
N_{layers}	4
N_{params}	36
$ I - \tilde{I} $	$7.4 \cdot 10^{-5}$
N_{shots}	10^6
Optimizer	L-BFGS



Toy model on a superconducting quantum chip

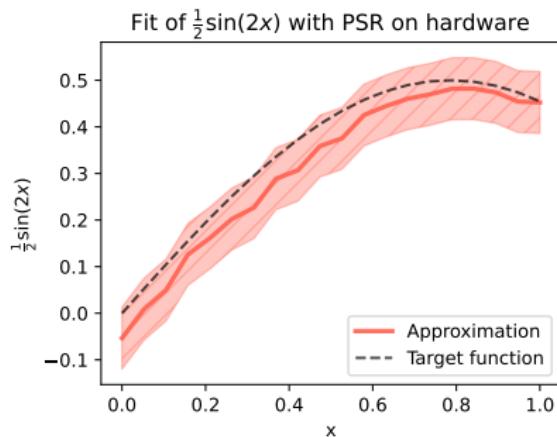
We finally tackle a dummy target using a real superconducting qubit:

$$I = \int_0^1 \frac{1}{2} \cos(2x) dx. \quad (4)$$

Toy model on a superconducting quantum chip

We finally tackle a dummy target using a real superconducting qubit:

$$I = \int_0^1 \frac{1}{2} \cos(2x) dx. \quad (4)$$



Parameter	Value
$N_{x,\text{train}}$	50
$N_{x,\text{est}}$	20
N_{runs}	10
N_{layers}	1
N_{params}	6
$ I - \tilde{I} $	$2.8 \cdot 10^{-2}$
N_{shots}	$2 \cdot 10^3$
Optimizer	L-BFGS

Conclusions

Conclusions and outlook

Some final comments

- ❑ the algorithm **successfully approximates** multi-dimensional integrals;
- ❑ once a winning architecture is defined, this can be used to fit the integrand and to perform **any differential distribution** of the integral;
- ❑ we have used **shallow models** as VQCs;
- ❑ computing the derivatives scales **exponentially** with the dimensionality;
- ❑ the hardware **noise** complicates the derivatives calculation.

Conclusions and outlook

Some final comments

- ◀ the algorithm **successfully approximates** multi-dimensional integrals;
- ◀ once a winning architecture is defined, this can be used to fit the integrand and to perform **any differential distribution** of the integral;
- ◀ we have used **shallow models** as VQCs;
- ◀ computing the derivatives scales **exponentially** with the dimensionality;
- ◀ the hardware **noise** complicates the derivatives calculation.

How to do better?

- ↗ change **differentiation strategies** (natural gradient, non-demolition, etc);
- ↗ define models that are **even more shallow** to reduce the number of gates;
- ↗ exploit **variables correlation** to reduce the number of required gates.

Conclusions and outlook

Some final comments

- ◀ the algorithm **successfully approximates** multi-dimensional integrals;
- ◀ once a winning architecture is defined, this can be used to fit the integrand and to perform **any differential distribution** of the integral;
- ◀ we have used **shallow models** as VQCs;
- ◀ computing the derivatives scales **exponentially** with the dimensionality;
- ◀ the hardware **noise** complicates the derivatives calculation.

How to do better?

- ↗ change **differentiation strategies** (natural gradient, non-demolition, etc);
- ↗ define models that are **even more shallow** to reduce the number of gates;
- ↗ exploit **variables correlation** to reduce the number of required gates.

⌚ Code available [here](#).

📄 Pre-print: arXiv:2308.05657

Thank you for your attention!