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

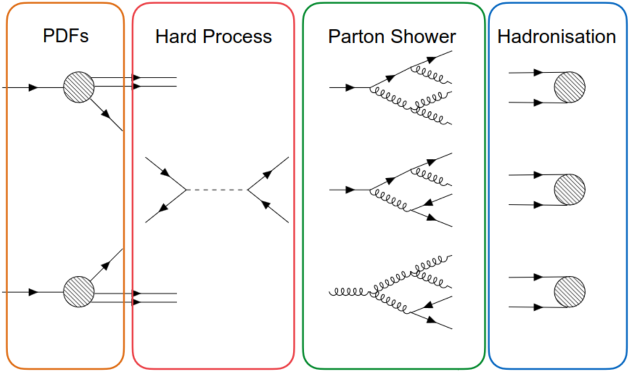
- Calculate multi-dimensional integrals like:

$$I(\alpha) = \int_{\boldsymbol{x}_a}^{\boldsymbol{x}_b} g(\alpha; \boldsymbol{x}) \mathrm{d}^n \boldsymbol{x}. \tag{1}$$

Aim and motivation

- Calculate multi-dimensional integrals like:

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



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: $\boldsymbol{y} = f(\boldsymbol{x})$;

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: $\boldsymbol{y} = f(\boldsymbol{x})$;
- we define a parameteric model with returns $\boldsymbol{y}_{\text{est}} = f_{\text{est}}(\boldsymbol{x}; \boldsymbol{\theta})$;

Machine Learning

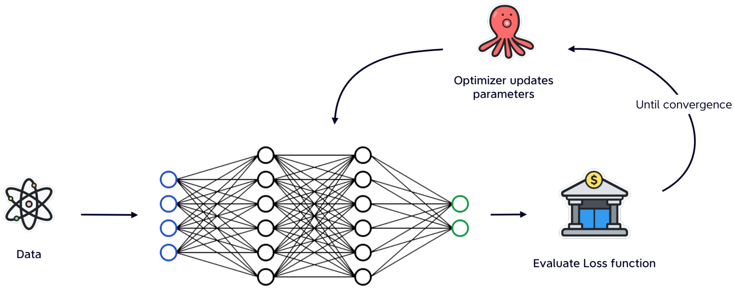
Machine Learning helps in solving statistical problems, such as data generation, classification, regression, forecasting, etc.

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

Machine Learning

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

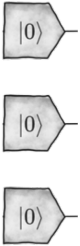
- we aim to know some hidden law between two variables: $\boldsymbol{y} = f(\boldsymbol{x})$;
- we define a parameteric model with returns $\boldsymbol{y}_{\text{est}} = f_{\text{est}}(\boldsymbol{x}; \boldsymbol{\theta})$;
- we define an optimizer, which task is to compute $\operatorname{argmin}_{\boldsymbol{\theta}}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



Parametric Quantum Circuits

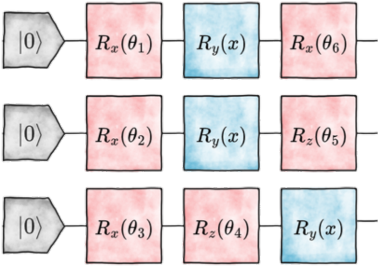
Parametric Quantum Circuits

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



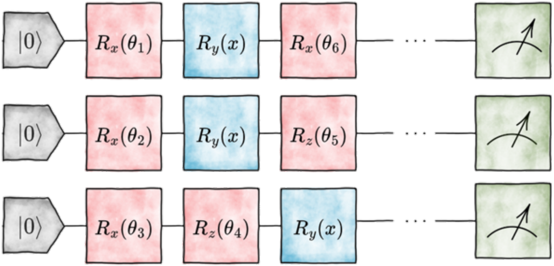
Parametric Quantum Circuits

- 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 $\mathcal{C}(\boldsymbol{\theta})$;



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 $\mathcal{C}(\boldsymbol{\theta})$;
- information is accessed calculating expected values $\mathrm{E}[\hat{O}]$ 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

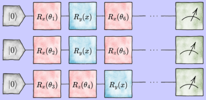
Quantum Computation
$\mathcal{Q}$: qubits;
$\mathcal{S}$: superposition;
$\mathcal{E}$: entanglement.

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



Machine Learning

$\mathcal{M}$: model;

$\mathcal{O}$: optimizer;

$\mathcal{J}$: loss function.

(x, y): data

Expected values

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

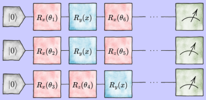
Quantum Computation

$\mathcal{Q}$: qubits;

$\mathcal{S}$: superposition;

$\mathcal{E}$: entanglement.

Circuit execution



Quantum Machine Learning - encoding the problem

Machine Learning

$\mathcal{M}$: model;

$\mathcal{O}$: optimizer;

$\mathcal{J}$: loss function.

(x, y): data

Expected values

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

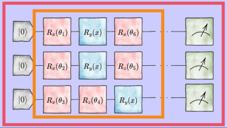
Quantum Computation

$\mathcal{Q}$: qubits;

$\mathcal{S}$: superposition;

$\mathcal{E}$: entanglement.

Circuit execution



Quantum Machine Learning!

Optimizer $\mathcal{O}$
Hybrid-strategy

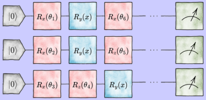
Machine Learning
$\mathcal{M}$: model;
$\mathcal{O}$: optimizer;
$\mathcal{J}$: loss function.
(x, y): data

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

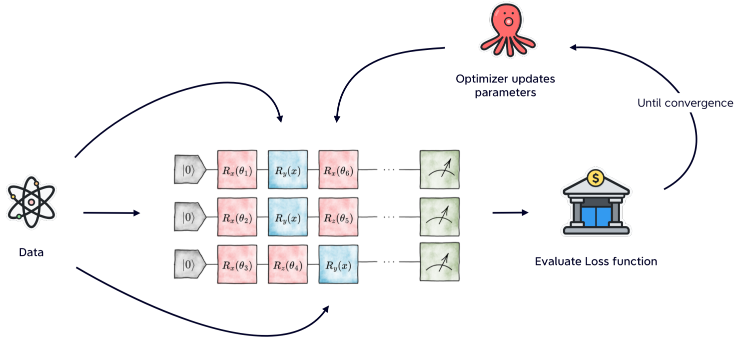
Expected values
$y_{est} \equiv \langle q_f | \hat{O} | q_f \rangle$

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

Circuit execution

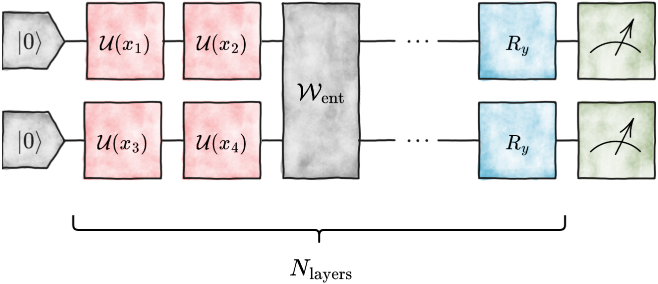


From ML to QML

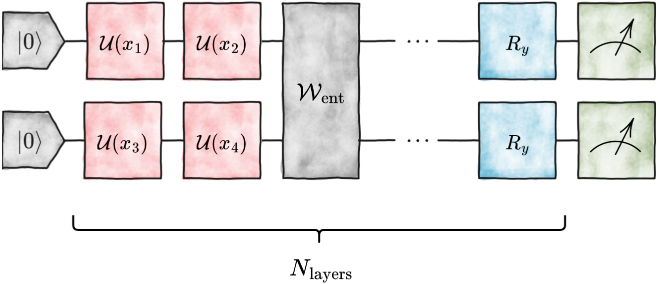


We define an uploading channel $U(x;\boldsymbol{\theta})$, and we repeat the uploading N times.

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



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

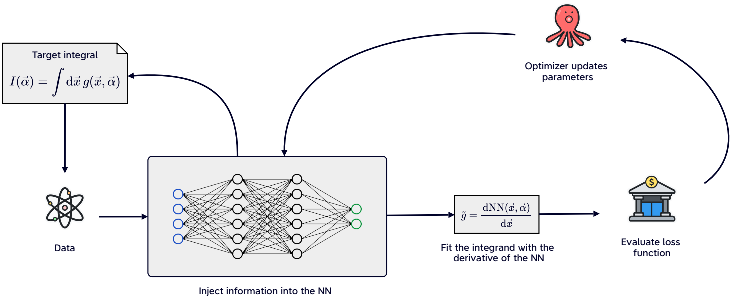


It has been proved this approach is equivalent to approximate a function with an N-term Fourier Series.

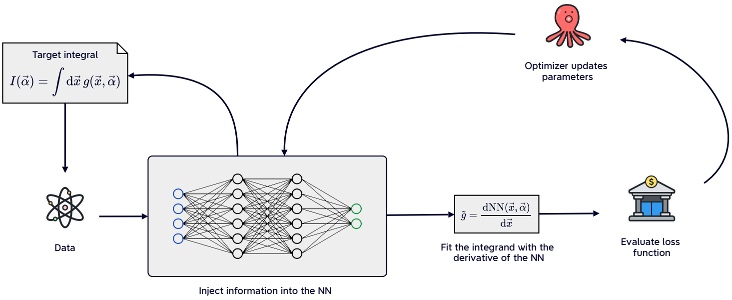
Two inspirations

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:

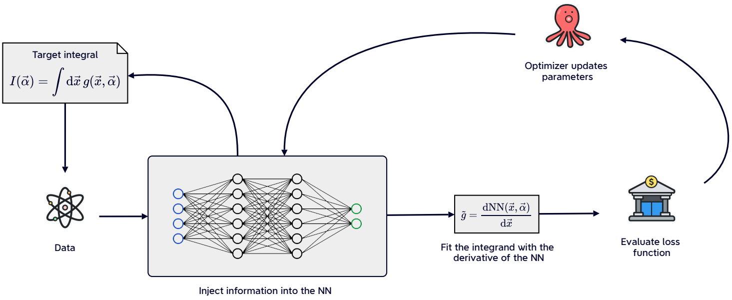


In 2022, D. Maitre and R. Santos-Mateus presented a method to estimate multi-variable integrals using Neural Networks:



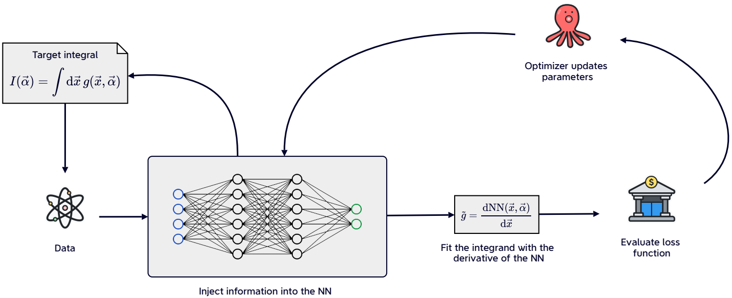
- both NN and dNN are models of the integral and integrand respectively;

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

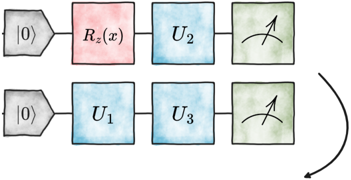


- both NN and dNN are models of the integral and integrand respectively;
- once trained, the NN can be called with any combination of data and parameters. Monte Carlo Integration (MCI), instead, has to be recomputed every time;

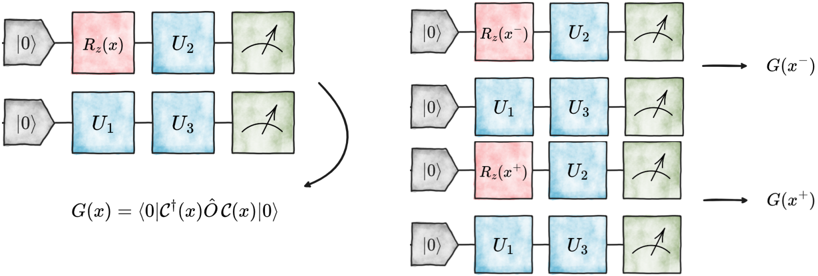
In 2022, D. Maitre and R. Santos-Mateus presented a method to estimate multi-variable integrals using Neural Networks:



- 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.



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



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:

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

Combining inspirations: `qinntegrate`

At this point, we know that:

At this point, we know that:

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

The `qinntegrate` algorithm

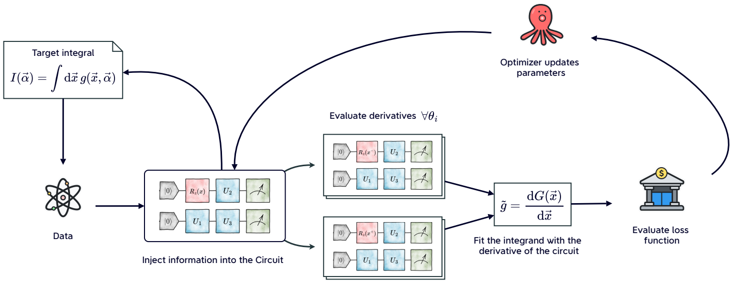
At this point, we know that:

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, $\frac{\mathrm{d}G(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}}$ is obtained by summing all PSR contributions.

Validation examples

Toy model: 3-dimensional trigonometric function

We firstly consider a simple multi-dimensional target:

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

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

Toy model: 3-dimensional trigonometric function

We firstly consider a simple multi-dimensional target:

$$I(\boldsymbol{\alpha}, \alpha_0; \boldsymbol{x}) = \int g(\boldsymbol{\alpha}, \alpha_0; \boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \\ = \int \cos(\boldsymbol{\alpha} \cdot \boldsymbol{x} + \alpha_0)\, \mathrm{d}\boldsymbol{x}. \tag{2}$$

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

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

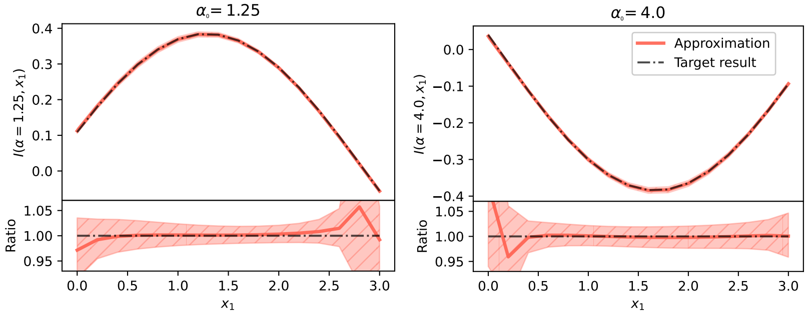
Toy model: 3-dimensional trigonometric function

We firstly consider a simple multi-dimensional target:

$$I(\boldsymbol{\alpha}, \alpha_0; \boldsymbol{x}) = \int g(\boldsymbol{\alpha}, \alpha_0; \boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \\ = \int \cos(\boldsymbol{\alpha} \cdot \boldsymbol{x} + \alpha_0)\, \mathrm{d}\boldsymbol{x}. \tag{2}$$

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

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



The u-quark PDF

We then consider the u-quark PDF:

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

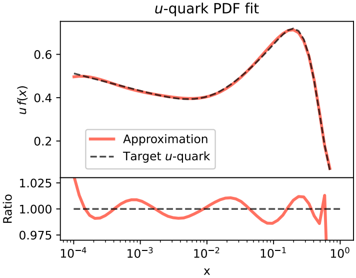
The u-quark PDF

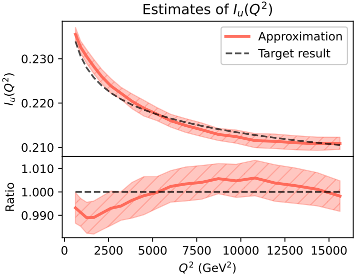
We then consider the u-quark PDF:

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

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

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





Toy model on a superconducting quantum chip

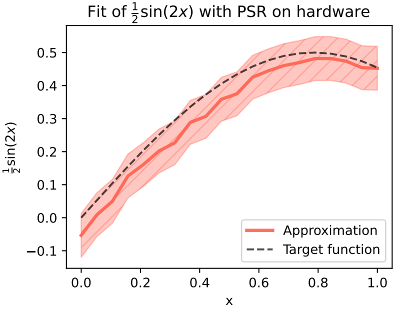
We finally tackle a dummy target using a real superconducting qubit:

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

Toy model on a superconducting quantum chip

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

$$I = \int_0^1 \frac{1}{2}\cos(2x)\,\mathrm{d}x. \tag{4}$$



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

Conclusions

Conclusions and outlook

As final comment we can say that:

Conclusions and outlook

As final comment we can say that:

- 👍 the proposed model **successfully approximates** multi-dimensional integrals;

Conclusions and outlook

As final comment we can say that:

- the proposed model **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;

Conclusions and outlook

As final comment we can say that:

- the proposed model **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;

Conclusions and outlook

As final comment we can say that:

- 👍 the proposed model **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 problem dimensionality;

Conclusions and outlook

As final comment we can say that:

- 👍 the proposed model **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 problem dimensionality;
- 👎 the hardware **noise** complicates the derivatives calculation.

Conclusions and outlook

As final comment we can say that:

- 👍 the proposed model **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 problem dimensionality;
- 👎 the hardware **noise** complicates the derivatives calculation.

How to do better?

Conclusions and outlook

As final comment we can say that:

- 👍 the proposed model **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 problem dimensionality;
- 👎 the hardware **noise** complicates the derivatives calculation.

How to do better?

- try **different differentiation strategies** (natural gradient, non-demolition measurements, etc);

Conclusions and outlook

As final comment we can say that:

- 👍 the proposed model **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 problem dimensionality;
- 👎 the hardware **noise** complicates the derivatives calculation.

How to do better?

- try **different differentiation strategies** (natural gradient, non-demolition measurements, etc);
- define models that are **even more shallow** to reduce the number of quantum gates;

Conclusions and outlook

As final comment we can say that:

- 👍 the proposed model **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 problem dimensionality;
- 👎 the hardware **noise** complicates the derivatives calculation.

How to do better?

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

Thank you for your attention!