A line search strategy for training CV variational quantum circuits

GPU Day 2025

Zoltán Kolarovszki

HUN-REN Wigner Research Centre for Physics

Eötvös Loránd University

May 22, 2025

Parameter shift rules

Continuous-variable quantum computing

Parameter shift rules for CV quantum circuits?

Line search strategy

Parameter shift rules

Variational quantum circuits

Variational quantum circuits (VQCs) constist of:

1. An initial state $|\psi(\boldsymbol{x})\rangle$ depending on some data $\boldsymbol{x}$.

Variational quantum circuits

Variational quantum circuits (VQCs) constist of:

1. An initial state $|\psi(\boldsymbol{x})\rangle$ depending on some data $\boldsymbol{x}$.
2. A quantum circuit $\hat{U}(\boldsymbol{\theta})$ depending on free parameters $\boldsymbol{\theta}$.

Variational quantum circuits

Variational quantum circuits (VQCs) constist of:

1. An initial state $|\psi(\boldsymbol{x})\rangle$ depending on some data $\boldsymbol{x}$.
2. A quantum circuit $\hat{U}(\boldsymbol{\theta})$ depending on free parameters $\boldsymbol{\theta}$.
3. A set of observables $\{\hat{O}_j\}_{j=1}^{N}$ to be measured.

Variational quantum circuits

Variational quantum circuits (VQCs) constist of:

1. An initial state $|\psi(\boldsymbol{x})\rangle$ depending on some data $\boldsymbol{x}$.
2. A quantum circuit $\hat{U}(\boldsymbol{\theta})$ depending on free parameters $\boldsymbol{\theta}$.
3. A set of observables $\{\hat{O}_j\}_{j=1}^{N}$ to be measured.

Expectation values:

$$f_j(\boldsymbol{\theta}) := \langle\psi(\boldsymbol{x})|\,\hat{U}^\dagger(\boldsymbol{\theta})\hat{O}_j\hat{U}(\boldsymbol{\theta})\,|\psi(\boldsymbol{x})\rangle\,. \tag{1}$$

Variational quantum circuits

Variational quantum circuits (VQCs) constist of:

1. An initial state $|\psi(\boldsymbol{x})\rangle$ depending on some data $\boldsymbol{x}$.
2. A quantum circuit $\hat{U}(\boldsymbol{\theta})$ depending on free parameters $\boldsymbol{\theta}$.
3. A set of observables $\{\hat{O}_j\}_{j=1}^{N}$ to be measured.

Expectation values:

$$f_j(\boldsymbol{\theta}) := \langle\psi(\boldsymbol{x})|\, \hat{U}^\dagger(\boldsymbol{\theta})\hat{O}_j\hat{U}(\boldsymbol{\theta})\,|\psi(\boldsymbol{x})\rangle\,. \tag{1}$$

Loss function $\mathcal{L}$:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(f_1(\boldsymbol{\theta}), \cdots, f_N(\boldsymbol{\theta})) \tag{2}$$

Variational quantum circuits

Variational quantum circuits (VQCs) constist of:

1. An initial state $|\psi(\boldsymbol{x})\rangle$ depending on some data $\boldsymbol{x}$.
2. A quantum circuit $\hat{U}(\boldsymbol{\theta})$ depending on free parameters $\boldsymbol{\theta}$.
3. A set of observables $\{\hat{O}_j\}_{j=1}^{N}$ to be measured.

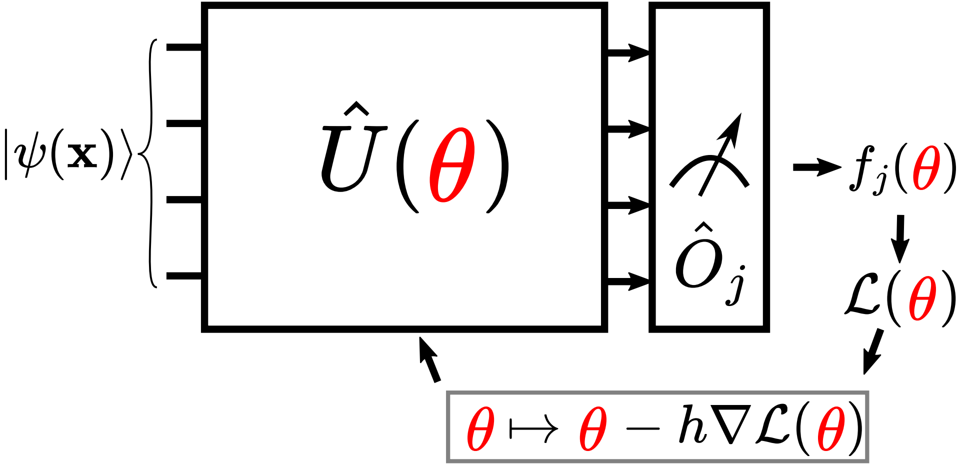
Expectation values:

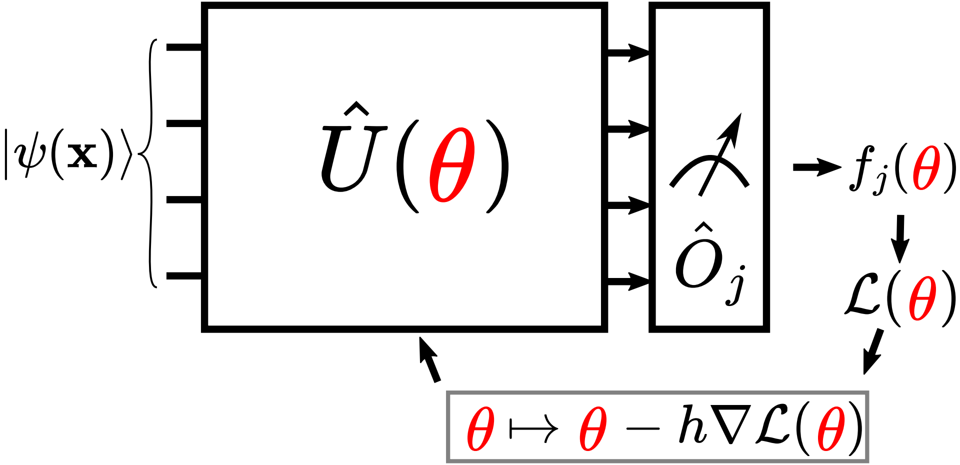
$$f_j(\boldsymbol{\theta}) := \langle\psi(\boldsymbol{x})|\, \hat{U}^{\dagger}(\boldsymbol{\theta})\hat{O}_j\hat{U}(\boldsymbol{\theta})\, |\psi(\boldsymbol{x})\rangle\,. \tag{1}$$

Loss function $\mathcal{L}$:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(f_1(\boldsymbol{\theta}), \cdots, f_N(\boldsymbol{\theta})) \tag{2}$$

Goal: minimize $\mathcal{L}$ by tuning $\boldsymbol{\theta}$.


$|\psi(\mathbf{x})\rangle$
$\hat{U}(\theta)$
$\hat{O}_j$
$f_j(\theta)$
$\mathcal{L}(\theta)$
$\theta \mapsto \theta - h\nabla\mathcal{L}(\theta)$



Question: How to determine $\nabla\mathcal{L}(\boldsymbol{\theta})$ efficiently?

How to determine gradients on a contentional computer?

Options:

How to determine gradients on a contentional computer?

Options:

1. **Symbolic differentiation:** using symboling computer algebra package

How to determine gradients on a contentional computer?

Options:

1. **Symbolic differentiation:** using symboling computer algebra package
2. **Automatic differentiation:** Builds computational graph, accumulates intermediate derivative, uses the chain rule. (Very efficient)

How to determine gradients on a contentional computer?

Options:

1. **Symbolic differentiation:** using symboling computer algebra package
2. **Automatic differentiation:** Builds computational graph, accumulates intermediate derivative, uses the chain rule. (Very efficient)
3. **Numerical differentiation:**

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) \approx \frac{f(x + \Delta x/2) - f(x - \Delta x/2)}{\Delta x}. \tag{3}$$

How to determine gradients on a contentional computer?

Options:

1. **Symbolic differentiation:** using symboling computer algebra package
2. **Automatic differentiation:** Builds computational graph, accumulates intermediate derivative, uses the chain rule. (Very efficient)
3. **Numerical differentiation:**

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) \approx \frac{f(x + \Delta x/2) - f(x - \Delta x/2)}{\Delta x}. \tag{3}$$

Are these applicable to quantum circuits?

How to determine gradient on a quantum computer?

Options:

How to determine gradient on a quantum computer?

Options:

1. **Symbolic differentiation:** symbols cannot be handled ✗

How to determine gradient on a quantum computer?

Options:

1. **Symbolic differentiation:** symbols cannot be handled ✗
2. **Automatic differentiation:** intermediate derivatives cannot be stored ✗

How to determine gradient on a quantum computer?

Options:

1. **Symbolic differentiation:** symbols cannot be handled ✗
2. **Automatic differentiation:** intermediate derivatives cannot be stored ✗
3. **Numerical differentiation:** parameters can be shifted with a small amount ✓

$$\partial_i \mathcal{L}(\boldsymbol{\theta}) \approx \frac{\mathcal{L}(\boldsymbol{\theta} + (\Delta x/2)\boldsymbol{e}_i) - \mathcal{L}(\boldsymbol{\theta} - (\Delta x/2)\boldsymbol{e}_i)}{\Delta x}. \tag{4}$$

How to determine gradient on a quantum computer?

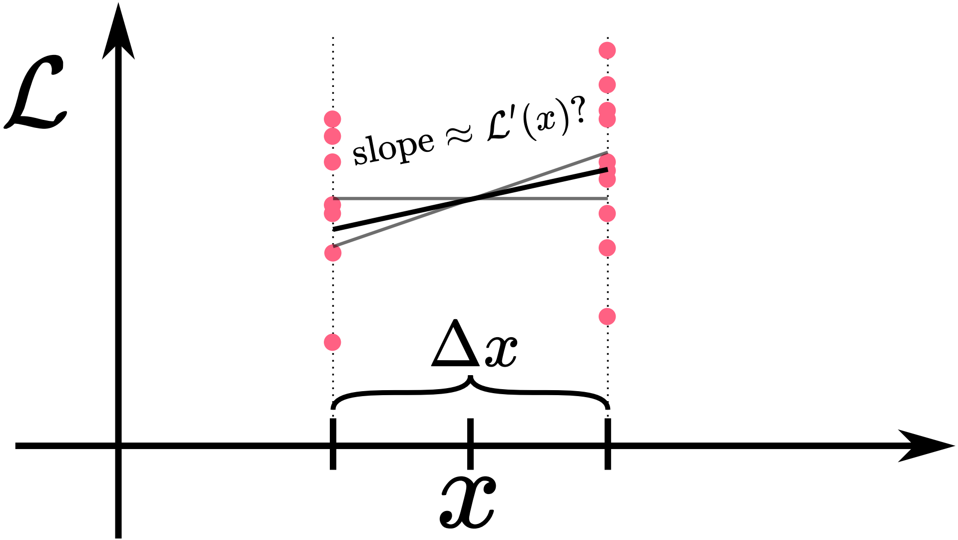
Options:

1. **Symbolic differentiation:** symbols cannot be handled ✗
2. **Automatic differentiation:** intermediate derivatives cannot be stored ✗
3. **Numerical differentiation:** parameters can be shifted with a small amount ✓

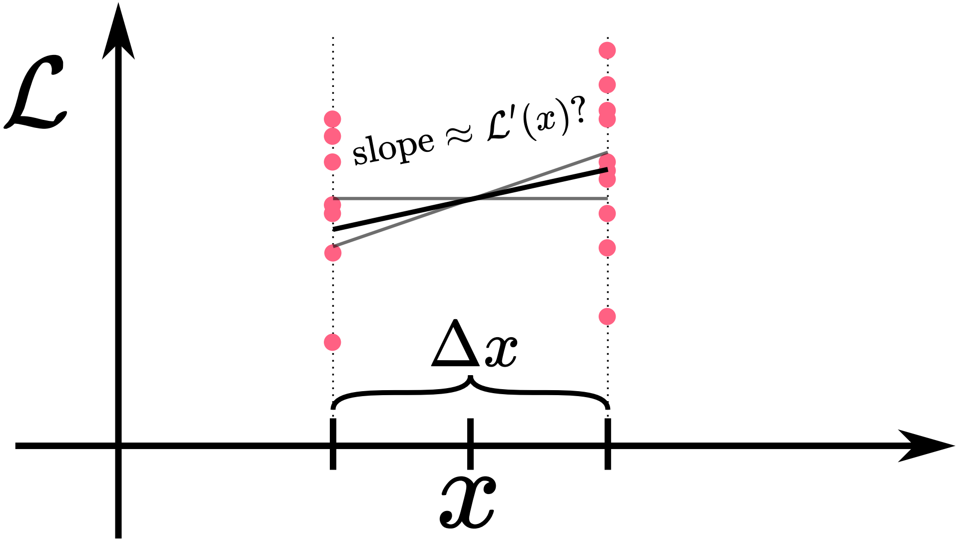
$$\partial_i \mathcal{L}(\boldsymbol{\theta}) \approx \frac{\mathcal{L}(\boldsymbol{\theta} + (\Delta x/2)\boldsymbol{e}_i) - \mathcal{L}(\boldsymbol{\theta} - (\Delta x/2)\boldsymbol{e}_i)}{\Delta x}. \tag{4}$$

Problems:

1. Near-term quantum devices are **noisy**.
2. The output is **stochastic** $\implies$ we can only estimate the expectation values from samples.



$\mathcal{L}$
slope $\approx \mathcal{L}'(x)$?
Δx
x



High errors of near-term quantum devices can make using finite difference formulas **inefficient**.

Parameter shift rules

Parameter shift rules help us to estimate gradients better.

$$\partial_i f(\boldsymbol{\theta}) = c\left[f(\boldsymbol{\theta} + s\,\boldsymbol{e}_i) - f(\boldsymbol{\theta} - s\,\boldsymbol{e}_i)\right], \tag{5}$$

where

- c is some constant,
- s is the parameter shift, **can be large**.

Parameter shift rules

Parameter shift rules help us to estimate gradients better.

$$\partial_i f(\boldsymbol{\theta}) = c\left[f(\boldsymbol{\theta} + s\,\boldsymbol{e}_i) - f(\boldsymbol{\theta} - s\,\boldsymbol{e}_i)\right], \tag{5}$$

where

- c is some constant,
- s is the parameter shift, **can be large**.

Point: The shifts are larger, but the formula is still **exact**, and improves sample-efficiency.

Parameter shift rules

Parameter shift rules help us to estimate gradients better.

$$\partial_i f(\boldsymbol{\theta}) = c\,[f(\boldsymbol{\theta} + s\,\mathbf{e}_i) - f(\boldsymbol{\theta} - s\,\mathbf{e}_i)]\,, \tag{5}$$

where

- c is some constant,
- s is the parameter shift, **can be large**.

Point: The shifts are larger, but the formula is still **exact**, and improves sample-efficiency.

Rough analogy: For

$$f(x) = \sin(x) \tag{6}$$

we can write

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) = \frac{1}{2}\,[\sin(x + \pi/2) + \sin(x - \pi/2)]\,. \tag{7}$$

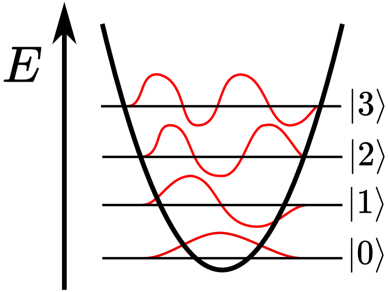
Continuous-variable quantum computing

Qubit-based vs. Continuous-variable quantum computation

	Qubit-based	Continuous-variable (CV)
Information unit	Qubit	Qumode
Hilbert space dimension	Finite	Infinite
Basis states	$\|0\rangle, \|1\rangle$	$\|0\rangle, \|1\rangle, \|2\rangle, \|3\rangle, \ldots$
Elementary gates	Hadamard, CNOT, Pauli gates	Squeezing, Rotation, Displacement
Typical measurements	Computational/Hadamard basis measurements	Particle number detection Homodyne/heterodyne detection

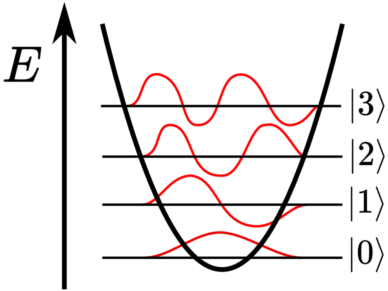
CV quantum computing

We model qumodes by quantum harmonic oscillators, and the states $|0\rangle$, $|1\rangle$, $|2\rangle$, $|3\rangle, \ldots$ correspond to excitations (**particles**).



CV quantum computing

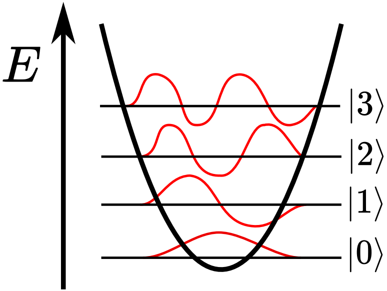
We model qumodes by quantum harmonic oscillators, and the states $|0\rangle$, $|1\rangle$, $|2\rangle$, $|3\rangle, \ldots$ correspond to excitations (**particles**).



In a sense, qubit-based quantum computation is “digital”, while CV quantum computation is “analog”.

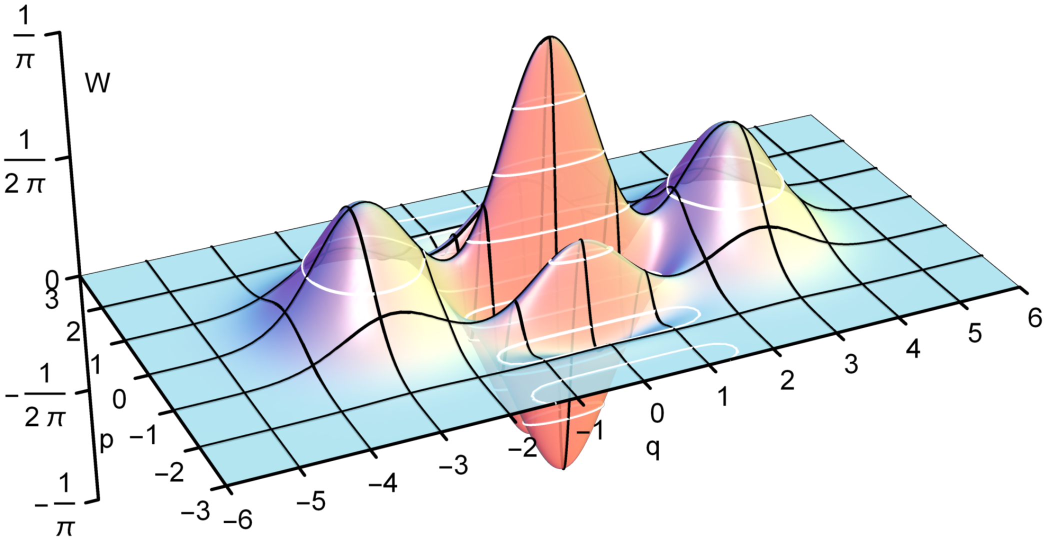
CV quantum computing

We model qumodes by quantum harmonic oscillators, and the states $|0\rangle$, $|1\rangle$, $|2\rangle$, $|3\rangle, \ldots$ correspond to excitations (**particles**).



In a sense, qubit-based quantum computation is "digital", while CV quantum computation is "analog".

CV quantum states can also be described by quasidistributions over the phase space.



$\frac{1}{\pi}$
W
$\frac{1}{2\pi}$
0
$-\frac{1}{2\pi}$
$-\frac{1}{\pi}$
p
q
3
2
1
0
−1
−2
−3
−6
−5
−4
−3
−2
−1
0
1
2
3
4
5
6

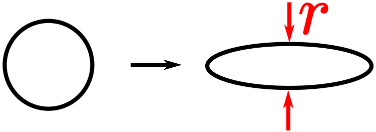
Linear gates

Roughly speaking, linear gates only **squeeze**, **rotate** and **displace** these distributions.

Linear gates

Roughly speaking, linear gates only **squeeze**, **rotate** and **displace** these distributions.

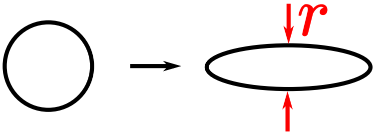
- **Squeezing** $S(r)$:



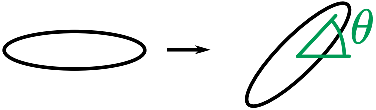
Linear gates

Roughly speaking, linear gates only **squeeze**, **rotate** and **displace** these distributions.

- **Squeezing** $S(r)$:



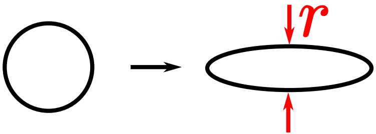
- **Rotation (or phaseshift)** $R(\theta)$ (particle number preserving)



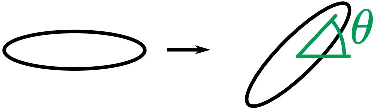
Linear gates

Roughly speaking, linear gates only **squeeze**, **rotate** and **displace** these distributions.

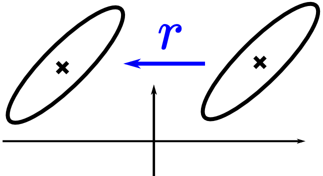
- **Squeezing** $S(r)$:



- **Rotation (or phaseshift)** $R(\theta)$ (particle number preserving)



- **Displacement** $D(r)$



Parameter shift rules for CV quantum circuits?

Applicable parameter shift rules so far

1. Circuits with only **linear gates**, no nonlinearities allowed [Mitarai'18, Schuld'18]

Applicable parameter shift rules so far

1. Circuits with only **linear gates**, no nonlinearities allowed [Mitarai'18, Schuld'18]
2. Circuits with only **particle number-preserving linear gates** (i.e., phaseshift gates), but particle number-preserving nonlinearities are allowed [Facelli'24].

Applicable parameter shift rules so far

1. Circuits with only **linear gates**, no nonlinearities allowed [Mitarai'18, Schuld'18]
2. Circuits with only **particle number-preserving linear gates** (i.e., phaseshift gates), but particle number-preserving nonlinearities are allowed [Facelli'24].

   In this method, trigonometric interpolation is used: $f(\theta)$ depends on θ as

   $$f(\theta) = \sum_{k=-R}^{R} c_k e^{ik\theta}, \tag{8}$$

   where
   - $R \propto$ number of particles,
   - $c_k \implies$ determined via **trigonometric interpolation**.

Applicable parameter shift rules so far

1. Circuits with only **linear gates**, no nonlinearities allowed [Mitarai'18, Schuld'18]
2. Circuits with only **particle number-preserving linear gates** (i.e., phaseshift gates), but particle number-preserving nonlinearities are allowed [Facelli'24].

   In this method, trigonometric interpolation is used: $f(\theta)$ depends on θ as

   $$f(\theta) = \sum_{k=-R}^{R} c_k e^{ik\theta}, \tag{8}$$

   where
   - $R \propto$ number of particles,
   - $c_k \implies$ determined via **trigonometric interpolation**.

   Calculating the derivative $f'(\theta)$ is straightforward!

Applicable parameter shift rules so far

1. Circuits with only **linear gates**, no nonlinearities allowed [Mitarai'18, Schuld'18]
2. Circuits with only **particle number-preserving linear gates** (i.e., phaseshift gates), but particle number-preserving nonlinearities are allowed [Facelli'24].

   In this method, trigonometric interpolation is used: $f(\theta)$ depends on θ as

$$f(\theta) = \sum_{k=-R}^{R} c_k e^{ik\theta}, \tag{8}$$

   where

   - $R \propto$ number of particles,
   - $c_k \implies$ determined via **trigonometric interpolation**.

   Calculating the derivative $f'(\theta)$ is straightforward!

Can we use similar interpolation techniques in CV circuits generally?

Interpolating phaseshift gates in CV circuits

Challenge: In the CV setup, the number of particles is usually not fixed.

Interpolating phaseshift gates in CV circuits

Challenge: In the CV setup, the number of particles is usually not fixed.

However: The contributions corresponding to higher particle numbers are "small" in usual cases:

$$f(\theta) \approx \hat{f}(\theta) := \sum_{k=-R}^{R} c_k e^{ik\theta} \tag{9}$$

Interpolating phaseshift gates in CV circuits

Challenge: In the CV setup, the number of particles is usually not fixed.

However: The contributions corresponding to higher particle numbers are "small" in usual cases:

$$f(\theta) \approx \hat{f}(\theta) := \sum_{k=-R}^{R} c_k e^{ik\theta} \tag{9}$$

Expectation: By increasing the degree of the trigonometric polynomial $\hat{f}$, the distance between f and $\hat{f}$ decreases sufficiently fast for optimization purposes.

Interpolating phaseshift gates in CV circuits

Challenge: In the CV setup, the number of particles is usually not fixed.

However: The contributions corresponding to higher particle numbers are "small" in usual cases:

$$f(\theta) \approx \hat{f}(\theta) := \sum_{k=-R}^{R} c_k e^{ik\theta} \tag{9}$$

Expectation: By increasing the degree of the trigonometric polynomial $\hat{f}$, the distance between f and $\hat{f}$ decreases sufficiently fast for optimization purposes.

Observation: similar strategy works for the Kerr gate $K(\kappa)$ (a nonlinear particle number-preserving gate) by increasing R.

Interpolating phaseshift gates in CV circuits

Challenge: In the CV setup, the number of particles is usually not fixed.

However: The contributions corresponding to higher particle numbers are "small" in usual cases:

$$f(\theta) \approx \hat{f}(\theta) := \sum_{k=-R}^{R} c_k e^{ik\theta} \tag{9}$$

Expectation: By increasing the degree of the trigonometric polynomial $\hat{f}$, the distance between f and $\hat{f}$ decreases sufficiently fast for optimization purposes.

Observation: similar strategy works for the Kerr gate $K(\kappa)$ (a nonlinear particle number-preserving gate) by increasing R.

Question: How do expectation values depend on the *active* linear gates?

Displacement gate interpolation

Denoting $\langle n | D(r) | m \rangle := D(r)_{n,m}$, we can write the following recursion [Miatto'20]:

$$D(r)_{0,0} = e^{-r^2/2}, \qquad D(r)_{n+1,0} = \frac{r}{\sqrt{n+1}} D(r)_{n,0} \,,$$
$$D(r)_{n,m+1} = \frac{1}{\sqrt{m+1}} \left(\sqrt{n} D(r)_{n-1,m} - r D(r)_{n,m} \right) , \tag{10}$$

Displacement gate interpolation

Denoting $\langle n| D(r) |m\rangle := D(r)_{n,m}$, we can write the following recursion [Miatto'20]:

$$D(r)_{0,0} = e^{-r^2/2}, \qquad D(r)_{n+1,0} = \frac{r}{\sqrt{n+1}} D(r)_{n,0} \,,$$
$$D(r)_{n,m+1} = \frac{1}{\sqrt{m+1}} \left(\sqrt{n} D(r)_{n-1,m} - r D(r)_{n,m} \right) , \tag{10}$$

and hence

$$g(r) := \langle\psi_0| D^\dagger(r) \hat{O} D(r) |\psi_0\rangle = e^{-r^2/2} \sum_{k=0}^{\infty} c_k r^k \quad (c_k \in \mathbb{C}) \tag{11}$$

for a fixed state $|\psi_0\rangle$. Omitting high particle number contributions we get

$$g(r) \approx \hat{g}(r) := p(r)\, e^{-r^2/2}, \qquad p \text{ polynomial in } r. \tag{12}$$

Displacement gate interpolation

Denoting $\langle n| D(r) |m\rangle := D(r)_{n,m}$, we can write the following recursion [Miatto'20]:

$$
\begin{aligned}
D(r)_{0,0} = e^{-r^2/2}, \qquad D(r)_{n+1,0} = \frac{r}{\sqrt{n+1}} D(r)_{n,0}\ , \\
D(r)_{n,m+1} = \frac{1}{\sqrt{m+1}} \left(\sqrt{n} D(r)_{n-1,m} - r D(r)_{n,m} \right) ,
\end{aligned}
\tag{10}
$$

and hence

$$
g(r) := \langle \psi_0 | D^\dagger(r) \hat{O} D(r) | \psi_0 \rangle = e^{-r^2/2} \sum_{k=0}^{\infty} c_k r^k \quad (c_k \in \mathbb{C})
\tag{11}
$$

for a fixed state $|\psi_0\rangle$. Omitting high particle number contributions we get

$$
g(r) \approx \hat{g}(r) := p(r)\, e^{-r^2/2}, \qquad p \text{ polynomial in } r.
\tag{12}
$$

We can interpolate $\hat{g}(r)$ by **polynomial interpolation** of $p(r)$.

Squeezing gate interpolation

Analogously, we can show that

$$h(r) := \langle\psi_0| S^\dagger(r)\hat{O}S(r)|\psi_0\rangle = \sum_{k=0}^{\infty} c_k(\tanh r)^k + d_k(\tanh r)^k \operatorname{sech} r \quad (c_k, d_k \in \mathbb{C}). \tag{13}$$

Squeezing gate interpolation

Analogously, we can show that

$$h(r) := \langle\psi_0| S^\dagger(r)\hat{O}S(r)|\psi_0\rangle = \sum_{k=0}^{\infty} c_k(\tanh r)^k + d_k(\tanh r)^k \operatorname{sech} r \quad (c_k, d_k \in \mathbb{C}). \tag{13}$$

Similarly, omitting high particle number contributions:

$$h(r) \approx \hat{h}(r) := p(\tanh r) + q(\tanh r)\operatorname{sech} r, \tag{14}$$

where p and q are polynomials. As before, we can use **polynomial interpolation**.

Determining gradient via interpolation

So far: we can approximate gradients for many gates (Squeezing, Rotation, Displacement, Kerr) using the interpolating polynomials.

Determining gradient via interpolation

So far: we can approximate gradients for many gates (Squeezing, Rotation, Displacement, Kerr) using the interpolating polynomials.

Fact: Linear gates + Kerr gate $\implies$ universality $\implies$ high expressivity!

Determining gradient via interpolation

So far: we can approximate gradients for many gates (Squeezing, Rotation, Displacement, Kerr) using the interpolating polynomials.

Fact: Linear gates + Kerr gate $\implies$ universality $\implies$ high expressivity!

However: numerical simulations show, that it might not be worth pursuing this direction.

Determining gradient via interpolation

So far: we can approximate gradients for many gates (Squeezing, Rotation, Displacement, Kerr) using the interpolating polynomials.

Fact: Linear gates + Kerr gate $\implies$ universality $\implies$ high expressivity!

However: numerical simulations show, that it might not be worth pursuing this direction.

$\boxed{\textbf{Question:} \text{ Can we use the interpolating polynomials for something better?}}$

Line search strategy

Basic idea

Idea [Nádori'25]: Instead of calculating gradients, use the minima from interpolating polynomials!

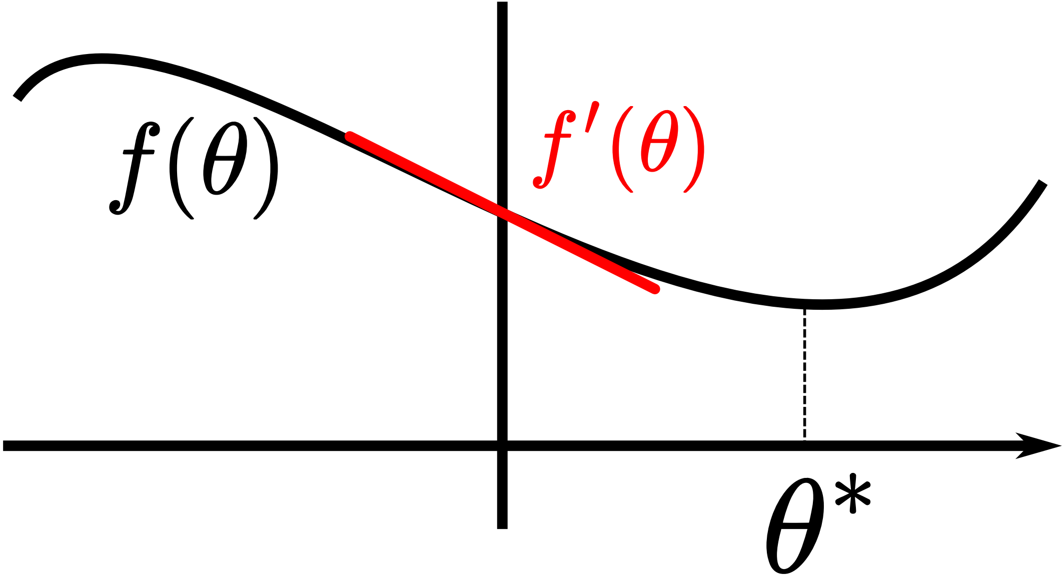
Basic idea

Idea [Nádori'25]: Instead of calculating gradients, use the minima from interpolating polynomials!

We sample parameters $\Lambda \in P(\{1, \cdots, L\})$, where L is the number of parameters, and then in each iteration step we modify the parameters $\boldsymbol{\theta}_i$ as

$$\boldsymbol{\theta}_i \mapsto \begin{cases} \boldsymbol{\theta}_i^* & i \in \Lambda, \\ \boldsymbol{\theta}_i & i \notin \Lambda, \end{cases} \tag{15}$$

where $\boldsymbol{\theta}_i^*$ is the **parameter-wise minimum** determined via interpolation.


$f(\theta)$
$f'(\theta)$
θ^*

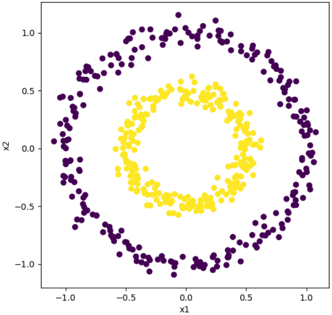
Simple example: Circle classifier

Consider a 2D binary classification datasets, with two circles, one contained in another:

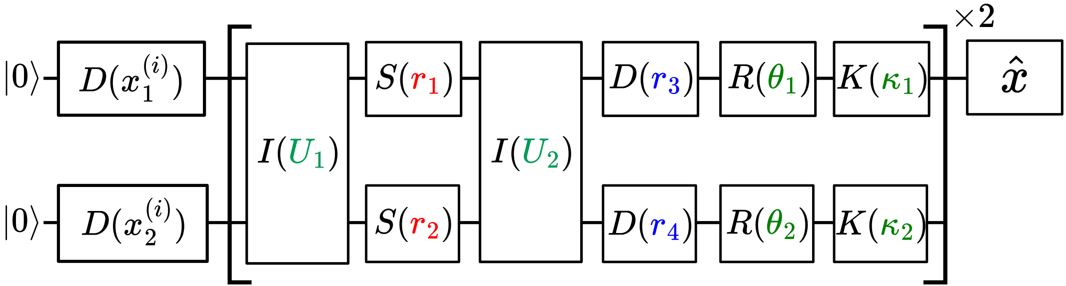
$$\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N_{\text{tr}}}, \tag{16}$$

where

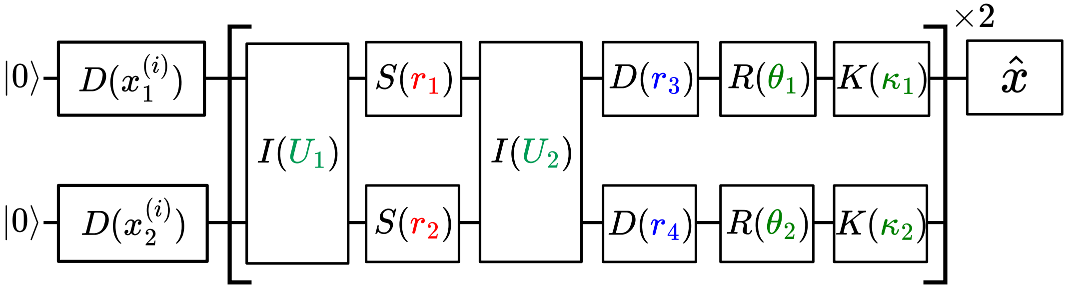
- $\boldsymbol{x}^{(i)}$: data features,
- $y^{(i)} \in \{0, 1\}$: associated labels.



Circuit


$|0\rangle$
$D(x_1^{(i)})$
$|0\rangle$
$D(x_2^{(i)})$
$I(U_1)$
$S(r_1)$
$S(r_2)$
$I(U_2)$
$D(r_3)$
$R(\theta_1)$
$K(\kappa_1)$
$D(r_4)$
$R(\theta_2)$
$K(\kappa_2)$
$\times 2$
$\hat{x}$

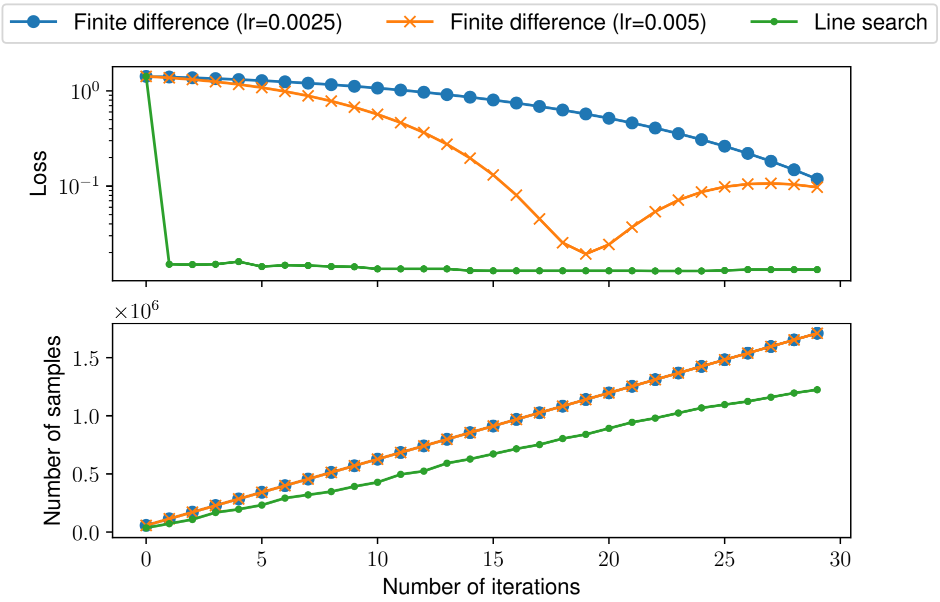
Circuit

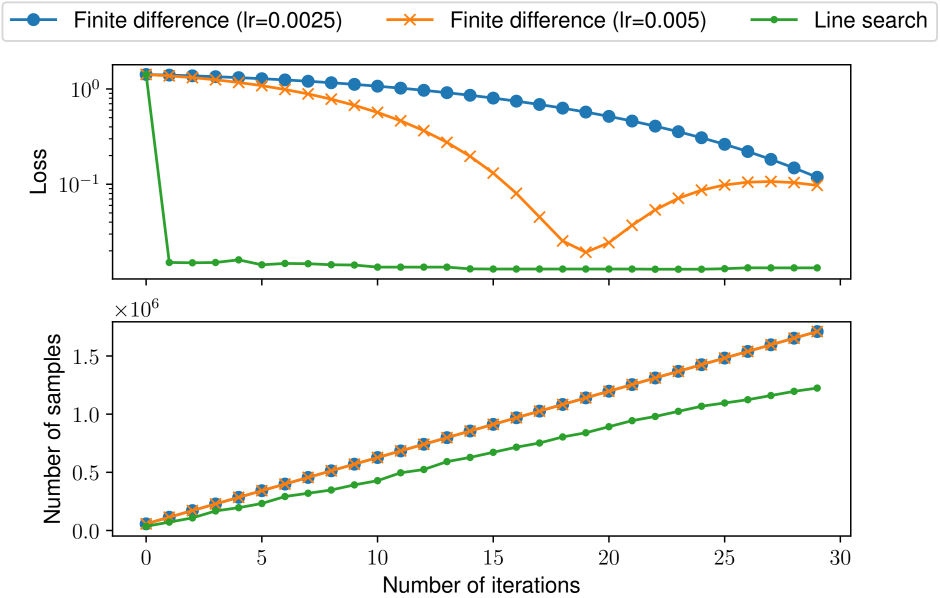


We use **mean-squared error** (MSE) as loss function:

$$\mathcal{L}(\boldsymbol{\theta}, \mathcal{D}) = \frac{1}{N_{\mathrm{tr}}} \sum_{i=1}^{N_{\mathrm{tr}}} \left(f(\boldsymbol{\theta}, \mathbf{x}^{(i)}) - 0.1\,(2y^{(i)} - 1) \right)^2, \tag{17}$$

where $f(\boldsymbol{\theta}, \mathbf{x}^{(i)})$ is an expectation value of $\hat{x}$.


Finite difference (lr=0.0025)
Finite difference (lr=0.005)
Line search
Loss
10^0
10^{-1}
$\times 10^6$
Number of samples
0.0
0.5
1.0
1.5
0
5
10
15
20
25
30
Number of iterations



(I used PIQUASSO [ZK'25])

Outlook & Remaining questions

- This is very heuristic at this point! $\implies$ we need to determine some guarantees regarding the interpolation error.

Outlook & Remaining questions

- This is very heuristic at this point! $\Longrightarrow$ we need to determine some guarantees regarding the interpolation error.
- How does noise affect the optimization procedure?

Outlook & Remaining questions

- This is very heuristic at this point! $\Longrightarrow$ we need to determine some guarantees regarding the interpolation error.
- How does noise affect the optimization procedure?
- Comparison with other optimization methods, e.g., COBYLA or SPSA.

Outlook & Remaining questions

- This is very heuristic at this point! $\Longrightarrow$ we need to determine some guarantees regarding the interpolation error.
- How does noise affect the optimization procedure?
- Comparison with other optimization methods, e.g., COBYLA or SPSA.
- Adapt to maximum-mean discrepancy (MMD) and Kullback-Leibler (KL) divergence as loss functions.

Outlook & Remaining questions

- This is very heuristic at this point! $\implies$ we need to determine some guarantees regarding the interpolation error.
- How does noise affect the optimization procedure?
- Comparison with other optimization methods, e.g., COBYLA or SPSA.
- Adapt to maximum-mean discrepancy (MMD) and Kullback-Leibler (KL) divergence as loss functions.
- Can we use the approximated gradients in training QPINNs?

Outlook & Remaining questions

- This is very heuristic at this point! $\implies$ we need to determine some guarantees regarding the interpolation error.
- How does noise affect the optimization procedure?
- Comparison with other optimization methods, e.g., COBYLA or SPSA.
- Adapt to maximum-mean discrepancy (MMD) and Kullback-Leibler (KL) divergence as loss functions.
- Can we use the approximated gradients in training QPINNs?
- Can this help mitigating barren plateaus?

Thank you for your attention!

Email: kolarovszki.zoltan@wigner.hun-ren.hu