Quantum Information Processing (2023) 22:132
https://doi.org/10.1007/s11128-023-03876-8

Evolution strategies: application in hybrid quantum-classical neural networks

Lucas Friedrich[1] · **Jonas Maziero**[1]

Received: 17 May 2022 / Accepted: 6 February 2023 / Published online: 28 February 2023

Abstract

With the rapid development of quantum computers, several applications are being proposed for them. Quantum simulations, simulation of chemical reactions, solution of optimization problems and quantum neural networks (QNNs) are some examples. However, problems such as noise, limited number of qubits and circuit depth, and gradient vanishing must be resolved before we can use them to their full potential. In the field of quantum machine learning, several models have been proposed. In general, in order to train these different models, we use the gradient of a cost function with respect to the model parameters. In order to obtain this gradient, we must compute the derivative of this function with respect to the model parameters. One of the most used methods in the literature to perform this task is the parameter-shift rule method. This method consists of evaluating the cost function twice for each parameter of the QNN. A problem with this method is that the number of evaluations grows linearly with the number of parameters. In this work, we study an alternative method, called evolution strategies (ES), which are a family of black box optimization algorithms which iteratively update the parameters using a search gradient. An advantage of the ES method is that in using it, one can control the number of times the cost function will be evaluated. We apply the ES method to the binary classification task, showing that this method is a viable alternative for training QNNs. However, we observe that its performance will be strongly dependent on the hyperparameters used. Furthermore, we also observe that this method, alike the parameter shift rule method, suffers from the problem of gradient vanishing.

Keywords Quantum neural networks · Evolution strategies · Binary classification · Hybrid quantum-classical neural networks

✉ Jonas Maziero
jonas.maziero@ufsm.br

Lucas Friedrich
lucas.friedrich@acad.ufsm.br

[1] Physics Department, Center for Natural and Exact Sciences, Federal University of Santa Maria, Roraima Avenue 1000, 97105-900 Santa Maria, RS, Brazil

1 Introduction

Many developments in science and technology in the last years were obtained with the aid of artificial intelligence. Its applications extend to the most varied areas of knowledge, such as for example computer vision [1–3], natural language processing [4, 5], drug discovery [6], analysis of astronomical images [7], and chemistry simulations [8]. With the development of quantum computers, several studies are being conducted with the aim of taking artificial intelligence to the quantum domain [9–16]. It is hoped that by utilizing phenomena such as entanglement and superposition, we will be able to create models more powerful than their classical counterparts.

Models such as quantum multilayer perceptron [17], quantum convolutional neural networks [18], quantum kernel method [19], and quantum-classical hybrid neural networks (HQCNN) [20–23] are some candidate models. In the era of noisy intermediate-scale quantum devices (NISQ), hybrid models are the most used. This era is characterized by the limited number of qubits we have access to and the presence of noise. Hybrid models are built using sequential classical and quantum layers. With this, we are able to create models with fewer qubits. For training these models, the gradient descent method or its variants has been used in the literature. The gradient descent method consists of using the gradient of a cost function to update the parameters of the neural network. To obtain the gradient of the quantum layers, one can use the method called parameter-shift rule (PSR), which consists of evaluating the cost function for each parameter of the quantum layers [24, 28]. In the NISQ era, where the number of qubits that we have access to is limited and the depth of the parameterizations is also limited by noise and decoherence, the use of the PSR method is the most indicated because with it we are able to obtain the derivatives of the cost function analytically. However, as quantum computers develop, increasing depth and qubit numbers make this method impractical. Therefore, it is necessary to find alternative ways for the task of training.

Evolution strategies (ES) [25–27] are a family of black box optimization algorithms. These algorithms have already been applied to a variety of classical problems. For example, ES has been shown to be an alternative to reinforcement learning [29]. However, application in the quantum domain is still understudied, with only a few works [30–32] using these methods. Such a strategy is promising, because the number of evaluations of the cost function does not scale with the number of parameters. Furthermore, as the cost function evaluations are independent, this method can be parallelized. However, its parallelized application in quantum computing is still limited in the NISQ era. Finally, we should note that this method will also be strongly influenced by the number of times the cost function will be evaluated, that is, the value of λ used, because the lower this value, the lower the accuracy of this method.

This article is organized as follows. In Sect. 2, we review the HQCNN models, where a parallel is made with the classical deep neural network models. After that in Sect. 2.1, we briefly discuss how classical layers work. In Sect. 2.2, we describe how a quantum layer works, and in Sect. 2.2.1, we discuss different ways of mapping classical data into quantum states. In the sequence, in Sect. 2.2.2, we discuss how the parameterization of a quantum layer is done. Finally, in Sect. 2.2.3, we describe how measurements can be made on HQCNN. In Sect. 3, we present a review of the ES

method, showing how to estimate the gradient of a given function. In Sect. 4, we start by describing the two HQCNN models that we use in this study. In Sect. 4.2, we show how a HQCNN model is trained, and we present our training proposal using ES. In Sect. 5, we briefly describe how the simulations are done, and, in Sect. 6, we present the results we obtained. Finally, in Sect. 7, we give our conclusions.

2 Quantum-classical hybrid neural networks

Classical deep neural network models are in great extent responsible for the success of artificial intelligence in the last few decades. These networks are characterized by the use of several concatenated classical layers. That is to say, for a model with depth d, we have

$$C = L_{n_{d-1}\to n_d} \circ L_{n_{d-2}\to n_{d-1}} \circ \cdots \circ L_{n_1\to n_2} \circ L_{n_0\to n_1}, \tag{1}$$

where each L represents a classical layer and the first and second indices represent the input size and the output size, respectively. HQCNN models are also characterized by the use of several concatenated layers, that is,

$$\mathcal{Q} = \mathcal{L}_{n_{d-1}\to n_d} \circ \mathcal{L}_{n_{d-2}\to n_{d-1}} \circ \cdots \mathcal{L}_{n_1\to n_2} \circ \mathcal{L}_{n_0\to n_1}, \tag{2}$$

where each $\mathcal{L}$ represents a classical or quantum layer. We can see that the difference between classical deep neural network models and hybrid quantum-classical models is due to the addition of quantum layers to the classical model. It is expected that by using quantum layers together with classical layers, we will be able to build models with greater power and accuracy than models using only classical layers.

2.1 Classical layer

The structures known as *layers* are one of the main building blocks of all modern classical deep neural network models. Such structures map an input of dimension n_0 to an output of dimension n_1. A typical example of these structures is a linear transformation followed by a nonlinear activation function, defined by

$$\mathcal{L}_{n_0\to n_1} = \phi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{3}$$

where $\boldsymbol{x}$ is an input vector with dimension n_0, $\boldsymbol{W}$ is a matrix with dimensions $n_1 \times n_0$ and $\boldsymbol{b}$ is a vector with dimension n_1. The elements of $\boldsymbol{W}$ are real values that are updated throughout training. One of the key pieces of deep neural network models is the nonlinearity implemented by the ϕ function. There are several functions that we can use to apply nonlinearity, such as the hyperbolic tangent, the Sigmoid or the ReLu.

Furthermore, we can mention the neural network architecture with convolutional layers, which are mainly applied to problems involving computer vision, and the long short-term memory (LSTM), which are used when dealing with problems related to time series, among many others.

2.2 Quantum layer

2.2.1 Encoder

The first task when building a quantum layer is to encode the data of interest in quantum states. For this, we can use different strategies, such as the wave function encoder

$$|\boldsymbol{x}\rangle := \frac{1}{\|\boldsymbol{x}\|_2^2} \sum_{i=1}^{2^N} x_i |i\rangle, \tag{4}$$

the dense angle coding

$$|\boldsymbol{x}\rangle = \bigotimes_{i=1}^{N/2} \cos(\pi x_{2i-1})|0\rangle + e^{2\pi i x_{2i}} \sin(\pi x_{2i-1})|1\rangle, \tag{5}$$

or the qubit encoding

$$|\boldsymbol{x}\rangle = \bigotimes_{i=1}^{N} \cos(x_i)|0\rangle + \sin(x_i)|1\rangle. \tag{6}$$

The performance of quantum layers is influenced by this choice [33–35]. For example, it is shown, in Ref. [34], that if we reload the data between the different layers of the parameterization, we will be able to create a model with greater classification capability.

For the purposes of this work, we use two alternative forms of data encoding, Figs. 2 and 4. Furthermore, in subsections model 1 and model 2, Figs. 1 and 6 show which parameterization is used for each case.

2.2.2 Parameterization

After mapping the data of interest to a quantum state, the next step is to apply a parameterization. We write our parameterization as

$$U(\boldsymbol{\theta}) = \prod_{i=1}^{L} U_i W_i, \tag{7}$$

with

$$U_i = \bigotimes_{j=1}^{N} R_\sigma^{j,i}(\theta_{j,i}), \tag{8}$$

where $R_\sigma^{i,j}(\theta_{j,i}) = e^{-i\theta_{j,i}\sigma/2}$ with $\sigma \in \{\sigma_x, \sigma_y, \sigma_z, \}$ being one of the Pauli matrices and W_i are unparameterized gates.

There are several ways to build the parameterization, and different parameterizations have different expressiveness. Expressiveness is defined as the ability of a given parameterization to explore the Hilbert space. However, the relationship between this choice and the performance of the model is not direct.

2.2.3 Measurements

After mapping the data of interest to a quantum state and applying the parameterization $U(\boldsymbol{\theta})$, Eq. (7), the next step is to perform the measurements. The measurements can be done globally, where all qubits are measured, or locally, where only a few qubits are measured individually or in pairs. As a result of these measurements, we can estimate the mean value

$$f_i = Tr[A_i \rho_x^{out}], \tag{9}$$

where A_i is a Hermitian operator and ρ_x^{out} is the density matrix at the end of the quantum circuit. That is, given an input $|\boldsymbol{x}\rangle$ and a parameterization $U(\boldsymbol{\theta})$, Eq. (7), we have that $\rho_x^{out} = U(\boldsymbol{\theta})|\boldsymbol{x}\rangle\langle\boldsymbol{x}|U(\boldsymbol{\theta})^\dagger$.

A particular case is

$$A_i = |i\rangle\langle i|. \tag{10}$$

In this case, the outputs will be the respective probabilities of our circuit being in a state of the computational basis. This is a definition of what we can call a global measurement, where all qubits contribute to the value of f_i.

Another case is when we define

$$A_i = \mathbb{I}_{\bar{i}} \otimes |0\rangle\langle 0|_i. \tag{11}$$

Here, the index $\bar{i}$ indicates that the identity operator will be applied to all qubits with exception to the qubit with index i. The index i indicates that $|0\rangle\langle 0|$ will be applied only to the qubit of index i. This is a definition for what we call local measurements, where only the qubit with index i contributes to the value of f_i. In this specific case, where $|0\rangle\langle 0|$ is used, we will get the probability for the qubit i to be in the state $|0\rangle$.

3 Evolution strategies

Given a function $f(\boldsymbol{z})$, with $\boldsymbol{z} \in \mathbb{R}^d$, in the evolution strategy (ES), we reparameterize this function as follows:

$$J(\theta) = \mathbb{E}_\theta[f(\boldsymbol{z})] = \int f(\boldsymbol{z})\pi(\boldsymbol{z}|\theta)\mathrm{d}\boldsymbol{z}. \tag{12}$$

By differentiating Eq. (12) with respect to θ, we get

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \int f(\boldsymbol{z})\pi(\boldsymbol{z}|\theta)\mathrm{d}\boldsymbol{z} \\ &= \int [f(\boldsymbol{z})\nabla_\theta \log \pi(\boldsymbol{z}|\theta)]\pi(\boldsymbol{z}|\theta)\mathrm{d}\boldsymbol{z} = \mathbb{E}_\theta[f(\boldsymbol{z})\nabla_\theta \log \pi(\boldsymbol{z}|\theta)].\end{aligned} \tag{13}$$

For more details about the derivation given in Eq. (13), see Ref. [27].

Thus, we can estimate the gradient using

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda}\sum_{k=1}^{\lambda} f(\boldsymbol{z}_k)\nabla_\theta \log \pi(\boldsymbol{z}_k|\theta), \tag{14}$$

where λ is the number of random samples in the parameter space. For the case of a Gaussian distribution, we have

$$\pi(\boldsymbol{z}|\theta) = \frac{1}{\sqrt{(2\pi)^d \det \boldsymbol{\Sigma}}} \exp\Big(-\frac{1}{2}(\boldsymbol{z}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{z}-\boldsymbol{\mu})\Big), \tag{15}$$

where $\boldsymbol{\mu}$ is a mean vector and $\boldsymbol{\Sigma}$ is the covariance matrix. So the parameter θ is defined as $\theta := \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. Therefore, we have that $\nabla_\theta \log \pi(\boldsymbol{z}|\theta)$ will be given by

$$\nabla_{\boldsymbol{\mu}} \log \pi(\boldsymbol{z}|\theta) = \boldsymbol{\Sigma}^{-1}(\boldsymbol{z}-\boldsymbol{\mu}) \tag{16}$$

and

$$\nabla_{\boldsymbol{\Sigma}} \log \pi(\boldsymbol{z}|\theta) = \frac{1}{2}\boldsymbol{\Sigma}^{-1}(\boldsymbol{z}-\boldsymbol{\mu})(\boldsymbol{z}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1} - \frac{1}{2}\boldsymbol{\Sigma}^{-1}. \tag{17}$$

The main ES methods use both Eqs. (16) and (17) to obtain an estimate of Eq. (14). In this study, we will use $\boldsymbol{\Sigma} = \sigma^2 I$, with σ being a constant. So we have

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda\sigma^2}\sum_{k=1}^{\lambda}(\boldsymbol{z}_k - \mu)\cdot f(\boldsymbol{z}_k), \tag{18}$$

with $\boldsymbol{z}_k \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 I)$. The algorithm that implements Eq. (18) is shown.

Input: $f(z), \theta_i, \sigma$
Output: $\nabla_\theta J(\theta)$
for $k = 1...\lambda$ **do**
  $z_k \sim \mathcal{N}(\mu, \sigma^2 I)$
  $f(z_k)$ / * Evaluates function f in z_k */
  $\nabla_\theta J(\theta) = (z_k - \mu)$ / * Calculates the derivative */
end
$\nabla_\theta J(\theta) = \frac{1}{\lambda\sigma^2}\sum_{k=1}^{\lambda}(z_k - \mu)\cdot f(z_k)$ / * Estimates the gradient*/

Algorithm 1: Gradient estimation using Eq. (18).

4 Method

4.1 Models

For this study, we will use two HQCNN models. The first model will be built using two quantum layers. The second model will be built using two classical layers and a quantum layer. In this work, we define the cost function as

$$C(\Delta) := \frac{1}{N}\sum_{j=1}^{N}(y_i^j - \bar{y}_i^j)^2, \tag{19}$$

where y_i is the vector obtained at the end of the network given the input x_i and $\bar{y}_i$ is the respective desired output. Here, $\Delta := \{\boldsymbol{W}, \boldsymbol{\theta}\}$ is defined as the set of network parameters.

4.1.1 Model 1

The first layer will be built using five qubits, as illustrated in Fig. 1. Data will be mapped using real amplitudes as illustrated in Fig. 2. This is a parameterization used for machine learning and quantum chemistry problems. The parameterization will be given by $U(\boldsymbol{\theta})$, Eq. (7). Figure 3 shows how the parameterization is done for each U_i. For measurements, we will use Eq. (9) with the Hermitian operator defined in Eq. (11) for the last three qubits.

For the second layer, three qubits are used. The function that will map our data into a quantum state is represented in Fig. 4. The parameterization used will also be given by $U(\boldsymbol{\theta})$, Eq. 7. For the measurements, we use Eq. (9) with H_i defined in Eq. (11). In this case, the last two qubits will be used.

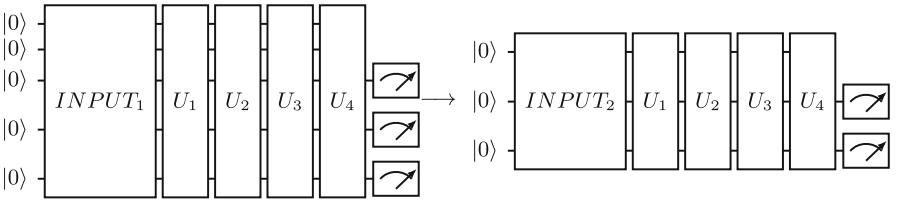


Fig. 1 Model 1: quantum-classical hybrid neural network

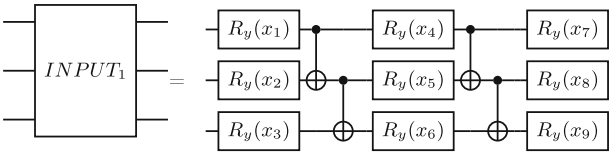


Fig. 2 $INPUT_1$ for the case of three qubits

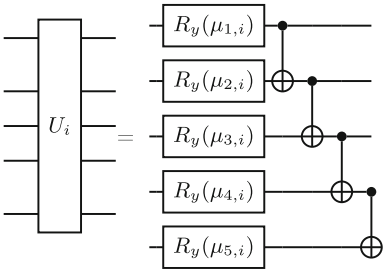


Fig. 3 Parameterization of unitary operators U_i. See Fig. 1

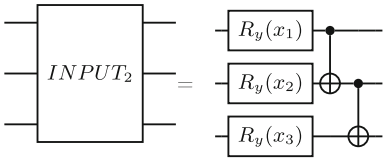


Fig. 4 $INPUT_2$ for the case of three qubits

4.1.2 Model 2

The second model will be built using three layers. The first and third being classical layers, with a quantum second layer. In Eq. (3), we defined the classical layer that we will use. This consists of an operation that takes an input $\boldsymbol{x}$ of size n into an output $\boldsymbol{y}$ of size m, as illustrated in Fig. 5. In this study, as we will work with data from the dataset MNIST, which consists of images of dimension 28×28, the input of the first layer has size $n = 784$. The output data are encoded in the quantum layer using the parameterization model presented in Fig. 4, that is, for each qubit, a value is encoded. Then, the output of the first layer is equal in size to the number of qubits used in the quantum layer. For nonlinearity, the hyperbolic tangent function will be used.

The second layer, which is a quantum layer, is represented in Fig. 6. In this layer, we use four qubits. Its parameterization will be given by $U(\boldsymbol{\theta})$, Eq. (7), with each U_i represented in Fig. 3. For measurements, we use Eq. (9) with the Hermitian operator defined in Eq. (11), with all qubits measured individually. The output of this quantum layer is used as input to a third layer, which is a classical layer, just like the first layer, with the difference being its size. The input of this third layer have dimension equal to the number of qubits used in the quantum layer and output $m = 2$.

4.2 Training

Given $\mathcal{D} := \{x_i, \bar{y}_i\}_{i=1}^n$, a dataset, and defined the HQCNN model that shall be used, the training consists of an iterative method where, given an input x_i to the model, it returns an output y_i. Comparing this output to the desired output $\bar{y}_i$, we can compute the performance of our model using the cost function of Eq. (19). This process is performed iteratively for all data in the set $\mathcal{D}$, several times, or, as it is commonly called, for several epochs. During this process, we aim to minimize the cost function, that is to say, we want to obtain

Fig. 5 Linear layer

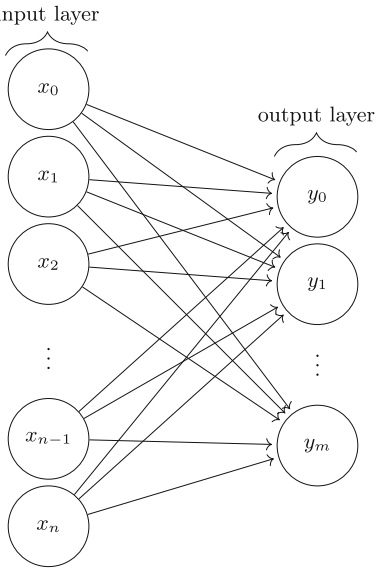
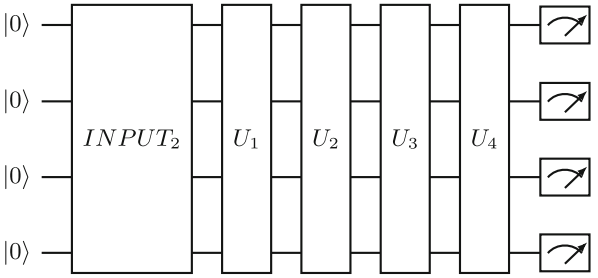


Fig. 6 Quantum layer



$$\Delta_{\text{opt}} = argmin_{\Delta} C(\Delta), \tag{20}$$

where Δ_{opt} is the set of optimal parameters. To obtain Δ_{opt}, at each iteration, the Δ parameters are updated generally using the gradient descent method or its variants. This method updates the parameters using the gradient of the cost function of Eq. (19). For this, we must use the chain rule. For example, let us consider the first model. Given an input $x_i = (x_i^1, x_i^2, \ldots, x_i^n)$ and its respective desired output $\bar{y}_i = (\bar{y}_i^1, \bar{y}_i^2, \ldots, \bar{y}_i^N)$, the cost function is given by

$$C(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = \frac{1}{N} \sum_{j=1}^{N} (\mathcal{L}_2(\mathcal{L}_1(x_i, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2)^j - \bar{y}_i^j)^2, \tag{21}$$

where $\mathcal{L}_1$ is the first layer with input x_i and parameters $\boldsymbol{\theta}_1$ and $\mathcal{L}_2$ are the second layer with input given by the output of the first layer and with parameters $\boldsymbol{\theta}_2$. Thus, the gradient of Eq. (19) in relation to the parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ will be

$$\nabla_{\boldsymbol{\theta}_2} C(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = \frac{2}{N} \sum_{j}^{N} (\mathcal{L}_2(\mathcal{L}_1(x_i, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2)^j - \bar{y}_i^j) \nabla_{\boldsymbol{\theta}_2} \mathcal{L}_2(\mathcal{L}_1(x_i, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2)^j, \quad (22)$$

$$\nabla_{\boldsymbol{\theta}_1} C(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) = \frac{2}{N} \sum_{j}^{N} (\mathcal{L}_2(\mathcal{L}_1(x_i, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2)^j - \bar{y}_i^j) \nabla_{\mathcal{L}_1} \mathcal{L}_2(\mathcal{L}_1(x_i, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2)^j \nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1(x_i, \boldsymbol{\theta}_1), \quad (23)$$

where in Eq. (23), the term $\nabla_{\mathcal{L}_1}$ indicates that we must obtain the derivatives with respect to the input of the second layer. Using gradient descent as an example, from Eqs. (22) and (23), we have that the new parameters will be

$$\boldsymbol{\theta}_1^{t+1} = \boldsymbol{\theta}_1^t - \eta \nabla_{\boldsymbol{\theta}_1} C(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) \quad (24)$$

and

$$\boldsymbol{\theta}_2^{t+1} = \boldsymbol{\theta}_2^t - \eta \nabla_{\boldsymbol{\theta}_2} C(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2) \quad (25)$$

where t represents the epoch and η the learning rate.

Let us consider the term $\nabla_{\boldsymbol{\theta}_1} \mathcal{L}_1(x_i, \boldsymbol{\theta}_1)$ in Eq. (23). Once this is a black box function, we can define $J(\boldsymbol{\theta}_1) := \mathcal{L}_1(x_i, \boldsymbol{\theta}_1)$. Here, we can see that up to an index, we can rewrite our quantum layer using Eq. (12). So, we can use Eq. (18) to estimate the gradient.

In Eq. (23), we see that we must also obtain the gradient with respect to the input data of the second layer. Again, we can consider the quantum layer to be a black box function with parameters given by $\mathcal{L}_2(\mathcal{L}_1(x_i, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2)$. Therefore, we can again use Eq. (12) to describe this layer. Then, the gradient can be estimated using Eq. (18).

4.3 Barren plateaus

The optimization of quantum circuits is done, in general, using the gradient in relation to its parameters. However, a current problem with this procedure is the phenomenon known as gradient vanishing, or barren plateaus. Given a function

$$C = Tr[OU(\boldsymbol{\theta})|\boldsymbol{x}\rangle\langle\boldsymbol{x}|U(\boldsymbol{\theta})^\dagger], \quad (26)$$

if $U(\boldsymbol{\theta})$ is a 2-*design*, then we have that

$$\langle \partial_k C \rangle = 0 \text{ and } Var[\langle \partial_k C \rangle] \approx 2^{-n}, \quad (27)$$

where n is the number of qubits. From the Chebyshev inequality,

$$Pr(|\partial_k C| \geq \delta) \leq \frac{Var[\langle \partial_k C \rangle]}{\delta^2}, \quad (28)$$

we have that the probability that $\partial_k C$ deviates from its mean, $\langle \partial_k C \rangle = 0$, by a value δ will tend to zero as the number of qubits increase.

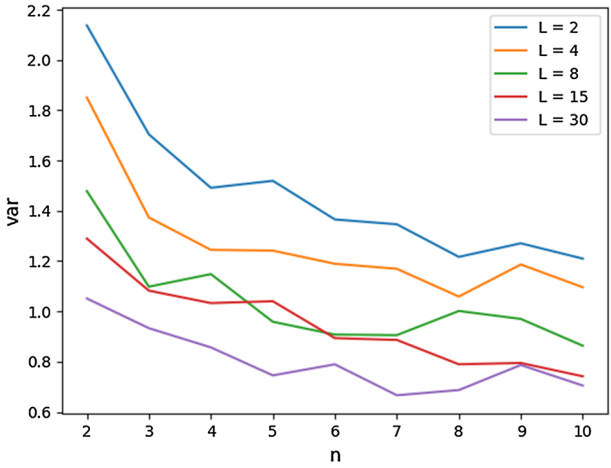


Fig. 7 In this figure, n is the number of qubits used. For the input data, $\boldsymbol{x} = (\pi/4, \pi/4, \pi/4, \ldots, \pi/4)$ was used. We can see that as the number of qubits increases, the variance tends to decrease. Furthermore, we can also see that the variance decreases as the number of layers of the parameterization increases. Thus, we see that the phenomenon of gradient disappearance is also present in the evolution strategy method

Furthermore, results from the literature show that other factors also influence this phenomenon such as the choice of cost function [36], expressiveness of parameterization [37], noise [38], and entanglement [39, 40]. It is also shown that gradient vanishing is present in gradient-free optimization methods [41]. Currently, a large number of methods to mitigate this problem are being proposed [42–46].

In Fig. 7, we showed experimentally that this phenomenon is also observed when using the ES method. To obtain these results, we use the cost function

$$C = \frac{1}{n}\sum_{i=1}^{n} Tr[H_i U(\boldsymbol{\theta})\rho U(\boldsymbol{\theta})^{\dagger}], \tag{29}$$

with $U(\boldsymbol{\theta})$ defined in Eq. (7) and H_i is defined in Eq. (11). To encode the input data, we use the parameterization shown in Fig. 4.

5 Experiments/simulations

For this work, we will use Qiskit [47] and Pytorch [48] to build our models, with the Qiskit package integrated into Pytorch. With this, some steps such as the application of the chain rule to obtain the derivatives in Eqs. (22) and (23) and parameters optimization are done automatically by Pytorch. For training, 2000 training images are used, half referring to zero digit images and half to the one digit images. For validation, 200 images are used, again half of each type. We also vary the value of the learning rate to see how the cost function behaves. Also, we perform the same experiment N times for obtaining the statistics. So, we can see how the ES method

behaves for different initializations. Also, for this study, we used $\sigma = \frac{\pi}{24}$ for all cases. As σ is a hyperparameter that must be defined when starting the training, as well as the number of epochs, learning rate and λ, your choice is free, so we use only one value for σ, since our objective is to analyze whether this method is capable of performing the optimization of the parameters and not which is the best set of hyperparameters. However, we must emphasize that there is no restriction on this choice. For the first experiments, we used

$$\lambda = 4 + 3\log(p), \tag{30}$$

where p is the number of parameters of the respective quantum layer. For the layers where we should get the gradient for the input data, we use the highest value of λ. That is, given the value of λ for the layer's input data and parameters, we will use the largest of them.

Search gradients give us freedom to select the number of function evaluations. The second experiment explores this by using the same values for η used in experiment one, but now with differing evaluation numbers, $\lambda = 2$, $\lambda = 4$, and $\lambda = 6$. Both experiments use HQCNN models applied to part of the MNIST dataset.

6 Results

The first experiment fed a subset of the MNIST dataset into models 1 and 2. Optimization was performed using Adam with a variable learning rate, η. For all experiments, 100 epochs were used for training. For each learning rate value, the same experiment was repeated four times. At each new training, the parameters were randomly initialized. With this, we can see how the ES method behaves at each startup.

In Fig. 8, we can see that for $\eta = 0.01$, the neural network becomes stuck in a local minima. For $\eta = 0.001$, we see that the neural network can learn as the epochs go by. For $\eta = 0.0001$, we see that for this number of epochs used, the neural network performed worse than in the other cases. But we can see that even as the epochs passed, the network was able to learn without getting stuck in a local minimum, as was the case for $\eta = 0.01$.

Results shown in Fig. 9 use Model 2, whereas results in Fig. 8 use Model 1. Both models used the same training and validation sets. Again, we use the Adam optimizer with variable learning rate. Unlike the previous model, we see that for both $\eta = 0.01$ and $\eta = 0.001$, the neural network was stuck in a local minimum, being unable to learn. The neural network was only able to learn when we used $\eta = 0.0001$, where we can see that it quickly converged to the desired minimum.

In the next graphs, in Figs. 10 and 11, Models 1 and 2 were used, respectively, with the data set defined as in the previous experiments, and with a variable learning rate. In these two cases, the difference lies in the number of times the cost function is evaluated, that is to say, the value of λ that is used to estimate the gradient, Eq. (18). In the case of two quantum layers, Fig. 10, we see that using $\eta = 0.01$ the neural network was not able to learn for any value of λ. For $\eta = 0.001$, its behavior was similar for all values of λ. As for $\eta = 0.0001$, its behavior was not better than using $\eta = 0.001$,

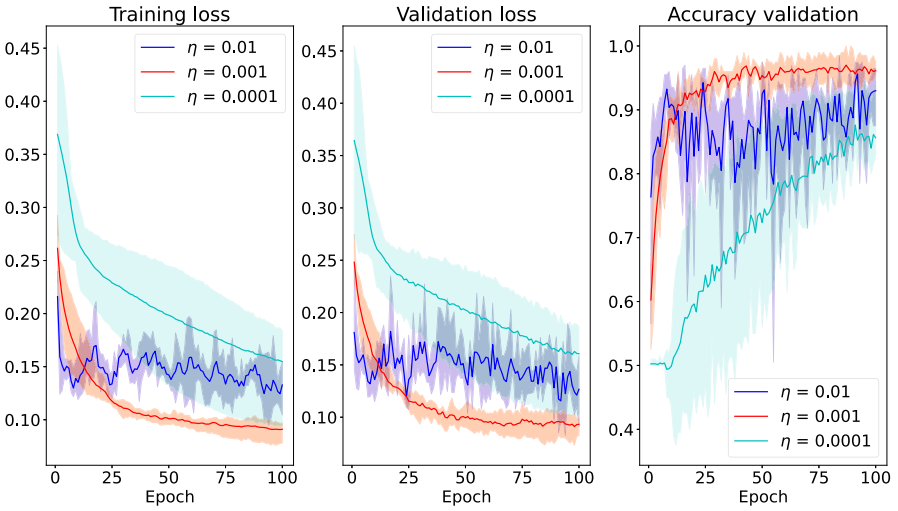


Fig. 8 Result for Model 1. In darker colors, the means for $N = 4$ experiments are shown. The lighter colors represent values between the minimum and maximum values. To obtain the gradient estimate, Algorithm 1 is used. In order to be able to update the parameters of the first layer, we must get the gradient from the input data of the second layer. For this, we can also use Algorithm 1, with $\boldsymbol{\theta}$ being the input data in this case

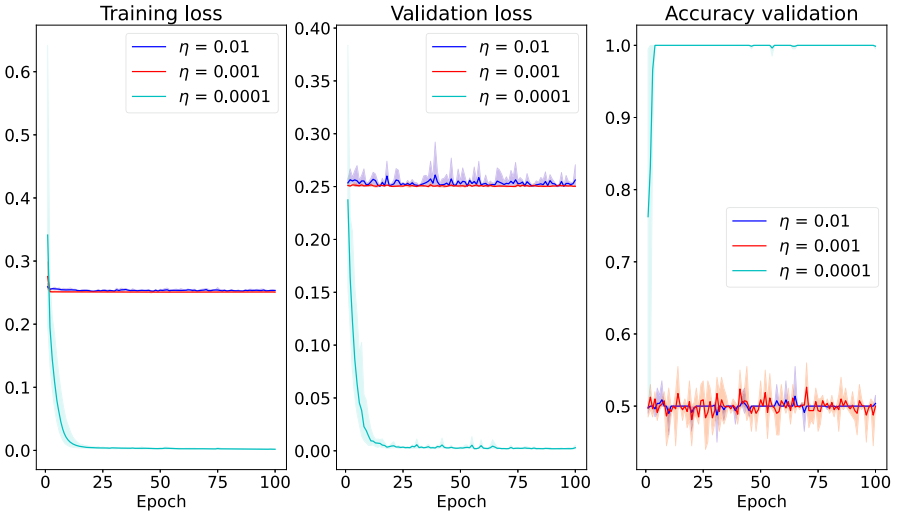


Fig. 9 Results for Model 2. The means for $N = 4$ experiments are shown in darker colors, while the lighter colors represent values between the minimum and maximum values. Algorithm 1 was used to obtain the gradient estimate. To update the parameters of the first layer, we must get the gradient from the input data of the second layer. For this, we can also use Algorithm 1, with $\boldsymbol{\theta}$ being the input data in this case

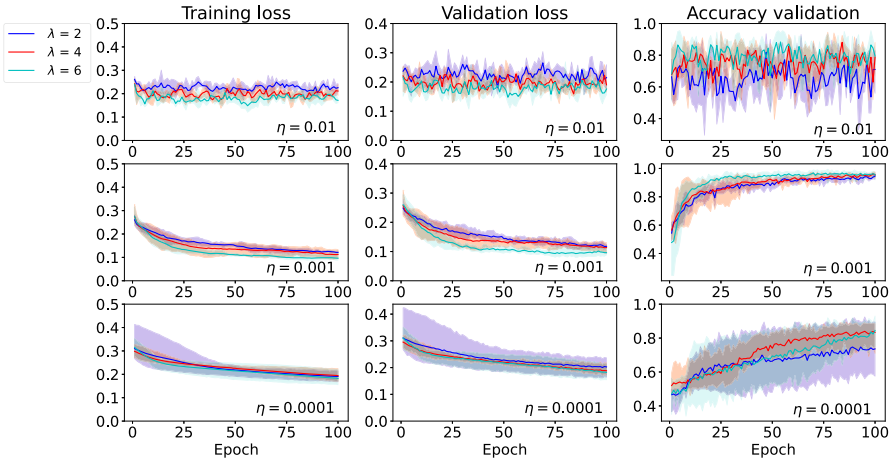


Fig. 10 Results for Model 1 with variable λ and learning rate η. In darker colors, the averages for $N = 4$ experiments are shown. The lighter colors represent values between the minimum and maximum values

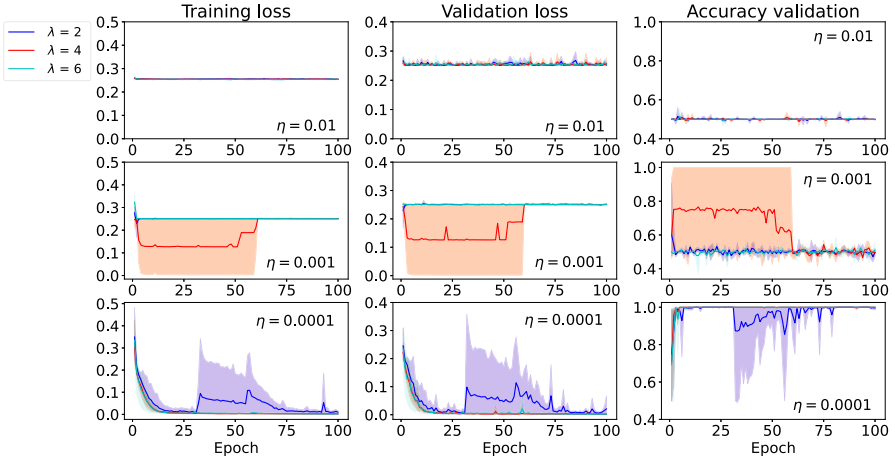


Fig. 11 Results for Model 2 with variable λ and learning rate η. In darker colors, the averages for $N = 4$ experiments are shown. The lighter colors represent values between the minimum and maximum values

but we can see that the neural network was able to learn over the epochs for all values of λ.

For the second model, we see that performance was only satisfactory for $\eta = 0.0001$. For $\eta = 0.01$ and $\eta = 0.001$, networks became trapped in local minima for all values of λ that we explored. With this, we can see experimentally that the performance of the ES method applied in hybrid quantum-classical models, where both classical and quantum layers are used, Model 2 has greater dependence on its hyperparameters.

7 Conclusions

In this article, we introduced the use of the method called evolution strategy in the optimization of classical-quantum hybrid neural networks. We showed that this method is strongly influenced by the hyperparameters η and λ. However, this is a promising method for optimizing hybrid models, once the appropriate hyperparameters are provided. Therefore, for future work, it would be interesting to study methods for selecting hyperparameters, e.g., through standard hyperparameter optimization techniques such as Bayesian optimization. In addition, another possible topic for research is the use of natural gradients, in what is known as a natural evolution strategy. In this case, it would be interesting to evaluate if the performance of the model will be better in relation to the method studied in this work. Finally, as we observed, although the method applied in this article does not directly use the derivatives in relation to the parameters of the quantum circuit to optimize them, it still suffers from the problem of vanishing gradients. So, in the future, one can investigate the use of methods such as the one introduced in Ref. [46], where the optimization of the parameters is done layer by layer to mitigate this problem, and also to see if by performing the training layer by layer the performance will be better than the performance obtained in this article.

Acknowledgements This work was supported by the Foundation for Research Support of the State of Rio Grande do Sul (FAPERGS), by the National Institute for the Science and Technology of Quantum Information (INCT-IQ), process 465469/2014-0, and by the National Council for Scientific and Technological Development (CNPq), process 309862/2021-3.

Data availability The Qiskit/Pytorch code used for implementing the simulations to obtain the data used in this article is available upon request to the authors.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2016)
2. Szegedy, C. et al.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2015)
3. Voulodimos, A., Doulamis, N., Doulamis, A., Protopapadakis, E.: Deep learning for computer vision: a brief review. Comput. Intell. Neurosci. **2018**, e7068349 (2018)
4. Devlin, J., Chang, M.-W. , Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805 (2018)
5. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. Adv. Neural Inf. Process. Syst. **27** (2014)
6. Vamathevan, J., et al.: Applications of machine learning in drug discovery and development. Nat. Rev. Drug Discov. **18**, 463 (2019)
7. Carrasco-Davis, Rodrigo, et al.: Deep learning for image sequence classification of astronomical events. Publ. Astron. Soc. Pac. **131**, 108006 (2019)

8. Cova, T.F.G.G., Pais, A.A.C.C.: Deep learning for deep chemistry: optimizing the prediction of chemical patterns. Front. Chem. **7**, 809 (2019)
9. Garg, S., Ramakrishnan, G.: Advances in quantum deep learning: an overview. arXiv:2005.04316 (2020)
10. Nghiem, N.A., Chen, S.Y.-C., Wei, T.-C.: A unified framework for quantum supervised learning. Phys. Rev. Res. **3**(033056), 2020 (2021)
11. Farhi, E., Neven, H.: Classification with quantum neural networks on near term processors. arXiv:1802.06002 (2018)
12. Tacchino, F., Barkoutsos, P., Macchiavello, C., Tavernelli, I., Gerace, D., Bajoni, D.: Quantum implementation of an artificial feed-forward neural network. Quantum Sci. Technol. **5**, 044010 (2020)
13. Verdon, G., Pye, J., Broughton, M.: A universal training algorithm for quantum deep learning. arXiv:1806.09729 (2018)
14. Beer, K., Bondarenko, D., Farrelly, T., Osborne, T.J., Salzmann, R., Scheiermann, D., Wolf, R.: Training deep quantum neural networks. Nat. Commun. **11**, 808 (2020)
15. Wei, S., Chen, Y., Zhou, Z., Long, G.: A quantum convolutional neural network on NISQ devices. arXiv:2104.06918 (2021)
16. S. Lloyd, S., Schuld, M., Ijaz, A., Izaac, J., Killoran, N.: Quantum embeddings for machine learning. arXiv:2001.03622 (2020)
17. Shao, C.: A quantum model for multilayer perceptron. arXiv:1808.10561 (2018)
18. Wei, S.J., Chen, Y.H., Zhou, Z.R., Long, G.L.: A quantum convolutional neural network on NISQ devices. AAPPS Bull. **32**, 2 (2022)
19. Schuld, M.: Supervised quantum machine learning models are kernel methods. arXiv:2101.11020 (2021)
20. Liu, J., et al.: Hybrid quantum-classical convolutional neural networks. Sci. China Phys. Mech. Astron. **64**, 290311 (2021)
21. Liang, Y., Peng, W., Zheng, Z.-J., Silvén, O., Zhao, G.: A hybrid quantum-classical neural network with deep residual learning. Neural Netw. **143**, 133 (2021)
22. Xia, R., Kais, S.: Hybrid quantum-classical neural network for calculating ground state energies of molecules. Entropy **22**, 828 (2020)
23. Houssein, E.H., Abohashima, Z., Elhoseny, M., Mohamed, W.M.: Hybrid quantum convolutional neural networks model for COVID-19 prediction using chest X-Ray images. J. Comput. Des. Eng. **9**, 343 (2022)
24. Schuld, M., Bergholm, V., Gogolin, C., Izaac, J., Killoran, N.: Evaluating analytic gradients on quantum hardware. Phys. Rev. A **99**, 032331 (2019)
25. Rechenberg, I. Evolutionsstrategien. Simulationsmethoden in der Medizin und Biologie, pp. 83–114. Springer, Berlin (1978)
26. Schwefel, H.P.: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie: mit einer vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie, vol. 1. Birkhäuser, Basel (1977)
27. Wierstra, D., et al.: Natural evolution strategies. J. Mach. Learn. Res. **15**, 949 (2014)
28. Crooks, G.E.: Gradients of parameterized quantum gates using the parameter-shift rule and gate decomposition. arXiv:1905.13311 (2019)
29. Salimans, T.: et al., Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864 (2017)
30. Yao, J., Bukov, M., Lin, L.: Policy gradient based quantum approximate optimization algorithm. arXiv:2002.01068 (2020)
31. Anand, A., Degroote, M., Aspuru-Guzik, A.: Natural evolutionary strategies for variational quantum computation. Mach. Learn. Sci. Technol. **2**, 045012 (2021)
32. Wilson, M., Stromswold, S., Wudarski, F., Hadfield, S., Tubman, N.M., Rieffel, E.: Optimizing quantum heuristics with meta-learning. Quantum Mach. Intell. **3**, 13 (2021)
33. Schuld, M., Sweke, R., Meyer, J.J.: The effect of data encoding on the expressive power of variational quantum machine learning models. Phys. Rev. A **103**, 032430 (2021)
34. Pérez-Salinas, A., Cervera-Lierta, A., Gil-Fuster, E., Latorre, J.I.: Data re-uploading for a universal quantum classifier. Quantum **4**, 226 (2020)
35. LaRose, R., Coyle, B.: Robust data encodings for quantum classifiers. Phys. Rev. A **102**, 032420 (2020)
36. Cerezo, M., Sone, A., Volkoff, T., Cincio, L., Coles, P.J.: Cost function dependent barren plateaus in shallow parametrized quantum circuits. Nat. Commun. **12**, 1 (2021)

37. Holmes, Z., Sharma, K., Cerezo, M., Coles, P.J.: Connecting ansatz expressibility to gradient magnitudes and barren plateaus. PRX Quantum **3**, 010313 (2022)
38. Wang, S., et al.: Noise-induced barren plateaus in variational quantum algorithms. Nat. Commun. **12**, 6961 (2021)
39. Marrero, C.O., Kieferová, M., Wiebe, N.: Entanglement-induced barren plateaus. PRX Quantum **2**, 040316 (2021)
40. Patti, T.L., Najafi, K., Gao, X., Yelin, S.F.: Entanglement devised barren plateau mitigation. Phys. Rev. Res. **3**, 033090 (2021)
41. Arrasmith, A., Cerezo, M., Czarnik, P., Cincio, L., Coles, P.J.: Effect of barren plateaus on gradient-free optimization. Quantum **5**, 558 (2021)
42. Friedrich, L., Maziero, J.: Avoiding barren plateaus with classical deep neural networks. Phys. Rev. A **106**, 042433 (2022)
43. Grant, E., Wossnig, L., Ostaszewski, M., Benedetti, M.: An initialization strategy for addressing barren plateaus in parametrized quantum circuits. Quantum **3**, 214 (2019)
44. Volkoff, T., Coles, P.J.: Large gradients via correlation in random parameterized quantum circuits. Quantum Sci. Technol. **6**, 025008 (2021)
45. Verdon, G. et al.: Learning to learn with quantum neural networks via classical neural networks. arXiv:1907.05415 (2019)
46. Skolik, A., McClean, J.R., Mohseni, M., van der Smagt, P., Leib, M.: Layerwise learning for quantum neural networks. Quantum Mach. Intell. **3**, 5 (2021)
47. McKay, D.C., et al.: Qiskit backend specifications for OpenQASM and OpenPulse experiments. arXiv:1809.03452 (2018)
48. Paszke, A., et al.: PyTorch: an imperative style, high-performance deep learning library. arXiv:1912.01703 (2019)