Improving gradient methods via coordinate transformations: Applications to quantum machine learning

Pablo Bermejo,[1,2,*] Borja Aizpurua,[1] and Román Orús[1,2,3,†]

[1]*Multiverse Computing, Paseo de Miramón 170, E-20014 San Sebastián, Spain*

[2]*Donostia International Physics Center, Paseo Manuel de Lardizabal 4, E-20018 San Sebastián, Spain*

[3]*Ikerbasque Foundation for Science, Maria Diaz de Haro 3, E-48013 Bilbao, Spain*

(Received 30 June 2023; accepted 3 April 2024; published 19 April 2024)

In this paper, we introduce a generic strategy to accelerate and improve the overall performance of machine-learning algorithms, both in their classical and quantum versions, heavily rely on optimization algorithms based on gradients, such as gradient descent. The overall performance is dependent on the appearance of local minima and barren plateaus, which slow down calculations and lead to nonoptimal solutions. In practice, this results in dramatic computational and energy costs for artificial intelligence applications. Our method is based on coordinate transformations, like variational rotations, adding extra directions in parameter space that depend on the cost function itself, and which allowus to explore the configuration landscape more efficiently. The validity of our method is benchmarked by boosting several quantum machine-learning algorithms, getting a very significant improvement in their performance.

DOI: 10.1103/PhysRevResearch.6.023069

I. INTRODUCTION

Machine learning is revolutionizing society. We have witnessed it recently with the advent of game-changing applications such as ChatGPT, where large language models [1] allow for unprecedented human-computer interaction. Such systems have a neural structure at their roots, with weights that must be optimized to minimize some error cost function. This optimization is usually done via gradient methods such as gradient descent, stochastic gradient descent, andadaptive moment estimation, which are not free from problems such as barren plateaus and local minima. One important consequence is that current artificial intelligence (AI) systems suffer from long and complex training procedures, amounting to dramatic and unsustainable computational and energy costs [2]. Given the increasingly huge demand of AI systems, the situation is even worse: We are in big need of more efficient machine learning.

Mathematically speaking, numerical algorithms used for optimization problems are mostly based on techniques that sweep the whole hyperspace of solutions. This landscape might present a tractable shape where the optimal solution can be easily found, as in convex problems. However, most interesting problems have an ill-defined landscape of solutions, with nonpredictable shapesand plenty of complexities impeding analytical and numerical methods to properly act on them. Gradient methods explore this landscape by computing local gradients of the cost function and updating the parameters according to the computed local slopes. More complex optimization methods, such as the widely used stochastic gradient descent and Adam optimizer, are based on the same idea.

Here, we address the two main caveats of employing gradient methods for optimization purposes: local minima and barren plateaus. These features emerge inherently due to the shape of the cost function. In fact, both limitations can be understood as coming from moving along certain directions in the landscape of solutions. Changes in the optimization variables are the result of changes in the cost function, as the variables are modified. This is evidently an obstacle when we find a flat region in the landscape of the cost function.

In this paper, we propose an alternative way to navigate the landscape of solutions by introducing extra freedom in the directions of the updateof the parameters. This is done by using (i) changes of coordinates and (ii) adding extra dimensions related to the cost itself. In our specific implementation, we consider changes to hyperspherical coordinates as well as frame rotations. To do that, we treat the cost as an extra variable to be optimized and implement a self-consistent variational method in which the cost axis not only represents the value to be optimized but also serves as an extra dimension to escape from local minima and barren plateaus. Importantly, we stress that our approach aims not to lower the cost function values but rather and specifically to move out from barren plateaus.

The structure of this paper is as follows. In Sec. II, we present our boosting methodology via coordinate transformations. In Sec. III, we show a set of benchmarks to validate our idea, where we systematically improve the performance of several well-known quantum machine-learning algorithms. Finally, in Sec. IV, we present our conclusions and discuss future directions.

*pablo.bermejo@dipc.org

†roman.orus@dipc.org

II. METHODOLOGY

An optimization problem can be stated as the minimization of a scalar real cost function $f(\vec{x})$ of n variables $\vec{x} \equiv (x_1, x_2, \ldots, x_n)$. Methods based on gradient descent apply an update of parameters mainly based on the calculation of changes in the function when there is a change in the coordinate values, i.e.,

$$\Delta f(\vec{x}) = \left(\frac{\partial f}{x_1}, \frac{\partial f}{x_2}, \ldots, \frac{\partial f}{x_n}\right)\Big|_{\vec{x}_0} \cdot \Delta\vec{x} = \vec{\nabla} f(\vec{x})|_{\vec{x}_0} \cdot \Delta\vec{x}, \quad (1)$$

where $\Delta\vec{x} = \vec{x}_1 - \vec{x}_0$ for some arbitrary $\vec{x}_1$, and $\vec{x}_0$ is the point in parameter space where the gradient $\vec{\nabla} f(\vec{x})$ of the function is computed. As is clear from this equation, a gradient close to zero represents flat regions in the landscape of solutions, implying no updates in the parameters, and therefore disabling further optimization steps, so that the optimization gets stuck.

To avoid that, we propose a method based on a change of coordinates. Our idea can be well understood by considering the case of optimization of a one-dimensional cost function, represented by the landscape in Fig. 1. In Fig. 1(a), the cost function has a plateau, and therefore, the gradient is zero, so that optimization is stalled. However, in Fig. 1(b), we see that we can avoid this by changing the polar coordinates of point P in the landscape, i.e., in the two-dimensional(2D) plane formed by the axis corresponding to the coordinate x to be optimized and the cost function $f(x)$. The new point P' can then be collapsed back to the landscape of the cost function, so that we can escape from the null-gradient region. Alternatively, we can also rotate the frame, as in Fig. 1(c). In the new rotated frame, the rotated cost function $f_r(x_r)$ does not present a null gradient with respect to the rotated parameter x_r, so that moving by gradient methods is again possible in the new frame.

While the above is a simple picture explaining the basic idea of our algorithm, it captures many of its basic ingredients. Long story short, if the problem coordinates do not work, then we change them by including the cost as an extra coordinate. This change of coordinates can be a rotation or a more generic one.

Let us now explain how to implement our method in full generality. As sketched above, there are two versions of it: changing to hyperspherical coordinates or rotating the frame. As we shall see, both approaches are similar though not equivalent. Let us start with the change to hyperspherical coordinates.

A. Option 1: Hyperspherical coordinates

Consider the cost function $f(\vec{x})$ and the original n Cartesian coordinates $\vec{x}$. In the landscape space, this defines a point P in the $(n+1)$-dimensional space described by the coordinates and the cost function:

$$P = [x_1, x_2, \ldots, x_n, f(\vec{x})]. \quad (2)$$

The procedure to follow in one iteration step is as follows:

(1) Make a change of coordinates. Without loss of generality, we will use $(n+1)$-dimensional hyperspherical coordinates. Therefore, make the change:

$$P = [x_1, x_2, \ldots, x_n, f(\vec{x})] \rightarrow P = [\theta_1, \theta_2, \ldots, \theta_n, r], \quad (3)$$

with $\{\vec{\theta}, r\}$ being the $n+1$ hyperespherical coordinates for point P.

(2) Perform gradient descent on the new set $\{\vec{\theta}, r\}$ with reference cost value $f(\vec{x})$. In practice, we will simply update the variables in the hyperspherical coordinates using changes of the cost function in the original Cartesian description (the one we want to minimize) each time we update these parameters.

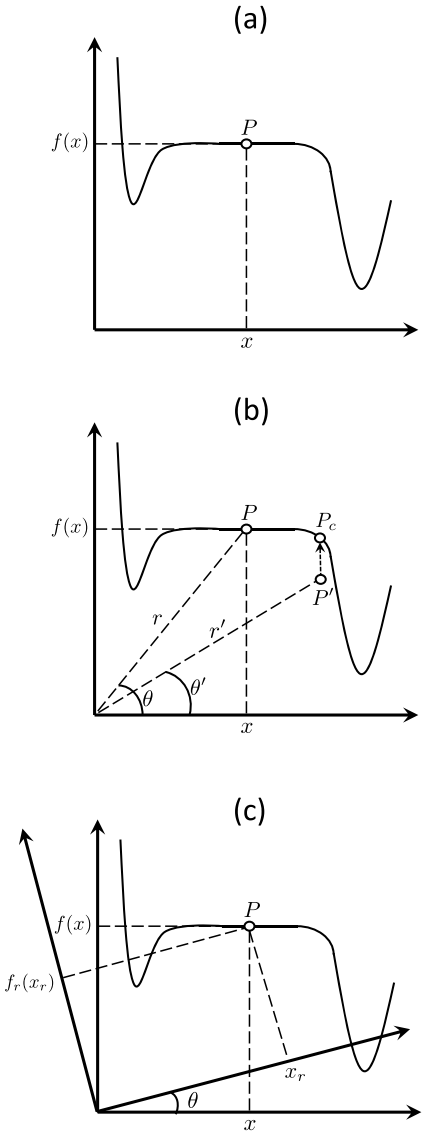

FIG. 1. (a) Cost function $f(x)$ presents a plateau as a function of optimization variable x, so that gradient methods get stalled at point P due to a null gradient. (b) A change in the polar coordinates of point P leads to a point P', which can then be collapsed back to the landscape of the cost function, leading to point P_c. Optimization from point P_c is no longer stalled since the gradient is nonzero. (c) A frame rotation leads to a description of point P with different Cartesian coordinates $f_r(x_r)$ and x_r. In the new rotated frame, the gradient at P is nonzero, so that gradient methods are no longer stalled.

We can do so by projecting each point in the hyperspherical description into the Cartesian cost function to obtain that reference value.

This translates into an iterative switching process from one description to the other to be able to move in the hyperspherical coordinates while the changes in the original cost function are the ones driving the parameter updates.

This procedure will end up giving us a new point in the $(n+1)$-dimensional plane, defined by the change $\{\vec{\theta}, r\} \rightarrow \{\vec{\theta}', r'\}$ after all the parameters have been updated:

$$P' = [\theta_1', \theta_2', \ldots, \theta_n', r']. \tag{4}$$

(3) Once the gradient descent is completed, we can make a final transformation $\{\vec{\theta}', r'\} \rightarrow \{\vec{x}_c, f_c(\vec{x}_c)\}$, retrieving the point P_c which is just defined by the projection of point P' into the original cost function:

$$P_c \equiv [x_1', x_2', \ldots, x_n', f'(\vec{x}')], \tag{5}$$

where c stands for collapsed.

These steps are to be repeated until convergence, as in usual methods based on gradient descent, with the difference that the gradient is computed via the change of coordinates. Additionally, this change of coordinates can be implemented only when we fall in local minima or barren plateaus to move out from the region or alternate between the two during the optimization.

B. Option 2: Plane rotations

An alternative to the change to hyperspherical coordinates is to rotate the frame. Consider again the cost function $f(\vec{x})$ and the original n Cartesian coordinates $\vec{x}$. Following Eq. (2), this generates a point P in the $(n+1)$-dimensional space defined by the coordinates and the cost function. This point can also be described as a vector $\vec{P}$ in the considered frame. The procedure to follow this time in one iteration step is then as follows:

(1) Make a change of coordinates in the frame. Without loss of generality, we will use a $(n+1)$-dimensional rotation of the axis:

$$\vec{P} = [x_1, \ldots, x_n, f(\vec{x})] \rightarrow \vec{P}_r = R \cdot \vec{P}, \tag{6}$$

with R the $(n+1)$-dimensional axis rotation and subscript r standing for rotated. A convenient simplification is to implement a 2D rotation of the plane formed by some individual coordinate x_i and the function $f(\vec{x})$, i.e.,

$$\begin{aligned} \vec{P} &= [x_1, \ldots, x_i, \ldots, x_n, f(\vec{x})] \\ \rightarrow \vec{P}_r &= R_2 \cdot \vec{P} = [x_1, \ldots, x_{ir}, \ldots, x_n, f_r(\vec{x}_r)] \end{aligned} \tag{7}$$

where subindex r indicates the rotated variables, and with $R_2 \in \mathrm{SO}(2)$, the 2×2 matrix corresponds to a rotation of the $[x_i, f(\vec{x})]$ plane. This is a good choice if we think that there may be a barren plateau or local minima in the direction of coordinate x_i. Additionally, the rotation angle may not be necessarily fixed, so that it can work as an extra hyperparameter in the optimization, which has been the case for most of our results. For simplicity, and without loss of generality, from now on, we assume such a 2D rotation.

(2) In the rotated frame, perform a similar procedure to the one explained in point (2) from Sec. II A to compute the gradient descent. Now instead of switching from one description to the other by means of projections from the updated parameters to the original cost function, we will simply move from one to the other by means of relative rotation matrices.

This will lead us to a final point, after all the updates have been performed, given by the inverse rotation R_2^{-1}, recovering a new updated cost value $f_u(\vec{x}_u)$:

$$\vec{P}_u = R_2^{-1} \cdot \vec{P}_r' = [x_{1u}, x_{2u}, \ldots, x_{nu}, f_u(\vec{x}_u)]. \tag{8}$$

Notice that we use subindex u to make explicit the fact that now the change of coordinates does not come from a projection, so there is no collapsed cost function, but itis instead directly obtained from relative matrix rotations.

Again, these steps are to be repeated until convergence, as in usual methods based on gradient descent. Furthermore, one could implement the rotations just when falling in local minima or barren plateaus to move out from the region or alternate between the two frames during the optimization.

It is also worth noticing that placing the rotating point elsewhere does not affect the performance of the method. All in all, computing the gradients with respect to any other point in the parameter space while retaining description of parameters with respect to the original should not give either any improvement.

The two methods presented here make use of a higher-dimensional space, where the cost function is included as an extra dimension and is optimized self-consistently. One may be tempted to say that all the advantage of our method comes from extending the variables to a higher-dimensional space. While this is tempting, it is actually not true. We have also tested implementations where we just extended the coordinate space but without adding the cost function as an extra coordinate, in a way similar to kernel methods. In such implementations, we have seen no significant advantage in convergence, in stark contrast with the two methods presented above. We therefore conclude that including the cost function as an extra coordinate to be optimized self-consistently is in fact fundamental. We want to remark that, physically, in the parameter space, we are not introducing any new dimension. The parameter landscape is built upon several variables which are combined in a certain manner to define a cost function. Thus, evaluating these parameters at a given point gives some cost value. We are proposing to use that same cost function, which has support on a given parameter space, to add one more dimension through which we can navigate to optimize the parameter search.

Concerning the learning rate, upon applying coordinate transformations to escape these plateaus, the optimization landscape undergoes a significant change. The scaleof the gradients can be substantially different in the new coordinates, which implies that the previously optimal learning rate may no longer be effective. A higher learning rate, in theory, can indeed introduce a greater degree of perturbation in the parameters, facilitating the escape from a plateau. However, this must be balanced carefully; too high a learning rate can lead to overshooting minima or increased oscillations in the training process, potentially destabilizing convergence. This is also aligned with the fact that thelearning rate is a

TABLE I. Advantage in number of iterations for several algorithms, with respect to the original implementation using Xanadu's PennyLane, and where the new implementation uses the change to hyperspherical coordinates. Numerical data are the average number of iterations.

Paper	Original	Hyperspherical
Alleviating barren plateaus [5]	29.14	4.7
Accelerating VQEs with quantum natural gradient: single-qubit VQE/hydrogen VQE (Adam) [6]	182/103	160/62
Function fitting using quantum signal processing—polynomial [7]	90	54
Variational classifier Iris classification [8]	100	30
Variational quantum thermalizer [9]	103	51

problem-dependent parameter. It usually requires a preliminary grid search to find the optimal learning rate for each use case. Here, we are altering the optimization process itself so, in the same way thelearning rate is problem dependent, one can think that any modification in the optimization process will require further tuning of important hyperparameters as the learning rate.

In what follows, we show the benefits of our approach by implementing several benchmarks.

III. RESULTS

For the purpose of implementation and testing, a set of quantum machine-learning algorithms has been used for the sake of comparison, with the original implementation being done via PennyLane [3]. Some of these algorithms introduce new methods for solving optimization processes, as it is the case of quantum natural gradient descent [4], or the construction of local cost functions in Ref. [5]. Some other methods propose optimization algorithms for solving certain tasks.

Here, we show how our algorithm boosts the performance of all the tested methods. For the purpose of the analysis, we have revisited 18 different algorithms, claiming significant advantages (i.e., faster convergence or lower cost function values) for 15 of them. Let us also stress that this improvement is obtained against optimal and finetuned implementations of the original algorithms using PennyLane and not merely naïve ones. This also accounts for using the most efficient optimizer for each case, which happens to be mostly Adam optimizer.

Let us start by showing in Table I a summary of our results for the method using the change to hyperspherical coordinates. As we see in the table, the change to hyperspherical allows for substantially faster convergence in five quantum machine-learning algorithms, with respect to their original optimal implementation. Let us now describe four of these examples.

In Ref. [5], it has been shown that local cost functions with a shallow quantum circuit improve the trainability of variational quantum algorithms. Global cost functions show barren plateaus that cannot be overcome by usual means, whereas local cost functions improve the convergence. In this case, we implemented simulations for a set of 50 different initial configurations to solve a certain optimization problem. The original global approach showed 30% success in overcoming barren plateaus with >100 iterations on average for convergence. Local cost functions showed a ratio of success of 98% with an average number of iterations equal to 29.14. By using our approach with hyperspherical coordinates, we reach a success of 100% with 4.7 iterations on average, implying a very significant improvement, see Fig. 2.

Next, in Ref. [7], the idea of quantum signal processing to perform an optimization process for function fitting was

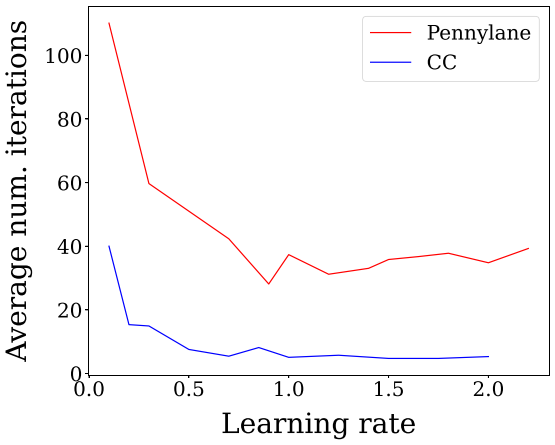

FIG. 2. Improvement in the average number of iterations as a function of the learning rate, for the algorithm in Ref. [5] to alleviate barren plateaus with local cost functions. Comparison between PennyLane and the change of coordinates (CC) implementations.

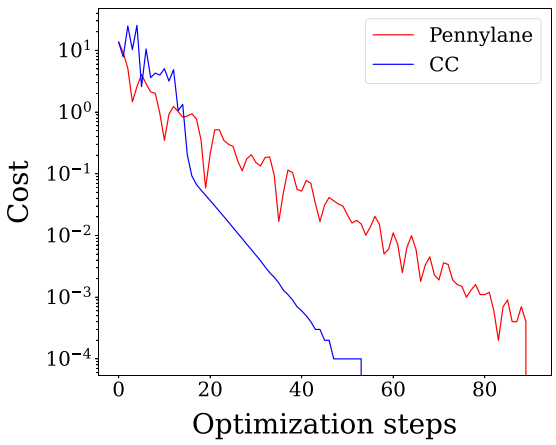

FIG. 3. Convergence of cost function vs number of optimization steps, in function fitting using a quantum signal processing polynomial, for the algorithm in Ref. [7]. Comparison between PennyLane and the change of coordinates (CC) implementations.

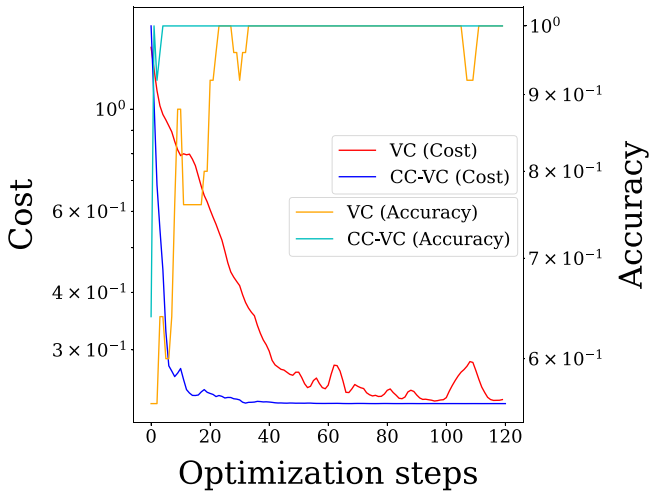

FIG. 4. Convergence of cost function and accuracy vs number of optimization steps, in the variational quantum classifier from Ref. [8] for the Iris dataset. Comparison between PennyLane and the change of coordinates (CC) implementations.

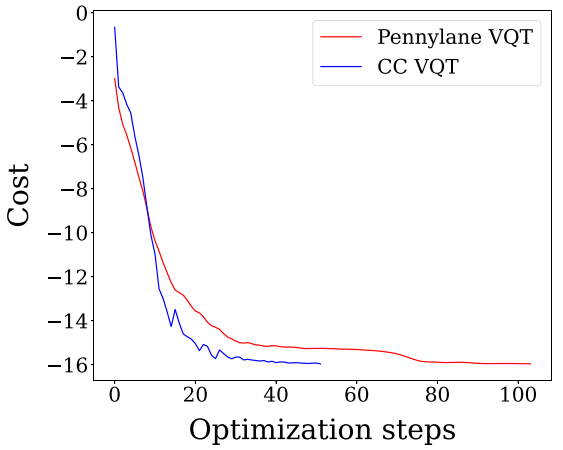

FIG. 5. Convergence of cost function vs number of optimization steps, in the variational quantum thermalizer from Ref. [9]. Comparison between PennyLane and the change of coordinates (CC) implementations.

considered. Again, we have observed that our implementation using hyperspherical coordinates provides a speed up in optimization steps for a faster convergence to lower values of the cost. To reach a limiting cost value of 10^{-5}, we were able to reduce the number of iterations for convergence from 90 in the original implementation to 54 in ours; this is a reduction of 40% of iterations, see Fig. 3.

Additionally, in Ref. [8], the authors proposed a quantum neural network based on a variational quantum circuit to perform classification of a dataset. We plot in Fig. 4 the convergence of the classification procedure for a specific quantum circuit configuration (namely, a fixed ansatz with the same number of qubits and layers) to classify the Iris dataset. The plot shows that our approach with hyperspherical coordinates significantlyimproves the convergence stability of the algorithm while reducing the number of steps to achieve convergence, both in the accuracy of the classification as well as in the cost function to be optimized. For instance, to achieve a similar value of the cost of ≈0.23, we reduce the number of steps from ∼100 to ∼30, implying a speedup of 70% with the new implementation. From this example, we also see that our new implementation provides a smoother convergence. Avoiding randomness and drastic variations in the optimizing curve can also become a concern for certain applications, and we understand this as a sign that our implementation fits the problem landscapebetter.

As a last example for the hyperspherical approach, we consider the algorithm in Ref. [9]. The authors proposed a generalization of the variational quantum eigensolver to compute thermal states via a hybrid optimization procedure. Again, as shown in Fig. 5, we can reduce the number of optimization steps to find the thermal states. We reach convergence in almost half the number of optimization steps, reducing from 103 in the original implementation to just 51 in ours.

Next, let us consider our second approach, namely, the frame rotations. Our results are summarized in Tables II and

TABLE II. Advantage in number of iterations for several algorithms, with respect to the original implementation using Xanadu's PennyLane, and where the new implementation uses frame rotations. Numerical data are the average number of iterations.

Paper	PennyLane	Rotational axis
Alleviating barren plateaus [5]	49-1/29.14	50-0/24.92
Quantum natural gradient (QNG/Adam) and (GD/Adam) [4]	26/88	19/75
Accelerating VQEs with quantum natural gradient: hydrogen VQE (QNG/GD) and (GD) [6]	20/66	16
Accelerating VQEs with quantum natural gradient: single-qubit VQE/hydrogen VQE (Adam) [6]	182/103	158/98
Function fitting using quantum signal processing—polynomial/nonpolynomial [7]	90/112	74/94
Variational quantum thermalizer [9]	103	51
Function fitting with photonic quantum neural network [10]	70	51
Perturbative gadgets for variational quantum algorithms [11]	140	110
Variational classifier Iris classification [8]	96	80
VQE in different spin sectors $S=0$ $S=1$ [6]	166/54	142/34

TABLE III. Advantage in cost function value for several algorithms, with respect to the original implementation using Xanadu's PennyLane, and where the new implementation uses frame rotations. Numerical data are the average number of iterations.

Paper	PennyLane	Rotational axis
VQCO: 8 qubits, 80 layers [12]	1.384×10^{-4}	6.942×10^{-5}
Quantum models as Fourier series [13]	9.67×10^{-13}	1.55×10^{-13}
Variational classifier Iris classification [8]	0.2323	0.2303
Variational quantum linear solver [14]	5.83×10^{-10}	1.16×10^{-10}

III, showing examples of improvement in the number of iterations and in the cost function value, respectively. The method of frame rotations allows for substantially faster convergence in 10 quantum machine-learning algorithms and lower cost functions in three, with respect to their original optimal implementation. Let us briefly sketch three of these examples.

In Ref. [14], the authors proposed an algorithm to solve linear systems of equations by using a hybrid quantum-classical approach suitable for near-term quantum devices. We have compared this approach with two different configurations of our rotation method, where rotations were implemented for different planes. As shown in Fig. 6, depending on the rotated plane, we can achieve faster convergence as well as lower values of the cost functionthan the original approach.

Next, Ref. [13] describes an optimization process built to test the expressive power of specific quantum circuits and their ability to fit certain functions by learning features defining their Fourier series. Our comparison is shown in Fig. 7. Even if the final cost is tiny for both cases, we can observe in the plot that the computed cost function in our approach is almost always lower than the one computed with the original method.

Additionally, in Ref. [6], the authors use a variational quantum eigensolver to find the lowest-energy state of the hydrogen molecule, directly fixing three different spin sectors. In Fig. 8, we plot the results for the case of total spin $S = 1$, showing a significant advantage of our method both in smoothness and number of iterations to converge to a certain cost value. To find the ground state of the molecule, our implementation needs 34 iterations, in sharp contrast to the 54 needed by the original method.

Also, in Ref. [11], the authors proposed to substitute the original Hamiltonian describing our quantum systems/cost function in variational quantum algorithms, by another one called agadget Hamiltonian. The idea is that the gadget Hamiltonian is built from local interactions, so that the landscape of solutions is less likely to present local minima and barren plateaus during the optimization. In this case, our implementation can reduce the number of optimization steps from 140 in the original method to 110 by using frame rotations, as shown in Fig. 9.

Last but not least, in Ref. [12], the authors introduced a variational quantum optimizer to tackle optimization problems with continuous variables, by encoding these variables inside the three continuous parameters within the Bloch sphereof each qubit. In Fig. 10, we show how the error in performing the integral of the function $\exp(-x^2)$, implemented by encoding the solution in Fourier modes. We see that the new approach reduces significantly the optimization steps for an arbitrary error bound.

IV. CONCLUSIONS AND FURTHER WORK

In this paper, we have proposed a boosting strategy for machine-learning algorithms. Our idea is based on using rotation matrices involving the parameters of the

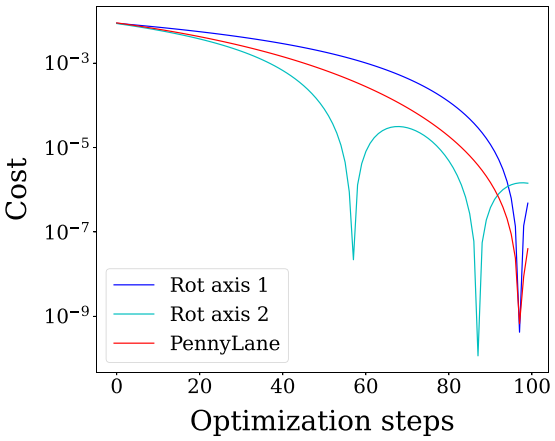

FIG. 6. Convergence of cost function vs number of optimization steps, in the variational quantum linear solver from Ref. [14]. Comparison between PennyLane and the rotation implementations, where we implemented two rotation approaches.

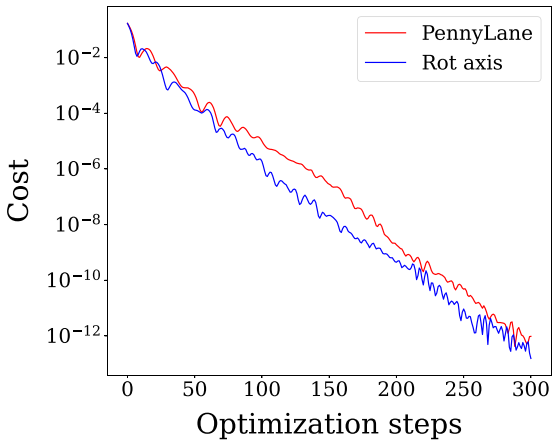

FIG. 7. Convergence of cost function vs number of optimization steps, in the quantum models for Fourier series from Ref. [13]. Comparison between PennyLane and the rotation implementations.

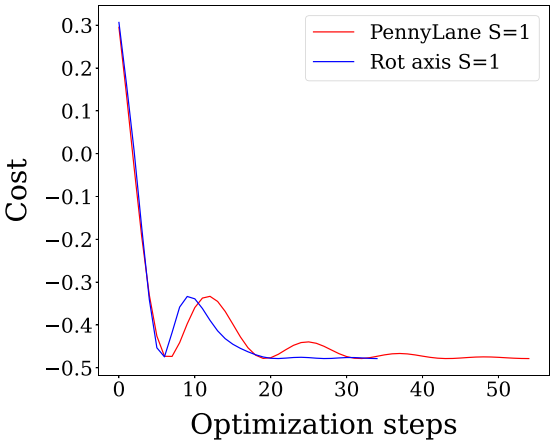

FIG. 8. Convergence of cost function vs number of optimization steps, in the variational quantum eigensolver for different spin sectors from Ref. [6]. Comparison between PennyLane and the rotation implementations.

hyperspace of solutions, the cost, and alternatively, the use of hyperspherical coordinates inside the hyperspace of solutions in which the axis of the cost is also included. The procedure allowsus to alleviate the effect of many local minima and barren plateaus, thus accelerating the convergence of machine-learning methods.

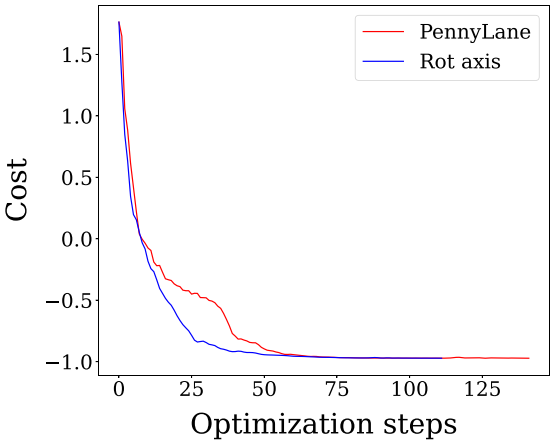

FIG. 9. Convergence of cost function vs number of optimization steps, for the perturbative gadgets for variational quantum algorithms from Ref. [11]. Comparison between PennyLane and the rotation implementations.

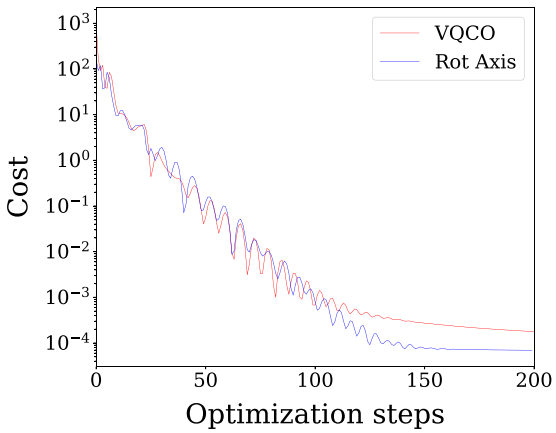

FIG. 10. Convergence of cost function vs number of optimization steps, for the variational quantum continuous optimizer from Ref. [12]. Comparison between PennyLane and the rotation implementations.

The procedure discussed in this paper is completely general and can be applied both to classical and quantum machine-learning algorithms. According to our benchmarks, the improvement in convergence, stability, and efficiency can be very significant. As such, we believe that our boosting procedure has a huge potential to improve the performance of AI systems, in turn reducing its computational cost and energy consumption.

There are different lines of work that can be explored in the future. For instance, it would be interesting to test different changes of coordinates, even dynamical and nonlinear ones. Moreover, the performance of the boosting should be further tested in the context of complex deep-learning algorithms such as convolutional neural networks, transformers, and generative models, which are computationally quite expensive. In addition, it would also be interesting to explore the performance of this boosting technique in the context of Tensor Network methods [15].

ACKNOWLEDGMENTS

The authors acknowledge Donostia International Physics Center (DIPC), Ikerbasque, Basque Government, and Diputación de Gipuzkoa for constant support as well as insightful discussions with the technical teams from Multiverse Computing and DIPC on the algorithms and technical implementations. We also acknowledge Norman Toporcer from Elion for supporting us with several patent applications in connection with parts of the work discussed here.

[1] C. D. Manning, Human language understanding and reasoning, Daedalus **151**, 127 (2022).
[2] The race of the AI labs heats up, The Economist, https://www.economist.com/business/2023/01/30/the-race-of-the-ai-labs-heats-up (2023).
[3] See, e.g., PennyLane Library, https://pennylane.ai.
[4] J. Stokes, J. Izaac, N. Killoran, and G. Carleo, Quantum natural gradient, Quantum **4**, 269 (2020).
[5] M. Cerezo, A. Sone, T. Volkoff, L. Cincio, and P. J. Coles, Cost function dependent barren plateaus in shallow parametrized quantum circuits, Nat. Commun. **12**, 1791 (2021).
[6] N. Yamamoto, On the natural gradient for variational quantum eigensolver, arXiv:1909.05074.
[7] J. M. Martyn, Z. M. Rossi, A. K. Tan, and I. L. Chuang, Grand unification of quantum algorithms, PRX Quantum **2**, 040203 (2021).

[8] E. Farhi and H. Neven, Classification with quantum neural networks on near term processors, arXiv:1802.06002.

[9] G. Verdon, J. Marks, S. Nanda, S. Leichenauer, and J. Hidary, Quantum Hamiltonian-based models and the variational quantum thermalizer algorithm, arXiv:1910.02071.

[10] N. Killoran, T. R. Bromley, J. M. Arrazola, M. Schuld, N. Quesada, and S. Lloyd, Continuous-variable quantum neural networks, Phys. Rev. Res. **1**, 033063 (2019).

[11] S. Cichy, P. K. Faehrmann, S. Khatri, and J. Eisert, Non-recursive perturbative gadgets without subspace restrictions and applications to variational quantum algorithms, arXiv:2210.03099.

[12] P. Bermejo and R. Orús, Variational quantum continuous optimization: a cornerstone of quantum mathematical analysis, arXiv:2210.03136.

[13] M. Schuld, R. Sweke, and J. J. Meyer, Effect of data encoding on the expressive power of variational quantum-machine-learning models, Phys. Rev. A **103**, 032430 (2021).

[14] C. Bravo-Prieto, R. LaRose, M. Cerezo, Y. Subasi, L. Cincio, and P. J. Coles, Variational quantum linear solver, Quantum **7**, 1188 (2023).

[15] R. Orús, A practical introduction to tensor networks: Matrix product states and projected entangled pair states, Ann. Phys. **349**, 117 (2014).