

# Learning To Optimize Quantum Neural Network Without Gradients

Ankit Kulshrestha

*Department of Computer and Information Sciences)*  
*University of Delaware*  
*Newark, USA*  
*akulshr@udel.edu*

Xiaoyuan Liu

*Fujitsu Research of America*  
*Sunnyvale, USA*  
*xliu@fujitsu.com*

Hayato Ushijima-Mwesigwa

*Fujitsu Research of America*  
*Sunnyvale, USA*  
*hayato@fujitsu.com*

Ilya Safro

*Department of Computer and Information Sciences*  
*University of Delaware*  
*Newark, US*  
*isafro@udel.edu*

**Abstract**—Quantum Machine Learning is an emerging sub-field in machine learning where one of the goals is to perform pattern recognition tasks by encoding data into quantum states. This extension from classical to quantum domain has been made possible due to the development of hybrid quantum-classical algorithms that allow a parameterized quantum circuit to be optimized using gradient based algorithms that run on a classical computer. The similarities in training of these hybrid algorithms and classical neural networks has further led to the development of Quantum Neural Networks (QNNs). However, in the current training regime for QNNs, the gradients w.r.t objective function have to be computed on the quantum device. This computation is highly non-scalable and is affected by hardware and sampling noise present in the current generation of quantum hardware. In this paper, we propose a training algorithm that does not rely on gradient information. Specifically, we introduce a novel meta-optimization algorithm that trains a *meta-optimizer* network to output parameters for the quantum circuit such that the objective function is minimized. We empirically and theoretically show that we achieve a better quality minima in fewer circuit evaluations than existing gradient based algorithms on different datasets.

## I. INTRODUCTION

Machine learning has evolved over time from solving small size pattern recognition problems to being able to capture latent structure in data and generalize to unseen data at scale. The state of machine learning is quickly approaching a point where classical computers would not be able to keep up with the ever increasing scale and complexity of data. On the other hand, quantum computing holds a theoretical promise of being able to scale beyond existing classical approaches. While the current state of quantum machines is still far from being practically competitive to the classical counterparts, the emergent field of Quantum Machine Learning (QML) is an important step towards the future in which quantum computers form the basis of many challenging computational tasks [1], [2].

The current bridge between classical and quantum algorithms is a class of algorithms called Variational Quantum Algorithms (VQAs) [3]. These algorithms are concerned with finding the

optimal set of parameters for a Variational Quantum Circuit (VQC) to minimize an objective function. The parameters are optimized using gradient descent which runs on a classical computer. Since the training procedure is very similar to modern Deep Neural Networks (DNNs), quantum variants of several existing DNN architectures have been proposed [4], [5], [6], [7]. The classical and quantum neural networks differ in one key aspect: the data has to be encoded as quantum states before it can be processed by a QNN. This encoding can be seen as a mapping to a high dimensional Hilbert space where the data is separable. It is thus entirely possible that with a powerful quantum computer, we will be able to capture a much richer representation of data and consequently outperform DNNs which is the motivation and hope behind several QML algorithms. The VQCs are deployed on quantum devices that support operations on quantum states. The current generation of devices, called Noisy Intermediate Scale Quantum (NISQ) devices, is quite limited due to a high presence of gate noise, inability to scale to beyond a few hundred qubits, and lack of reliable error mitigation and detection algorithms [3]. These bottlenecks directly affect the performance of QNNs and hence motivate better algorithms to train them.

**The need for gradient-free optimization algorithms:** The bottlenecks in current generation of quantum devices are not the only reasons that motivate a better optimization algorithm. Other factors also make the case for a better optimization algorithm more compelling. For instance, for a QNN running on a quantum device, the gradients are computed using the parameter shift rule [8]. This method scales as  $O(N)$  where  $N$  is the number of parameters of the QNN. A full forward-backward pass then scales as  $O(N^2)$ . Clearly, if the QNN is to be scaled to large problem instances (e.g., Imagenet [9] classification) the quadratic cost of computation must be improved.

Another contributing reason is that in the current training regime, the QNN is treated as a black-box from the perspective of the optimizer. While conventional optimization algorithms

like RMSProp [10], Adam [11], SGD etc. still work for QNNs, it is possible that when QNNs are scaled to large problem instances, hand-designed optimization rules may not be able to fully capture the complexities of the probabilistic nature of QNNs. These reasons, coupled with inherent issues in NISQ devices motivate the development of efficient learned optimizers. More crucially, in order to be broadly applicable we demand that the optimizer makes as few calls to the quantum circuit as possible and estimate the direction of optimization in a gradient free manner.

One way of designing a new learning rule is to *learn* it during training for a fixed task and dataset. If  $\theta^t \in \mathbb{R}^N$  are the parameters of QNN at timestep  $t$  and  $C(\theta)$  is a cost function we are interested in minimizing, then a learned update rule is of the form  $\theta^{t+1} = \mathcal{R}_\Phi(\theta^t, \nabla_\theta C(\theta^t))$  where  $\nabla_\theta C(\theta^t)$  is the gradient of QNN cost w.r.t  $\theta^t$ . Here,  $\mathcal{R}_\Phi$  is a DNN parameterized by  $\Phi \in \mathbb{R}^M$  where  $M$  is the number of parameters in the DNN.  $\mathcal{R}_\Phi$  is trained using a meta-loss function  $\mathcal{L}(\Phi)$ . For any given dataset and QNN, we are interested in finding an optimal set of meta-parameters  $\Phi^*$  by minimizing  $\mathcal{L}(\Phi)$  such that the QNN cost  $C(\theta)$  is minimized. For the rest of the paper, we refer to  $\mathcal{R}_\Phi$  as a *meta-optimizer* and the problem of minimizing  $\mathcal{L}(\Phi)$  as meta-optimization.

Meta-optimization approaches are well studied in the deep learning literature. Andrychowicz *et al.* [12] introduced the idea of optimizing a deep neural network using an LSTM based meta-optimizer that accepted  $(\theta^t, \nabla_\theta C(\theta^t))$  as inputs. Li and Malik [13], [14] explore the idea from the reinforcement learning perspective where they leverage the LSTM based optimizer to learn a *policy* for predicting the next set of parameters. Metz *et al.* [15] propose an alternative algorithm for fast training of meta-optimizers that generalizes better than the gradient descent training.

These findings have been inherited for training variational quantum circuits (VQCs) in the works of Wilson *et al.* [16] and Verdon *et al.* [17]. In the former work, the authors build on the algorithm in [12] for a VQC and leverage  $\nabla_\theta C(\theta)$  as an input feature for the meta-optimizer. In the latter work, the authors propose using the meta-optimization framework for learning initial parameters using a recurrent neural network and then initializing a VQC with these learned parameters. The VQC is then proposed to be fine-tuned by a regular optimizer until a desired accuracy level is reached. In all the aforementioned works, a common denominator is the reliance over the gradient as input features to the meta-optimizer (except [17] which uses  $C(\theta)$ ). Given that computing gradients for QNNs is not cheap, the challenge is to develop a meta optimizer that can learn a proper update rule *without* any explicit gradient computation step.

**Our Contribution** We address this critical challenge and propose a novel meta-optimization algorithm that learns to train QNNs without relying on any gradient information. More specifically, we make the following contributions:

(1) We design a novel algorithm for training QNNs using a meta-optimizer in a way that does not involve any computation or approximation of  $\nabla_\theta C(\theta)$  on a quantum device. We show

that using better features and training schedule can result in a meta-optimizer which is competitive with gradient based optimizers that are currently used for training QNNs.

(2) We theoretically and empirically verify that our algorithm provides a significant speedup in training time over conventional gradient based optimization algorithms.

(3) We empirically demonstrate that our algorithm achieves a minima which is comparable to that attained by the conventional first-order methods. Moreover, we show that with right initialization our algorithm can outperform classical gradient based algorithms with the same initialization.

To the best of our knowledge, this is a first meta-optimization algorithm for training QNNs that does not rely on computing gradients. We advocate that our algorithm can serve as blueprint for gradient-free meta-optimization algorithms.

## II. QUANTUM NEURAL NETWORKS

For completeness, we first introduce the idea of Quantum Neural Networks (QNNs) [18], [4]. Given a dataset consisting of  $m$  samples  $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^m$ , where  $\mathbf{x}_i \in \mathbb{R}^d$  are the data points with corresponding labels  $y_i$ <sup>1</sup>. A quantum neural network  $f_\theta : \mathbb{R}^d \mapsto \mathbb{R}$  is a variational quantum circuit that learns to assign  $y$  to  $\mathbf{x} \in \mathbb{R}^d$ . A parameterized quantum circuit implementing a quantum neural network can be written as a product of  $n$  parametrized unitary matrices  $\mathbf{U}_i(\theta_i)$ ,  $\mathbf{U}(\theta) = \prod_{i=1}^n \mathbf{U}_i(\theta_i)$ . Before the data can be processed by the VQC, an encoding circuit is applied to prepare the input state  $|\psi\rangle = \mathbf{U}_0(\mathbf{x})|0\rangle$ , where  $\mathbf{U}_0(\mathbf{x}) = e^{-i\mathbf{x}\mathbf{G}}$  is a unitary that encodes each component of a single  $d$  dimensional data vector into a quantum state consisting of  $q = \log(d)$  qubits. Here,  $\mathbf{G}$  is a gate generating Hermitian matrix. Overall we can describe the QNN  $f_\theta(\mathbf{x}, \theta)$  as:

$$f_\theta(\mathbf{x}, \theta) = \langle 0 | \mathbf{U}_0^\dagger(\mathbf{x}) \mathbf{U}^\dagger(\theta) \hat{\mathbf{O}} \mathbf{U}(\theta) \mathbf{U}_0(\mathbf{x}) | 0 \rangle, \quad (1)$$

where  $\hat{\mathbf{O}}$  is a quantum observable that maps the output quantum state to a scalar number and  $\mathbf{U}^\dagger$  are the complex conjugate of the unitary matrix  $\mathbf{U}$ . We select the minimum squared error as the cost function of our choice. Averaged over all  $m$  points of the dataset, we express it as:

$$C(\theta) = \frac{1}{m} \sum_{i=1}^m \|y_i - f_\theta(\mathbf{x}_i, \theta)\|^2 \quad (2)$$

The objective is to find  $\theta^* = \text{argmin}_\theta C(\theta)$ . In this work we shall assume that the QNN is minimizing the cost function,  $C(\theta)$ , in Equation (2).

## III. META-OPTIMIZATION FRAMEWORK

We now discuss the meta-optimization framework in more detail. For simplicity, we adopt a similar nomenclature as in [12] and refer to the QNN as the *optimizee* network and the meta-optimizer as the *optimizer* network.

In the meta-optimization framework, the task of the optimizee network is to evaluate the parameters suggested by the optimizer network. In turn, the optimization network accepts

<sup>1</sup>The index subscript will be omitted where it is clear from the context.

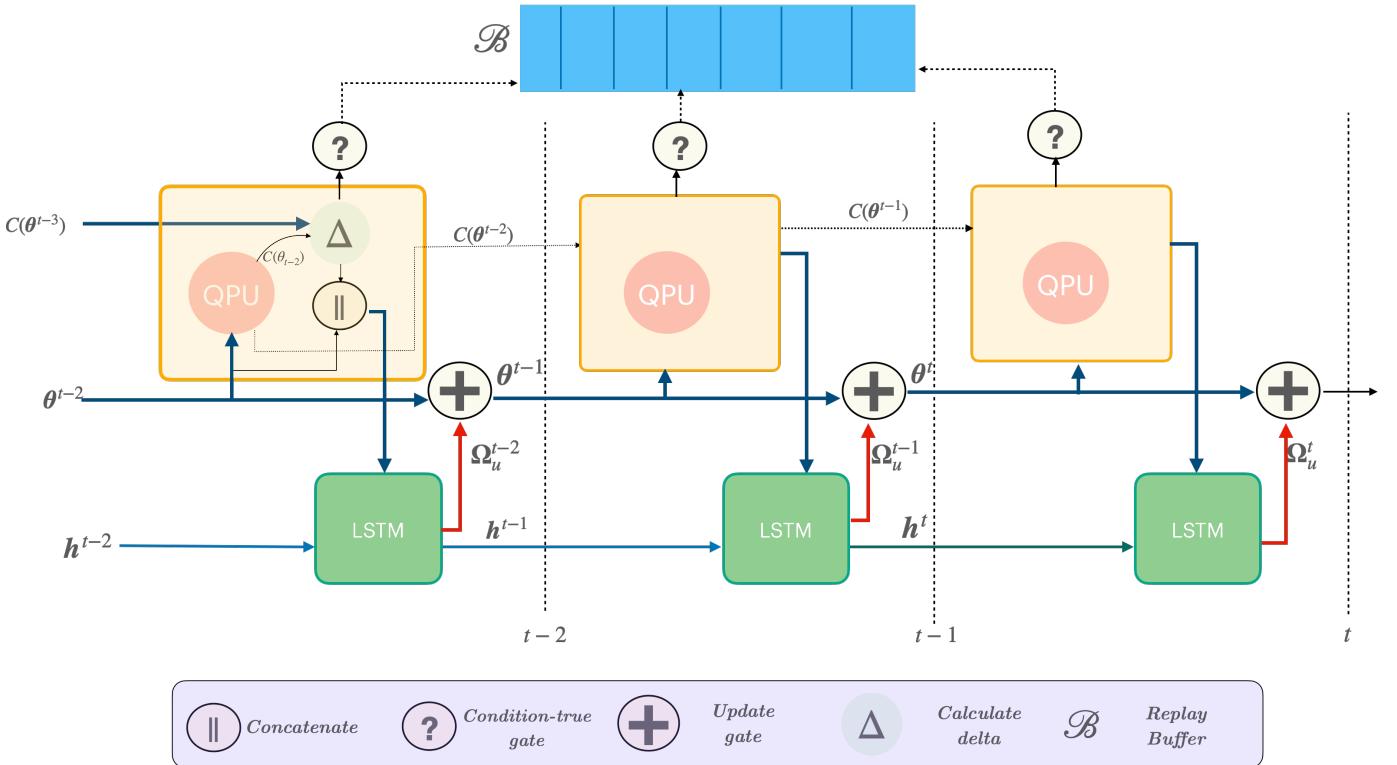


Fig. 1: An overview of our proposed meta-learning algorithm. The blue arrows indicate key inputs to the QNN and the LSTM and the red arrows indicate the output from LSTM. Best viewed in color. See Section IV for more details.

some input meta-features and performs an internal computation to generate new parameters for the optimizee network while adjusting its own parameters. In particular, if  $\theta^t$  are the optimizee parameters at time-step  $t$  during training, then the next parameters are discovered following [12]:

$$\begin{aligned} \theta^{t+1} &= \theta^t + g^t, \\ \begin{bmatrix} g^t \\ h^{t+1} \end{bmatrix} &= \mathcal{R}_\Phi(\mathcal{I}(\theta^t), h^t) \end{aligned} \quad . \quad (3)$$

Here we specify  $\mathcal{R}_\Phi$  to be an instance of LSTM that accepts a hidden state  $h^t$  and the meta-parameters  $\Phi$  and outputs the update  $g^t$ . The LSTM accepts  $\mathcal{I}(\theta^t)$  which we define to be a function that combines information from the training process and presents it as input features to the optimizer network. For instance, in [12] and similar works [14], [16],  $\mathcal{I}(\theta) = \nabla_\theta C(\theta)$ . The general form of meta-loss function tries to minimize the loss over a finite horizon window of size  $T$ . In this work, we follow a meta-loss function inspired by the deep learning literature:

$$\mathcal{L}(\Phi) = \sum_{i=1}^T w_i C(\theta^i), \quad (4)$$

where  $\mathbf{W}^m = [w_1, w_2, \dots, w_T]$  are the weights over the cost function evaluations at time step  $t$ . A uniform weighting strategy sets all weights to be equal. The *magnitude* of the weights is a matter of choice and the decision is made based on the dataset that is being used. The optimization of  $w$  in a

data driven manner is out of scope of this paper. In general, we suggest that a line of work based on hyper-parameter optimization [19] can be developed and used to dynamically adjust the weights for any given data.

#### IV. OUR ALGORITHM

We will now discuss the main components of our algorithm. Although, we generally follow the meta-optimization framework discussed above, the constraints induced by the need for gradient free optimization lead us to introduce the algorithmic tools that have not been proposed in literature in the meta-optimization context for quantum machine learning.

**Input Preprocessing:** In earlier works, the input function involved passing some gradient information from the optimizee network to the optimizer network. Since we are constrained to not use gradient information, we utilize the tuple  $(\theta^t, \Delta C(\theta))$  as inputs where  $\Delta C(\theta)$  is a *pseudo-gradient* that computes the difference between the current and previous cost function evaluation, i.e.,  $\Delta C(\theta) = C(\theta^{t-1}) - C(\theta^t)$ . These inputs are concatenated in a single vector and the input function  $\mathcal{I}(\theta)$  is prepared as:

$$\mathcal{I}(\theta) = \frac{\mathcal{P}([\theta^t; \Delta C(\theta^t)])}{p}. \quad (5)$$

Here we use a non-linear function  $\mathcal{P} : \mathbb{R}^d \mapsto [0, 1]$  that normalizes the input values to between 0 and 1. The value  $p$  controls the strength of normalization. In our experiments we use  $\mathcal{P}(x) = e^x$  with  $x$  being the input vector for the LSTM.

We empirically found  $p = 50$  to work best for the datasets considered in this study. In a future work, we would like to explore the possibility of learning an adaptive normalization strength that is derived from observing history of updates during training.

**Non Linear Parameter Updates:** In Equation (3), we defined a generic update rule in the meta-optimization framework. That update rule simply adds the output of LSTM to previously obtained parameters. In this work, we introduce a new update rule of the form:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t + \alpha \cdot \sigma(\boldsymbol{\Omega}_u^t), \quad (6)$$

where  $\boldsymbol{\Omega}_u^t$  are the updated parameters obtained from the LSTM. Compared to Equation (3), we have changed  $\mathbf{g}^t = \boldsymbol{\Omega}_u^t$  to  $\mathbf{g}^t = \alpha \cdot \sigma(\boldsymbol{\Omega}_u^t)$  where  $\alpha$  is a hyper parameter and  $\sigma$  is a non-linear activation function ( $\tanh$  in our implementation). This update rule applies a non-linear activation to the LSTM parameters and controls the strength of the update using  $\alpha$ . Informally,  $\alpha$  can be interpreted as a learning rate which helps the quantum learner in adjusting the new parameters. Since there is no direct gradient information, a destructive update (i.e.,  $\boldsymbol{\theta}^t = \boldsymbol{\Omega}_u^t$ ) would make the optimizee's parameters prone to oscillation due to the noisy output from LSTM. Additionally, the raw parameter updates from LSTM are unbounded in  $\mathbb{R}$ . A clipping function (e.g.,  $\tanh$ ) that clips the values between certain maximum and minimum values, leads to more stable updates which consequently yield a smoother parameter update.

**Replay Buffer:** In earlier works, the LSTM network was able to compute a good descent direction based on gradient information provided to it during training. Even in such works as [17], the LSTM network was only used to learn the initial parameters for optimizing a QNN and then the conventional gradient descent method was used. *However, we consider a harder case where no gradient information is available.* It is already known that a short horizon bias problem exists for learned optimizers for classical neural networks [20]. In the case of optimizing quantum networks without using any gradient information, this problem becomes even harder to overcome.

In meta-optimization a short horizon bias occurs when the meta-optimizer becomes biased to providing updates akin to taking short steps towards minima. These updates do not cause a significant overall decrease in optimizee's cost function. This effect is more pronounced when we operate in parameter space as opposed to gradient space since the LSTM network does not get any information about the curvature of the optimization surface. Running optimization in a finite horizon window in parameter space can then cause LSTM to suggest incorrect updates to the optimizee network. In the specific case of QNNs, the parameters correspond to a rotation of given input state about a particular axis. An incorrect update can very easily cause the quantum state to be rotated incorrectly and therefore lead to an increase in the value of the objective function we're interested in minimizing. To overcome this issue, we develop a technique inspired by work in reinforcement learning [21]. Throughout training, we keep track of past history of parameters

and their corresponding cost function values in a “replay buffer”. At the start of training we instantiate a double ended queue dubbed as a *replay-buffer*  $\mathcal{B}$  of a finite capacity  $R$ . For meta-iteration  $t = 1 \dots T$  we observe a history of parameters  $\boldsymbol{\theta}^t$  and the corresponding cost  $C(\boldsymbol{\theta}^t)$ . If  $C(\boldsymbol{\theta}^{t+1}) < C(\boldsymbol{\theta}^t)$  then we add the state  $s = [\boldsymbol{\theta}^{t+1}, C(\boldsymbol{\theta}^{t+1}), \Delta C(\boldsymbol{\theta}), \mathbf{h}^{t+1}]$  to the replay buffer. Once the meta-iteration ends and  $\mathcal{L}(\Phi)$  is computed, if the QNN cost function is diverging, we seed the parameters for the next meta-iteration by performing the following update:

$$\begin{aligned} \boldsymbol{\theta}^{T+1} &= \tau \cdot \boldsymbol{\theta}^T + (1 - \tau) \cdot \boldsymbol{\theta}_s \\ \tau &= \frac{\tau}{(1 + \zeta \cdot t)}. \end{aligned} \quad (7)$$

where  $\zeta$  is a decay factor that adjusts the blending coefficient  $\tau$  as the training progresses and  $\boldsymbol{\theta}^T$  are the optimizee parameters at the end of a previous unrolled meta-iteration loop.  $\boldsymbol{\theta}_s$  are the sampled parameters from the replay buffer that are chosen according to a fixed policy function  $\pi(\mathcal{B})$ . In our work this policy corresponds to choosing the parameters that lead to most cost decrease over the previous unroll iterations. We expect that future works in this directions will be able to learn  $\pi(\mathcal{B})$  along with parameters. We set  $\tau$  to be 0.9 to put more emphasis on the current parameters than sampled parameters. As training progresses and we observe divergent behavior, we decrease  $\tau$  according to Equation 7 with  $\zeta = 0.99$  and  $t$  indicating the overall global step of optimization iteration.

**Overall Algorithm:** The overall flow of the algorithm is shown in Figure 1. At a given timestep  $t$ , we expect the cost function value  $C(\boldsymbol{\theta}^{t-2})$  and the parameters  $\boldsymbol{\theta}^{t-1}$ . We then compute the cost  $C(\boldsymbol{\theta}^{t-1})$  using Equation (2). A cost delta is computed  $\Delta C(\boldsymbol{\theta}) = C(\boldsymbol{\theta}^{t-2}) - C(\boldsymbol{\theta}^{t-1})$ , which is then used as a decision metric and a feature in the input. In the former role, if  $\Delta C(\boldsymbol{\theta}) < 0$  then a sample (described earlier) is committed to the replay buffer  $\mathcal{B}$ . Then, depending on the availability of elements in  $\mathcal{B}$ , the parameters are either sampled or retained from previous step. An input feature is then pre-processed using Equation (5). The updated parameters  $\boldsymbol{\Omega}_u^t$  are then obtained from the LSTM and  $\boldsymbol{\theta}^t$  is computed using Equation (6). In the initial conditions, we set  $\Delta C(\boldsymbol{\theta}) = 1.0$ .

## V. THEORETICAL PERFORMANCE

We now analyze the theoretical performance gain that our algorithm can provide in the context of parameterized quantum circuits. We note that our algorithm is not predicated on an existence of an efficient data structure like QRAM [22], rather the performance gains are expected due to non-computation of gradient on quantum devices.

**Theorem V.1.** Consider a variational quantum circuit consisting of  $q$  qubits,  $L$  layers and  $k$  single qubit gates per layer. Let  $\mathcal{A}$  be an optimization algorithm running on a classical device helping the circuit minimize a cost function  $C(\boldsymbol{\theta})$ . Then:

- If  $\mathcal{A}$  is a gradient-dependent algorithm the the total time for one full pass (forward and backward) takes  $O(2(qLk)^2\delta_f)$ , where  $\delta_f$  is the time for a forward pass.

- If  $\mathcal{A}$  does not require a gradient, then the total time for one full pass takes  $O(2(qLk)\delta t_f)$ .

*Proof.* Consider an instance of gradient dependent algorithm  $\mathcal{A}^g$ . To update the parameters for the next step, it requires  $\nabla_{\theta} C(\theta)$ . For a circuit running on NISQ computer, currently the only method to compute gradients is to use the parameter shift method [8]. The expected gradient w.r.t single component  $\theta_i$  is given as:

$$\nabla_{\theta_i} C(x; \theta_i) = \langle \psi_x | \nabla_{\theta_i} \mathcal{F}_{\theta_i}(\hat{O}) | \psi_x \rangle, \quad (8)$$

where  $|\psi_x\rangle = U_0(x)|0\rangle$ . The quantity  $\nabla_{\theta_i} \mathcal{F}_{\theta_i}(\hat{O})$  is the gradient w.r.t to the  $i^{th}$  component of the parameter vector. The parameter shift rule states:

$$\begin{aligned} \nabla_{\theta_i} \mathcal{F}_{\theta_i}(\hat{O}) &= c[\mathcal{F}_{\theta_i+s}(\hat{O}) - \mathcal{F}_{\theta_i-s}(\hat{O})], \\ \mathcal{F}_{\theta_i}(\hat{O}) &= U^\dagger(\theta_i) \hat{O} U(\theta_i). \end{aligned} \quad (9)$$

where  $c$  is a scaling constant ( $c = 0.5$ ) and  $s$  is a constant shift angle about which the gradient is computed. To successfully evaluate  $\nabla_{\theta_i} C(\theta)$  one needs two evaluations of the quantum circuit with shifted parameters. Consequently, the time for one gradient computation over the entire quantum circuit is  $O(2qlK)$ . Then, for a full pass (i.e. computing the cost and gradient), the computation time for  $\mathcal{A}^g$  scales as  $O(2(qlK)^2\delta t_f)$ . In contrast, a gradient free method (like ours), does not incur the penalty of computing the gradient and thus the overall computation time scales as  $O(2(qlk)\delta t_f)$ .  $\square$

The reduction of a quadratic run-time to a linear run-time is significant since this will allow a QNN to scale to a larger number of data points. Although, our method does incur a storage overhead of  $O(R)$ , for all practical scenarios  $O(R) \ll O(qlK)$ .

## VI. NUMERICAL EXPERIMENTS

In this section we show numerical experiments that demonstrate the effectiveness of our method. The goal of our experiments is twofold. First, we wish to empirically demonstrate that using a LSTM based meta-optimizer can lead to better convergence with lower number of circuit executions than existing gradient based or gradient free methods. Second, we wish to show that our method is more useful than gradient based methods in realistic scenarios where the number of shots on a quantum device are limited. In this latter case we demonstrate that the LSTM based optimizer is able to evolve a significantly better optimization trajectory than a gradient based algorithm.

### A. Experiments on Machine Learning Datasets

We first present our experiments on a pattern classification task using an instance of a layered ansatz as shown in Figure 3. The first layer in the ansatz embeds the input data  $x$  into the corresponding angles of a RX gate, where  $RX(x_j) = e^{-ix_j\sigma_x/2}$  and  $\sigma_x$  is the Pauli-X matrix. The subsequent layers apply a parameterized RY rotation and are entangled using the CZ gate. Since the number of input qubits is equal to the dimensionality of data, we consider three datasets with increasing dimensionality - Gaussian, Spheres and Iris. The

Gaussian dataset is a synthetic dataset where two-dimensional clusters are instantiated by drawing samples from a multivariate Gaussian distribution of given parameters. Similarly, the spheres dataset is a collection of 3-d points which form concentric spheres with one sphere enveloping the other. Finally, we consider a truncated Iris dataset that consists of only two classes and four features that describe a particular species. In all these datasets, the labels are encoded into  $\{-1, +1\}$  and the optimization task is to find variational parameters that minimize Equation 2 with Pauli-Z gate being the observable.

In the experiments we do not apply any pre-processing or post processing on the input data. We benchmark our LSTM optimizer against two commonly used gradient based algorithms - ADAM [11] and Gradient Descent. For the LSTM optimizer, the ansatz is run for 50 iterations while for gradient based algorithms we run it for 25 iterations. We then profile the cost obtained by the corresponding algorithms with the number of circuit evaluations. The learning rate for gradient based algorithms is set as  $1e^{-2}$  and  $\alpha = 0.1$  for the LSTM optimizer.

Figure 2 shows the results of our experiments on the three datasets. The top (bottom) rows show the performance without(with) replay buffer sampling. In all cases, the LSTM optimizer is able to find parameters that minimize the cost function in far fewer circuit evaluations than gradient based optimizers. This result is an empirical validation of Theorem V.1 and intuitively makes sense since parameter shift rules evaluate the same circuit twice per parameter component to estimate the gradient at a given timestep.

Figure 2a shows a phenomenon unique to LSTM based optimizers. After finding a good descent direction, the LSTM based optimizers tend to over optimize and thus oscillate about some fixed point. This “sinusoidal” oscillation is hard to control in optimization problems since it’s not easy to guess the stationary points in multivariate objective functions. Our replay buffer sampling strategy is aimed to control this phenomenon in a more principled manner and it’s effects are visible in Figure 2d where the parameter mixing and decay lead to a more stabler convergence. The effects are also visible in Figures 2e and Figures 2f where the cost function does not diverge after reaching a minimum point.

### B. Experiments with Limited Shots

In the experiments on the machine learning datasets we simulated the performance of the optimization algorithms assuming that number of available shots on the device were infinite. This assumption is however not realistic since most NISQ devices have limited number of shots for measurement. Hence, we measure the performance of the LSTM optimizer in the setting where the number of shots is highly limited.

Figure 4 shows the results of our experiments on a simulated quantum device with only 100 shots for measurement. We *a-priori* expect the stochasticity in the sampling to manifest in the optimization process and affect the overall quality of minima for both gradient free and gradient based optimizers. In the former case, the noisy measurement of cost functions with

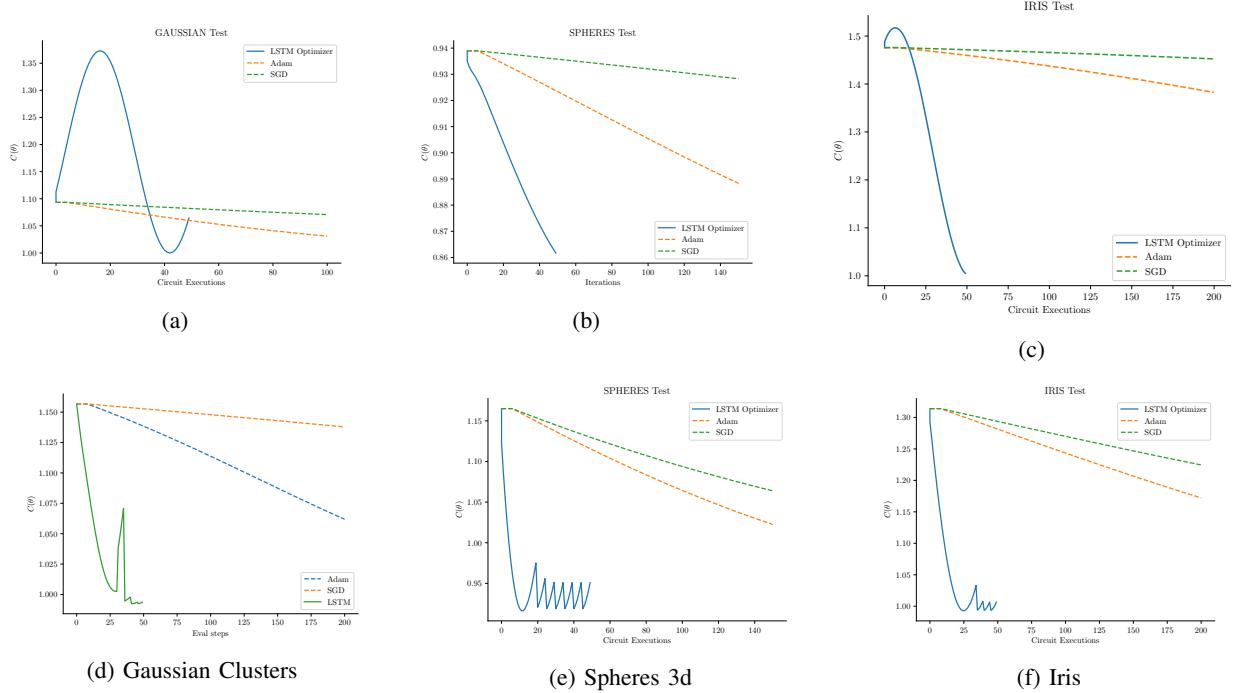


Fig. 2: Performance of LSTM-based meta-optimizer on the binary classification task for three datasets. All parameters are initialized using the normal distribution. The top row shows the performance of the algorithms without any replay buffer sampling and the bottom row shows the performance with replay buffer sampling. In both cases, the LSTM based optimizer is able to achieve a lower cost in significantly fewer circuit evaluations.

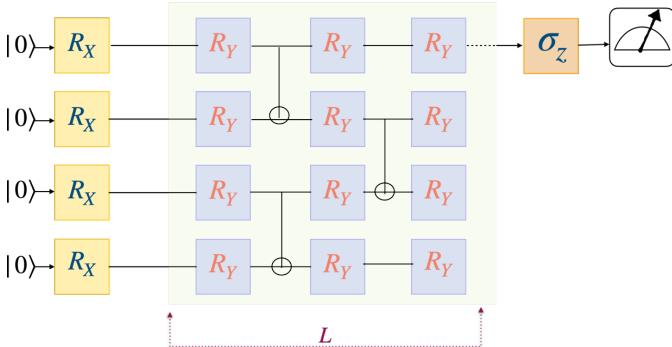


Fig. 3: The quantum circuit used in our study. The  $R_Y$  gates are parameterized by variational parameters. The  $R_X$  gates applied to each qubit constitute  $U_0$  in Equation 1

given parameter values make it hard for the LSTM to suggest “good” updates and in the latter case the noisy measurements of cost functions are coupled with noisy estimation of gradient. The noisy estimation of gradient in this case is not equivalent to stochastic averaging over mini-batches in classical deep learning algorithms and hence does not guarantee a better convergence rate. In the figure, we see that the LSTM based optimizer is able to obtain a lower minima in atleast two datasets and is able to find a minima faster than gradient based algorithms in all datasets. This demonstrates that stochastic

measurement of gradient can lead to slower and sub-optimal coverage. We further demonstrate the performance of the QNN on these datasets with 1000 shots in the appendix. To conclude, in a limited shot setting a gradient free algorithm like ours can be useful as a drop in replacement for a gradient based optimization algorithm.

### C. Comparison Against Other Gradient Free Approaches

We now benchmark our algorithm against another well known gradient-free algorithm - Simultaneous Perturbation Stochastic Approximation (SPSA) algorithm [23] which approximates gradients by taking finite differences between perturbed parameter vectors. The perturbations are generated stochastically.

We perform experiments with a four qubit, five layer variational circuit whose minimum parameters correspond to the minima of the following cost function:

$$C(\theta) = \bigotimes_{i=1}^n \langle \psi(\theta) | \sigma_z(i) | \psi(\theta) \rangle \quad (10)$$

Where  $|\psi(\theta)\rangle$  is a variational circuit consisting of strongly entangling layers i.e. rotation gates with all-to-all entanglement and  $\sigma_z(i)$  is the Pauli-Z matrix acting on the  $i^{th}$  qubit as the observable. The results of our experiments are shown in Figure 5. We can see that the LSTM Optimizer is able to find a better quality minima than SPSA as well. Moreover, SPSA makes an extra call to the circuit per optimization step making

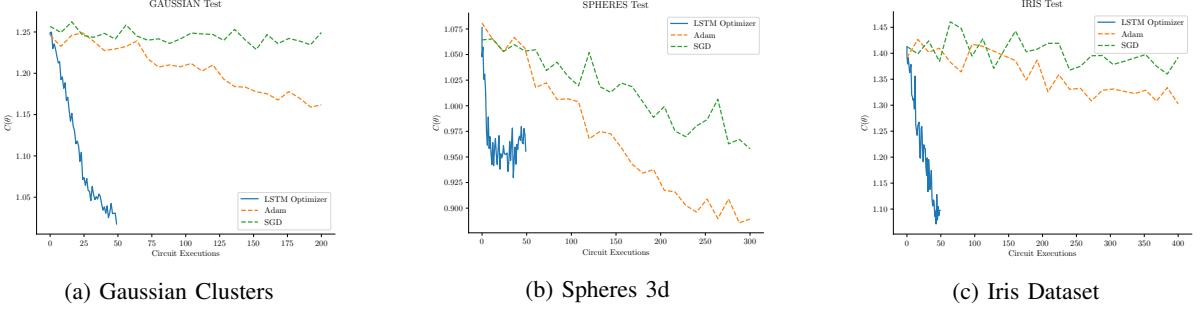


Fig. 4: Results on experiments with limited number of shots. In the limited shot setting, the LSTM based optimizer is able to minimize the cost function more effectively than gradient based approaches.

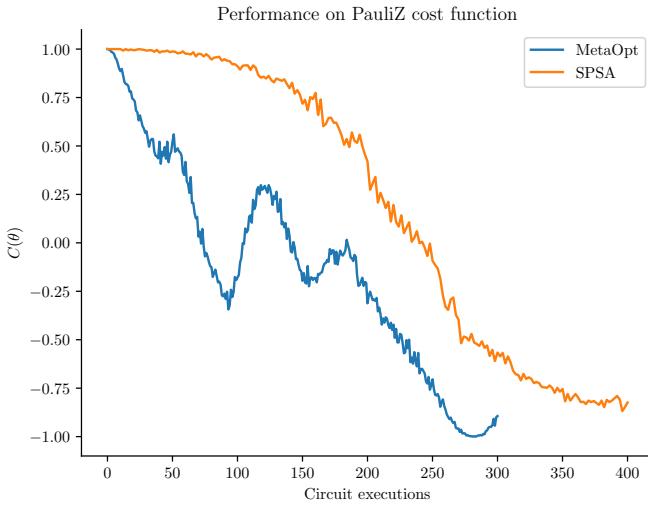


Fig. 5: Performance of LSTM based optimizer benchmarked against the SPSA algorithm. For the same cost function, SPSA makes an extra call to the quantum circuit which results in more number of circuit evaluations. LSTM based optimizer outperforms SPSA in terms of cost function minimum.

it slightly more expensive than LSTM optimizer. We conclude that the our gradient free algorithm can help VQCs scale to larger and more complex problem instances in the future.

#### D. Time Profiling Results

In our earlier experiments we noticed that LSTM optimizer converges to a minima in fewer circuit evaluations than other methods. This implied that our method could be used to scale to data with larger number of points. To test this hypothesis, we perform experiments for a pattern classification task by generating  $N$  samples from a Gaussian distribution of finite mean and covariance. We benchmarked the time per epoch (i.e. time it takes to go through an entire dataset) for our method against Adam, Gradient Descent Optimizer and RMSProp [10].

The results of our experiment are shown in Figure 6. We vary the sample size  $N = \{100, 200, 400, 600, 800\}$  and measure the scaling of the time per epoch it takes for the optimizers. It is clear from the figure that the LSTM optimizer is able to scale

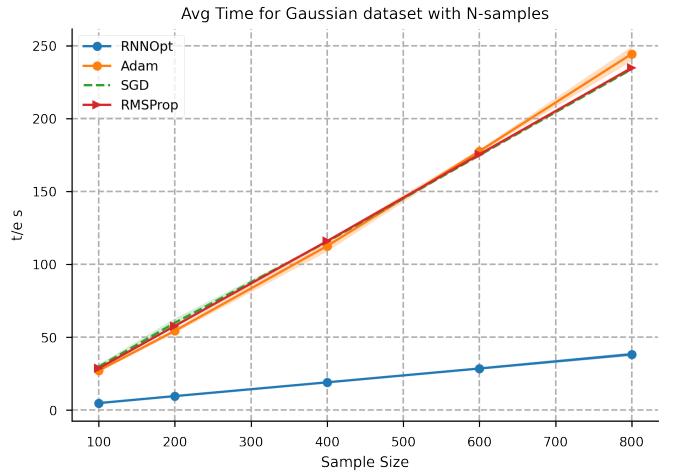


Fig. 6: Time profiling results for different sizes of Gaussian datasets.

to larger data instances without a significant increase in the time per epoch. In fact, our method is nearly *three* times faster than any gradient based method on the same dataset. This result is a positive indication that development of novel gradient free methods is a novel research direction and improvements may lead to VQCs being able to scale to handle data at the scale which is currently being handled by classical algorithms.

## VII. RELATED WORK

Meta optimization is an actively researched area in the field of deep learning. Many different meta-optimization algorithms have been proposed. Notably, [12] propose an LSTM based meta-optimizer that accepts the cost function gradient and the previous time parameters as inputs to a LSTM network and recover new parameters in the form of LSTM output while Ravi *et al.* [24] embed parameters of neural network into the cell state of the LSTM and the gradient as the state of the candidate cell state. The output is then a non-linear combination of these two embeddings in the LSTMcell. A reinforcement learning perspective is considered by [13], [25] where a guided policy search algorithm is used to find new parameters given information from previous time steps (over a finite horizon).

Another direction in meta-optimization is considered by [26] in the form of neural architecture search over the input space formed by the symbols in various update rules. They evaluate the viability of different optimization algorithms resulting from different combinations of the symbols and analyze the efficacy of the best performing combination. In the quantum case, there has been sparse but consistent work towards meta-optimization as well. The motivation for these works is towards exploring solutions to problems in quantum chemistry (e.g. finding the lowest energy Hamiltonian for a given system) and graph optimization (e.g. finding the parameters corresponding to the max-cut problem in QAOA). In [17], the authors propose a strategy of learning the best initial parameters using the LSTM setup of [12]. These initial parameters are then used in a quantum circuit to avoid barren plateaus and are optimized using conventional gradient based algorithms. Wilson *et al.* [16] propose a similar setup to [12] and utilize similar inputs to the LSTM based meta-optimizer. Their assumption is based on the cheapness of gradient evaluation on the quantum circuit, which unfortunately for the NISQ devices is not the case. Another work by [27] proposes a reinforcement learning based approach in a Quantum Approximate Optimization Algorithm (QAOA) setting by trying to learn a policy that suggests optimal parameters for a max-cut problem with two clusters and SVM/SVR for predicting the circuit depth [28]. Their proposed method leverages PPO [29] algorithm for estimating the optimal policy. Our work can also be interpreted as a variant of reinforcement learning method where we learn the policy directly instead of just suggesting the mean of a Gaussian policy. Second, our method makes no presumption on the size of the action space unlike this work where the action space is limited to just two parameters. To the best of our knowledge, our work is the first to consider a gradient free meta-optimization algorithm with significantly different inputs and update strategies. Table I summarizes the existing meta-optimization methods in quantum computing and highlights the differences from our method.

Method	Quantum Computation	Gradient	Replay Buffer	Adaptive Updates	Parameter
Meta-optimization w/ Gradients [16]	✓		✗	✗	
Learning initial parameters w/ LSTMs [17]	✓		✗	✗	
Ours	✗		✓	✓	

TABLE I: Algorithmic differences between quantum meta-optimization algorithms proposed in literature.

### VIII. CONCLUSION

We have proposed a gradient free meta optimization algorithm for quantum neural networks that can potentially scale up to larger dataset sizes in a reasonable amount of time. Experiments on different datasets show that our algorithm has a comparable quality to classical optimization algorithms while significantly outperforming them in terms of computation time. We believe that our method can be useful in exploring the answer to several open questions in the theory

of variational quantum algorithms. For instance, the barren plateau problem [30] is a notorious problem that occurs in randomly parametrized circuits even when their depth is shallow. It has been shown that initializing distributions of parameters play a key role in preventing barren plateaus [31]. It would be interesting to explore if meta-optimizers can suggest parameters that prevent occurrence of barren plateaus. In a future work, we would also like to study the generalization performance of QNNs when trained by meta-optimizers.

### IX. DATA AVAILABILITY

Data and code are available upon reasonable request from the authors.

### REFERENCES

- [1] D. Herman, C. Googin, X. Liu, A. Galda, I. Safro, Y. Sun, M. Pistoia, and Y. Alexeev, “A survey of quantum computing for finance,” *arXiv preprint arXiv:2201.02773*, 2022.
- [2] A. Ajagekar and F. You, “Quantum computing for energy systems optimization: Challenges and opportunities,” *Energy*, vol. 179, pp. 76–89, 2019.
- [3] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio *et al.*, “Variational quantum algorithms,” *Nature Reviews Physics*, vol. 3, no. 9, pp. 625–644, 2021.
- [4] E. Farhi and H. Neven, “Classification with quantum neural networks on near term processors,” *arXiv preprint arXiv:1802.06002*, 2018. [Online]. Available: <https://arxiv.org/abs/1802.06002>
- [5] M. Henderson, S. Shakya, S. Pradhan, and T. Cook, “Quanvolutional neural networks: powering image recognition with quantum circuits,” *Quantum Machine Intelligence*, vol. 2, no. 1, pp. 1–9, 2020.
- [6] J. Romero, J. P. Olson, and A. Aspuru-Guzik, “Quantum autoencoders for efficient compression of quantum data,” *Quantum Science and Technology*, vol. 2, no. 4, p. 045001, 2017.
- [7] I. Cong, S. Choi, and M. D. Lukin, “Quantum convolutional neural networks,” *Nature Physics*, vol. 15, no. 12, pp. 1273–1278, 2019. [Online]. Available: <https://www.nature.com/articles/s41567-019-0648-8>
- [8] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, “Evaluating analytic gradients on quantum hardware,” *Phys. Rev. A*, vol. 99, p. 032331, Mar 2019. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevA.99.032331>
- [9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [10] T. Tieleman, G. Hinton *et al.*, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,” *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.
- [11] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [12] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, “Learning to learn by gradient descent by gradient descent,” *Advances in neural information processing systems*, vol. 29, 2016.
- [13] K. Li and J. Malik, “Learning to optimize,” *arXiv preprint arXiv:1606.01885*, 2016.
- [14] ———, “Learning to optimize neural nets,” *arXiv preprint arXiv:1703.00441*, 2017.
- [15] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein, “Understanding and correcting pathologies in the training of learned optimizers,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 4556–4565.
- [16] M. Wilson, R. Stromswold, F. Wudarski, S. Hadfield, N. M. Tubman, and E. G. Rieffel, “Optimizing quantum heuristics with meta-learning,” *Quantum Machine Intelligence*, vol. 3, no. 1, pp. 1–14, 2021.
- [17] G. Verdon, M. Broughton, J. R. McClean, K. J. Sung, R. Babbush, Z. Jiang, H. Neven, and M. Mohseni, “Learning to learn with quantum neural networks via classical neural networks,” *arXiv preprint arXiv:1907.05415*, 2019. [Online]. Available: <https://arxiv.org/abs/1907.05415>

- [18] M. Cerezo, A. Poremba, L. Cincio, and P. J. Coles, “Variational quantum fidelity estimation,” *Quantum*, vol. 4, p. 248, 2020. [Online]. Available: <https://quantum-journal.org/papers/q-2020-03-26-248/>
- [19] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, “Algorithms for hyperparameter optimization,” *Advances in neural information processing systems*, vol. 24, 2011.
- [20] Y. Wu, M. Ren, R. Liao, and R. Grosse, “Understanding short-horizon bias in stochastic meta-optimization,” *arXiv preprint arXiv:1803.02021*, 2018.
- [21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.
- [22] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum algorithm for linear systems of equations,” *Phys. Rev. Lett.*, vol. 103, p. 150502, 2009.
- [23] H. Robbins and S. Monro, “A stochastic approximation method,” *The annals of mathematical statistics*, pp. 400–407, 1951.
- [24] S. Ravi and H. Larochelle, “Optimization as a model for few-shot learning,” in *International conference on learning representations*, 2017.
- [25] Y. Li and S. C. Benjamin, “Efficient variational quantum simulator incorporating active error minimization,” *Physical Review X*, vol. 7, no. 2, p. 021050, 2017. [Online]. Available: <https://journals.aps.org/prx/abstract/10.1103/PhysRevX.7.021050>
- [26] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, “Neural optimizer search with reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 459–468.
- [27] S. Khairy, R. Shaydulin, L. Cincio, Y. Alexeev, and P. Balaprakash, “Learning to optimize variational quantum circuits to solve combinatorial problems,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 03, pp. 2367–2375, Apr. 2020. [Online]. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/5616>
- [28] R. Shaydulin, S. Hadfield, T. Hogg, and I. Safro, “Classical symmetries and the quantum approximate optimization algorithm,” *Quantum Information Processing*, vol. 20, no. 11, pp. 1–28, 2021.
- [29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [30] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, “Barren plateaus in quantum neural network training landscapes,” *Nature communications*, vol. 9, no. 1, p. 4812, 2018. [Online]. Available: <https://www.nature.com/articles/s41467-018-07090-4>
- [31] A. Kulshrestha and I. Safro, “Beinit: Avoiding barren plateaus in variational quantum algorithms,” *arXiv preprint arXiv:2204.13751*, 2022.