

Learning to learn with quantum neural networks via classical neural networks

Guillaume Verdon,^{1,2,4,*} Michael Broughton,^{1,3,*} Jarrod R. McClean,¹ Kevin J. Sung,^{1,5} Ryan Babbush,¹ Zhang Jiang,¹ Hartmut Neven,¹ and Masoud Mohseni¹

¹Google LLC, Venice, CA 90291

²Department of Applied Mathematics, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada

³Department of Computer Science, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada

⁴Institute for Quantum Computing, University of Waterloo, Waterloo, Ontario, N2L 3G1, Canada

⁵Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109

(Dated: July 12, 2019)

Quantum Neural Networks (QNNs) are a promising variational learning paradigm with applications to near-term quantum processors, however they still face some significant challenges. One such challenge is finding good parameter initialization heuristics that ensure rapid and consistent convergence to local minima of the parameterized quantum circuit landscape. In this work, we train classical neural networks to assist in the quantum learning process, also known as meta-learning, to rapidly find approximate optima in the parameter landscape for several classes of quantum variational algorithms. Specifically, we train classical recurrent neural networks to find approximately optimal parameters within a small number of queries of the cost function for the Quantum Approximate Optimization Algorithm (QAOA) for MaxCut, QAOA for Sherrington-Kirkpatrick Ising model, and for a Variational Quantum Eigensolver for the Hubbard model. By initializing other optimizers at parameter values suggested by the classical neural network, we demonstrate a significant improvement in the total number of optimization iterations required to reach a given accuracy. We further demonstrate that the optimization strategies learned by the neural network generalize well across a range of problem instance sizes. This opens up the possibility of training on small, classically simulatable problem instances, in order to initialize larger, classically intractably simulatable problem instances on quantum devices, thereby significantly reducing the number of required quantum-classical optimization iterations.

I. INTRODUCTION

With the advent of noisy intermediate-scale quantum (NISQ) devices [1], there has been a growing body of work [1–24] aiming to develop algorithms which are suitable to be run in this near-term era of quantum computing. A particularly promising category of such algorithms are the so-called quantum-classical variational algorithms [4, 9], which involve the optimization over a family of parameterized quantum circuits using classical optimization techniques (see Fig. 1). These variational algorithms are promising because they have flexible architectures, are adaptive in nature, can be tailored to fit the gate allowances of near-term quantum devices, and are partially robust to systematic noise.

Many quantum-classical variational algorithms consist of optimizing the parameters of a parameterized quantum circuit to extremize a cost function (often consisting of the expectation value of a certain observable at the output of the circuit). This optimization of parameterized functions is similar to the methods of classical deep learning with neural networks [25–27]. Furthermore, the training and inference processes for classical deep neural networks have been shown to be embeddable into this quantum-classical PQC optimization framework [5, 28]. Given these connections, it has become common to sometimes refer to certain PQC ansatzes as *Quantum Neural Networks* [2, 7, 23, 29] (QNN's).

Optimization of QNN's in the NISQ era is currently faced with two main challenges. The first is local opti-

mization; the stochastic nature of the objective function in combination with readout complexity considerations has made direct translation of classical local optimization algorithms challenging. Proposed gradient-based optimizers either rely on a quantum form of backpropagation of errors [28] that requires additional gate depth and quantum memory, or use finite-difference gradients [2, 23] which typically require numerous quantum circuit evaluations for each gradient descent iteration. Recent works have proposed sampling analytic gradients [30, 31] to reduce this cost. However, these approaches also require many measurement runs and consequently remain expensive, and further advances are needed in this area.

The second major challenge for QNN optimization is parameter initialization. Although there have been some proposals for QNN parameter initialization heuristics [8, 32, 33], we believe there is a need for more efficient and more flexible variants of such heuristics. By initializing parameters in the neighborhood of a local minimum of the cost landscape, one ensures more consistent local optimization convergence in a fewer number of iterations and a better overall answer with respect to the global landscape. Good initialization is thus crucial to promote the convergence of local optimizers to local extrema and to select reasonably good local minima.

In this paper, we tackle the second problem of parameter initialization by exploring methods that leverage classical neural networks trained to optimize control parameters of parametrized quantum circuits. Taking inspiration from the growing body of work on meta-

learning, also known as *learning to learn* [34–38], we use a classical recurrent neural network (RNN) as a black-box controller to optimize the PQC parameters directly, as shown in Figure 2. We train this RNN using random problem instances from specific classes of problems. We explore the performance of this approach for the following problem classes: quantum approximate optimization algorithm (QAOA) for MaxCut [3], QAOA for Sherrington-Kirkpatrick Ising models [32], and a Variational Quantum Eigensolver (VQE) ansatz for the Hubbard model [6, 39, 40].

Through numerical simulations, we show that a recurrent neural network trained to optimize small quantum neural networks can learn parameter update heuristics that generalize to larger system sizes and problem instances, while still outperforming other initialization strategies at this scale. This opens up the possibility of classically training RNN optimizers for specific problem classes using instances of classically simulatable QNN’s with reasonable system sizes as training data. After this training is done, these RNN optimizers could then be used on problem instances with QNN’s whose system sizes are beyond the classically simulatable regime.

For reasons explained further in sections III C we use the RNN as a few-shot global approximate optimizer, which is used to initialize a local optimizer, such as Nelder-Mead [41–43]. In principle, the neural network could initialize any other local optimizer (such as SPSA [44], BOBYQA [45], and many others [46]), however, the focus of this paper is not to benchmark and compare various options for local optimizers. We have found that our approach compares favorably over other standard parameter initialization methods for all local optimizers studied. To the authors’ knowledge, this work is a first instance where meta-learning techniques have been successfully applied to enhance quantum machine learning algorithms.

II. QUANTUM-CLASSICAL META-LEARNING

A. Variational Quantum Algorithms

Let us first briefly review the theory of variational quantum algorithms, and how one can view the hybrid quantum-classical optimization process as a hybrid computational graph. Variational quantum algorithms are comprised of an iterative quantum-classical optimization loop between a classical processing unit (CPU) and a quantum processing unit (QPU), pictured in Figure 1.

An iteration begins with the CPU sending the set of candidate parameters θ to the QPU. The QPU then executes a parameterized circuit $\hat{U}(\theta)$, which outputs a state $|\psi_\theta\rangle$. For the types of QNN of interest in this work, namely QAOA [3] and VQE [4], the function to be optimized is the expectation value of a certain Hamiltonian operator $f(\theta) \equiv \langle \psi_\theta | \hat{H} | \psi_\theta \rangle$. We will refer to this function to as the *cost function* of the variational quantum

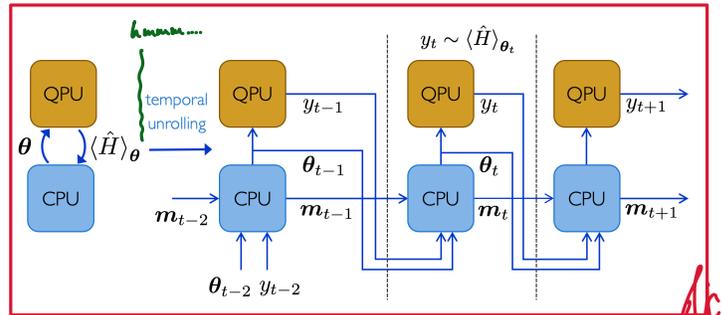


Figure 1. Unrolling the temporal quantum-classical hybrid computational graph of a general hybrid variational quantum algorithm. At the t^{th} optimization iteration, the CPU is fed the previous iterations’ parameters θ_t , and the expectation value of the Hamiltonian at the previous step $y_t = \langle \hat{H} \rangle_{\theta_t}$, it also has access to its own internal memory m_t . A classical optimization algorithm then suggest a new set of parameters θ_t , which is fed to the QPU. The QPU then executes multiple runs to obtain y_{t+1} , the expectation value of the cost Hamiltonian at the output of the parameterized quantum circuit evaluated at these given parameters.

algorithm, defined by the *cost Hamiltonian*. The expectation value of the cost Hamiltonian $\langle \hat{H} \rangle_{\theta} \equiv \langle \psi_{\theta} | \hat{H} | \psi_{\theta} \rangle$ is estimated using the quantum expectation estimation procedure [47] via many repeated runs of the QPU. Following this, the estimated expectation value is relayed back to the CPU, where the classical optimizer running on the CPU is then tasked with suggesting a new set of parameters for the subsequent iteration [48].

From an optimization perspective, the CPU is given a parametrized black-box function $f : \mathbb{R}^m \rightarrow \mathbb{R}$ for which it is tasked to find a set of parameters minimizing this cost function $\theta^* = \text{argmin}_{\theta \in \mathbb{R}^m} f(\theta)$. In many cases, finding an approximate minimum is sufficient. Typically, one must consider this function to have a stochastic output which serves as a noisy unbiased estimate (under some assumptions) of the true output value of the function we are ultimately trying to optimize. Optimizing this output rapidly and accurately, despite only having access to noisy estimates poses a significant challenge for variational quantum algorithms.

Even for perfect quantum gates and operations, for a finite number of measurement runs, there is inherent noise in the quantum expectation estimate [47]. Usually when performing quantum expectation estimation, the cost Hamiltonian can be expressed as a linear combination of k -local Pauli’s, $\hat{H} = \sum_{j=1}^N \alpha_j \hat{P}_j$, where the α_j ’s are real-valued coefficients and the \hat{P}_j ’s are Pauli’s that are at most k -local [49]. The measurement of expectations of k -local Pauli observables is fairly straightforward [9], while the linear combination of expectation values is done on the classical device. For a Hamiltonian \hat{H} with such a decomposition, we define its *Pauli coefficient norm*, denoted $\|\hat{H}\|_*$, as the one-norm of the vector of coefficients in its Pauli decomposition, namely $\|\hat{H}\|_* \equiv \|\alpha\|_1 = \sum_{j=1}^N |\alpha_j|$. For such Hamiltonians, the expected

number of repetitions is bounded by $\sim \mathcal{O}(\|\hat{H}\|_*^2/\epsilon^2)$ to get an estimate that is accurate within ϵ from the unbiased value with a desired probability [6, 50].

Finally, note that the quantum-classical optimization loop can be unrolled over time into a single temporal quantum-classical computational graph, as depicted in Figure 1. This hybrid computational graph can be considered as hybrid quantum-classical neural networks. We developed methods to propagate gradients through such hybrid computational graph using reverse-mode auto-differentiation, also known as *backpropagation* [51, 52]. We achieve this by converting hybrid quantum-classical backpropagation methods from previous work [28], originally formulated for quantum optimizers, to a form suitable for classical optimizers, which are most relevant for the NISQ era and are our focus in this paper. During the writing of this paper, other works have also employed hybrid quantum-classical backpropagation for various quantum machine learning tasks [30, 53].

B. Meta-learning with Neural Optimizers

Meta-learning, also called *learning to learn* [34, 35], consists of a set of meta-optimization techniques which aim to learn how to modify the parameters (or hyperparameters) of learning algorithms to further tailor them for a specific purpose. This could be to ensure that the learning generalizes well (minimizes test set error), to better fit the given data in less iterations (minimize training set error) [35], or to perform transfer learning (adapt a pre-trained neural network to a new task) [37, 54]. In recent years, there have been many new works in the meta-learning literature [34, 35, 37, 54, 55], and we aim to transfer some of the tools developed in the context of classical deep learning to quantum variational algorithms.

Our aim will be to train a classical *optimizer* neural network to learn parameter update heuristics for *optimizee* quantum neural networks. As mentioned previously, for our QNN's of interest, the cost function to be optimized is the expectation value of a certain Hamiltonian operator \hat{H} , with respect to a parameterized class of states $|\psi_\theta\rangle$ output by a family of parametrized quantum circuits; $f(\theta) = \langle \hat{H} \rangle_\theta$. To emulate the statistical noise of quantum expectation estimation, we will assume that the optimizer is fed noisy unbiased estimates of the QNN cost function at test time.

In many studies of meta-learning [35], it is assumed that this black box (in our case a QNN) is differentiable and that the learner has oracular access to the gradients of the function with respect to its parameters, $\nabla f(\theta)$. Since precise gradient estimations on NISQ devices are hampered by the large number of runs required and by the noise of the device, we focus on the case where the learner will only have access to black box function queries at test time. We will, however, use gradients for neural optimizer network training. This access to gra-

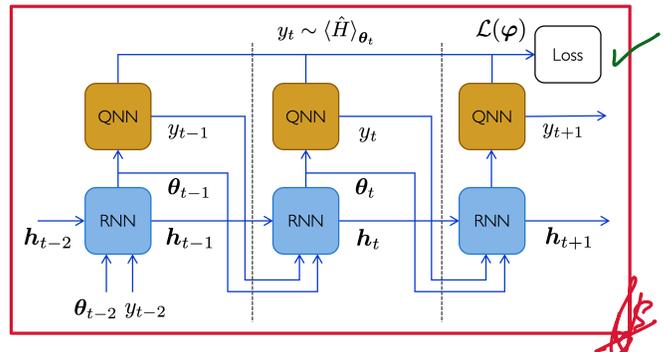


Figure 2. Unrolled temporal quantum-classical computational graph for the meta-learning optimization of the recurrent neural network (RNN) optimizer and a quantum neural network (QNN). This graph is similar to the general VQA graph in Figure 1 except that the memory of the optimizer is encoded in the hidden state of the RNN \mathbf{h} , and we represent the flow of data used to evaluate the meta-learning loss function. This meta loss function \mathcal{L} is a functional of the history of expectation value estimate samples $\mathbf{y} = \{y_t\}_{t=1}^T$, and is thus indirectly dependent on the RNN parameters φ . We see that backpropagating from the meta-loss node to the RNN's necessitates gradients to pass through the QNN.

dients during training is not strictly necessary, but can speed up training in some cases. Note that gradients of hybrid quantum-classical computational graphs can be obtained by backpropagation (automatic differentiation) when simulating quantum circuits, or by using techniques for backpropagation through black boxes [56], or hybrid quantum-classical backpropagation [28, 30].

To choose an architecture for the optimizer network, we interpret the QNN parameters and cost function evaluations over multiple quantum-classical iterations as a sequence-to-sequence learning problem. A canonical choice of neural network architecture for processing such sequential data is a recurrent neural network (RNN) [57, 58]. Generally speaking, a recurrent neural network is a network where, for each item in a sequence, the network accepts an input vector, produces an output vector, and potentially keeps some data in memory for use with subsequent items. The computational graph of a RNN usually consists of many copies of the network, each sharing the same set of parameters, and each representing a time step. The recurrent connections, which can be interpreted as self-connections representing the data flow over time, can be represented as connections between copies of the network representing subsequent time steps. In this way, the computational graph can then be pictured in an *unrolled* form, as depicted in Figure 2. A particular type of RNN architecture which has had demonstrable successes over other RNN architectures is the Long Short Term Memory Network (LSTM) [59]. The LSTM owes its successes to its internal tunable mechanisms which, as its name implies, allow it to identify both long-term and short-term dependencies in the data.

The meta-learning neural network architecture used in this paper is depicted in Figure 2, there, an LSTM

recurrent neural network is used to recursively propose updates to the QNN parameters, thereby acting as the classical black-box optimizer for quantum-classical optimization. At a given iteration, the RNN receives as input the previous QNN query’s estimated cost function expectation $y_t \sim p(y|\theta_t)$, where y_t is the estimate of $\langle \hat{H} \rangle_t$, as well as the parameters for which the QNN was evaluated θ_t . The RNN at this time step also receives information stored in its internal hidden state from the previous time step h_t . The RNN itself has trainable parameters φ , and hence it applies the parameterized mapping

Code Panel - 7

$$h_{t+1}, \theta_{t+1} = \text{RNN}_{\varphi}(h_t, \theta_t, y_t) \quad (1)$$

which generates a new suggestion for the QNN parameters as well as a new hidden state. Once this new set of QNN parameters is suggested, the RNN sends it to the QPU for evaluation and the loop continues. Note that this specific meta-learning architecture was adapted from previous work [34], where the task of ‘learning to learn without gradient descent by gradient descent’ was considered.

The QNN architectures we chose to focus on were part of a class of ansätze known as Quantum Alternating Operator Ansatz [10], which are generalizations of the Quantum Approximate Optimization Algorithm [3]. These ansätze can be interpreted as a method for variationally-optimized bang-bang-controlled quantum-dynamical evolution in an energy landscape [3, 28, 60, 61]. In this case, the QNN’s variational parameters are the control parameters of the dynamics, and by appropriately tuning these parameters via quantum-classical optimization, one can cause the wavefunction to effectively descend the energetic landscape towards lower-energy regions. In recent works [28, 61], explicit connections between these quantum-dynamical control parameters and the hyperparameters of gradient descent algorithms were established for the continuum-embedded variants of QAOA. Thus, one can interpret these QAOA parameters as analogous to the hyperparameters of a quantum form of energetic landscape descent. In a recent work in classical meta-learning [35], the authors use a RNN to control the hyperparameters of a neural network gradient-based training algorithm, drastically improving the neural networks training time and quality of fit when compared to stochastic gradient descent. It is thus natural to consider using an RNN to optimize QAOA parameters, given the above intuition about their analogous relation to gradient descent hyperparameters. Another reason for choosing this particular type of QNN is that the number of variational parameters in these ansätze do not depend on the system size, only on the number of alternating operators. This means we can use this approach to train our RNN on smaller QNN instance sizes for certain problems, then test for generalization on larger QNN instance sizes.

Before we dive further into the specific quantum-classical meta-learning experiments, we will provide further details on how one would train an RNN to optimize

such QNN’s.

1. Meta-Training & Loss functions

The objective of quantum-classical meta-learning is to train our RNN to learn an efficient parameter update scheme for a family of cost functions of interest, i.e., to discover an optimizer which efficiently optimizes a certain distribution of optimizees, on average. We consider an efficient optimizer to be one which finds sufficiently optimal approximate local minima of cost functions in as few function queries as possible. What qualifies as sufficiently optimal will depend on the class of problems at hand and the domain of application of interest.

In the original work by DeepMind [34], the neural optimizer was to be used as a general optimizer; little to no assumptions were made about the optimizee (the network being optimized) apart from the dimension of the parameter space. The optimizer network was to be trained on one ‘data set’ of optimizees, yet had to be applicable to a wide array of optimizees previously unseen. To learn such a general optimization strategy, the optimizer RNN was trained on random instances of a fairly general distribution of functions, namely, functions sampled from Gaussian processes [62].

Since we are focused on QNN optimization landscapes which are known to differ from classical Gaussian process optimization landscapes [29] we instead aimed to train specialized neural optimizers that are tailored to specific classes of problems and QNN ansätze. To explore how effective this is, we trained our RNN on random QNN instances within a targeted class of problems, namely QAOA and VQE, and tested the trained network on (larger) previously unseen instances from their respective classes. In Section III we describe in greater detail the various classes of problems and corresponding ansätze which were considered for training and testing.

Given a distribution of optimizees of interest, we must choose an adequate meta-learning loss function $\mathcal{L}(\varphi)$ with respect to which we will want to optimize the RNN parameters φ . For a given QNN with cost function $f(\theta) = \langle \hat{H} \rangle_{\theta}$, we know that the RNN’s meta-learning loss function $\mathcal{L}(\varphi)$ will generally be dependent on the estimated cost function (quantum expectation estimate) history $\{\mathbb{E}_{f,y}[f(\theta_t)]\}_{t=1}^T$, but there is some flexibility in choosing exactly what this dependence is. Choosing the appropriate meta-learning loss function for the task at hand can be tricky, and depends on what is the particular application of the QNN. To be most general, we will want to pick a loss function which can learn to rapidly find optima of the parameter landscape yet is still constantly driven to find higher quality optima.

A simple choice of meta-learning loss function would be to use the expected final cost value at the end of the optimization time horizon, averaged over our samples from our distribution of functions f , i.e., $\mathcal{L}(\varphi) = \mathbb{E}_{f,y_T}[f(\theta_T)]$. In practice, this is a sparse signal, and would require

backpropagation through a large portion of the computational graph for the loss signal to reach early portions of the RNN graph. A practical option of the same vein is the *cumulative regret*, which is simply the sum of the cost function history uniformly averaged over the time horizon $\mathcal{L}(\varphi) = \sum_{t=1}^T \mathbb{E}_{f,\mathbf{y}}[f(\theta_t)]$. This is a better choice as the loss signal is far less sparse, and the cumulative regret is a proxy for the minimum value achieved over the optimization history. In practice, this loss function may not be optimal as it will prioritize rapidly finding an approximate optimum and staying there. What is needed instead is a loss function that encourages exploration of the landscape in order to find a better optimum. The loss function we chose for our experiments is the *observed improvement* at each time step, summed over the history of the optimization:

Code Block
3

$$\mathcal{L}(\varphi) = \mathbb{E}_{f,\mathbf{y}} \left[\sum_{t=1}^T \min\{f(\theta_t) - \min_{j<t}[f(\theta_j)], 0\} \right], \quad (2)$$

The observed improvement at time step t is given by the difference between the proposed value, $f(\theta_t)$, and the best value obtained over the past history of the optimization until that point, $\min_{j<t}[f(\theta_j)]$. If there is no improvement at a given time step then the contribution to the loss is nil. However, a temporary increase of the cost function followed by a significant improvement over the historical best will be rewarded rather than penalized (in contrast to the behavior of the cumulative regret loss).

In order to train the RNN, we need to differentiate the above loss function $\mathcal{L}(\varphi)$. One option to achieve this is via backpropagation of gradients through the unrolled RNN graph (depicted in Fig. 2). This approach is called *backpropagation through time*, and can be tricky to scale to arbitrarily deep networks due to vanishing/exploding gradient problems [58]. For practical purposes, a small time horizon is preferable, as it limits the complexity of the training of the RNN optimizer and avoids the pathologies of backpropagation through long time horizons. Since our loss function $\mathcal{L}(\varphi)$ is dependent on the QNN evaluated at multiple different parameter values $\{\theta_t\}_{t=1}^T$, in order to perform backpropagation through time we need to backpropagate gradients through multiple instances of the QNN.

As our approach was backpropagation-based, to avoid problems of gradient blowup and to minimize the complexity of training, we keep a small time horizon for our numerical experiments featured in Section III C. As such, our RNN optimizer is intended to only run for a fixed number of iterations, and will be used as an initializer for other optimizers that perform local search. In principle, one could let the RNN optimize over more iterations at inference time than it was originally trained for, though the performance for later iterations may suffer. In our case the output of the RNN optimizer after a fixed number of iterations is used to initialize the parameters of the QNN's near a typical optimal set of parameters. It has been observed [32], and in some cases formally proven

[63], that QAOA-like ansatz have a concentration of optimal parameters. Thus, the neural optimizer is used to learn a problem-class-specific initialization heuristic, and the fine-tuning is left for other optimizers. Since the neural optimizer would eventually learn a local heuristic for the fine-tuning, the added complexity cost of training for long time horizons if not justified by corresponding improvements in optimization efficiency, and we find that the combination of the neural optimizer as initializer and a greedy heuristic such as Nelder-Mead works quite well in practice, as shown in Figure 3. Now, let us cover which ansatz we applied our RNN to learn to optimize.

III. NUMERICAL EXPERIMENTS

In this section, we provide a brief overview of the quantum neural network ansatz and problem instances considered for the hybrid meta-learning numerical experiments (results presented in Section III C). We trained and tested different ‘specialist’ RNN optimizers for each of these three problem classes: quantum approximate optimization for MaxCut (MaxCut QAOA), quantum approximate optimization for Sherrington-Kirkpatrick Ising models [32] (Ising QAOA), and a Trotter-based variational quantum eigensolver ansatz for the Hubbard model [39] (Hubbard VQE). We provide a brief introduction to each of these three classes, as well as describe the distribution of instances from these classes from which we sampled to generate training and testing instances.

A. Quantum Approximate Optimization Algorithms

Let us first introduce a general QAOA ansatz before we specialize to applications to MaxCut problems and Ising (Sherrington-Kirkpatrick; SK) Hamiltonians. The goal of the QAOA is to prepare low-energy states of a *cost Hamiltonian* \hat{H}_C , which is usually a Hamiltonian which is diagonal in the computational basis. To achieve this, we typically begin in an eigenstate of a *mixer Hamiltonian* \hat{H}_M , which does not commute with the cost Hamiltonian; $[\hat{H}_C, \hat{H}_M] \neq 0$. Applied onto this initial state is a sequence of exponentials of the form

Hang on!

$$\hat{U}(\theta) = \prod_{j=1}^P e^{-i\theta_m^{(j)} \hat{H}_M} e^{-i\theta_c^{(j)} \hat{H}_C}, \quad (3)$$

where $\theta = \{\theta_m, \theta_c\}$ are variational parameters to be optimized. Note that in the above and throughout this paper we will use the operator product notation convention where $\prod_{j=1}^M \hat{U}_j = \hat{U}_M \dots \hat{U}_1$. The objective function for this optimization is simply the expectation of the cost Hamiltonian after applying $\hat{U}(\theta)$ to the initial state. This sequence of exponentials is the quantum alternating operator ansatz [3, 10]. This is an algorithm which is well-suited for the NISQ era as the number of

gates scales linearly with P , the exponentials of \hat{H}_M and \hat{H}_C are usually easy to compile without any need for approximation via Trotter-Suzuki decomposition [64]. The cost Hamiltonian is typically a sum of terms that are diagonal in the computational basis and often simple to compile. Furthermore, there is no need to split the quantum expectation estimation over multiple runs in order to estimate the various terms; each repetition yields an estimate of the cost function directly.

Now that we have introduced the general QAOA approach, we can explore the specialization of the QAOA to two specific domains of application; namely, MaxCut QAOA and QAOA for Sherrington-Kirkpatrick Ising models.

1. MaxCut QAOA

The problem for which the QAOA was first explored was for MaxCut [3]. Let us first provide a brief introduction to the MaxCut problem. Suppose we have a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ where \mathcal{E} are the edges and \mathcal{V} the vertices. Given a partition of these vertices into a subset \mathcal{P}_0 and its complement $\mathcal{P}_1 = \mathcal{V} \setminus \mathcal{P}_0$, the corresponding *cut set* $C \subseteq \mathcal{E}$ is the subset of edges that have one endpoint in \mathcal{P}_0 and the other endpoint in \mathcal{P}_1 . The *maximum cut* (MaxCut) for a given graph \mathcal{G} is the choice of \mathcal{P}_0 and \mathcal{P}_1 which yields the largest possible cut set. The difficulty of finding this partition is well known to be an NP-Complete problem in general.

To translate this problem to a quantum Hamiltonian, we can assign a qubit to each vertex $j \in \mathcal{V}$. The computational basis states of these qubits can then be used as binary labels to indicate which partition each qubit is in, i.e., if the qubit j is in the state $|l\rangle_j$, $l \in \{0, 1\}$, we assign it to the partition \mathcal{P}_l . We can evaluate the size of a cut by counting how many edges have endpoints in different partitions. In order to do this counting, we can compute the XOR of the bit values for the endpoints of each edge and add up these clauses. This cut cardinality can thus be encoded into the cost Hamiltonian for the QAOA as follows:

$$\hat{H}_C = \sum_{\{j,k\} \in \mathcal{E}} \frac{1}{2} (\hat{I} - \hat{Z}_j \hat{Z}_k). \quad (4)$$

Now, for our choice of mixer, the standard choice is the sum of Pauli \hat{X} on each qubit, $\hat{H}_M = \sum_{j \in \mathcal{V}} \hat{X}_j$. This is a good choice as each term is non-commuting with the cost Hamiltonian and it is easy to exponentiate with minimal gate depth. The standard choice of initial state is the uniform superposition over computational bitstrings $|+\rangle^{\otimes |\mathcal{V}|}$, which is an eigenstate of the mixer Hamiltonian. We can now construct our ansatz following [3] by choosing some value for P and substituting in our MaxCut \hat{H}_C and \hat{H}_M . By applying and variationally optimizing the QAOA, one obtains a wavefunction which, when measured in the computational basis, has a high proba-

bility of yielding a bitstring corresponding to a partition of large cut size [3].

In order to train and test the RNN optimizer on MaxCut QAOA problems, we generated random problem instances in the following fashion: we first fixed an integer n , and then randomly sampled an integer uniformly from the range $k \in [3, n-1]$. Finally, we tossed a random graph from $\mathcal{G}_{n,p}$ with $p = k/n$ and constructed the corresponding MaxCut QAOA QNN of the form of [3] for $P = 2$. Note that a random $\mathcal{G}_{n,p}$ graph is a graph on n nodes where an edge between any two nodes is added independently with probability p . To generate training data, we uniformly sampled $n \in [6, 9]$, yielding QNN system sizes of at most nine qubits. To train the RNN, 10000 sampled instances from this training set were used. To generate our testing data, we fixed $n = 12$, yielding QNN system sizes of 12 qubits, and sampled 50 instances using the procedure described above.

It has been observed that for random 3-regular graphs, at fixed parameter values of the QAOA ansatz, the expected value of the cost function $\langle \hat{H}_C \rangle_\theta$ concentrates [63]. Our results displayed in Figures 3 and III C corroborate this finding while operating on a slightly broader ensemble of random graphs. This is made clear by noting that initially the MaxCut QAOA has a much narrower 95% confidence interval across problem instances regardless of optimization algorithm when compared to Ising (SK) QAOA.

2. Ising QAOA

Another domain of application where we tested quantum-classical meta-learning was with the QAOA for finding low energy states of a type of Ising spin glass model known as the Sherrington-Kirkpatrick (SK) model. Many problems in combinatorial optimization can be mapped to these models [65] (for example, training Boltzmann machine neural networks [24, 66]). In general, finding the lowest energy state of such models is known to be NP-Hard. Using the QAOA, we aim to find low-energy states of an SK Ising Hamiltonian on the graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, which has the form

$$\hat{H}_C = \frac{1}{\sqrt{n}} \sum_{\{j,k\} \in \mathcal{E}} J_{jk} \hat{Z}_j \hat{Z}_k + \sum_{j \in \mathcal{V}} h_j \hat{Z}_j \quad (5)$$

where $n = |\mathcal{V}|$ is the number of vertices, and J_{jk} and h_j are coupling and bias coefficients. For our numerical experiments we considered only the case of the fully connected model where \mathcal{G} is the complete graph. Like the MaxCut QAOA, the choice of mixer Hamiltonian is the sum of the transverse field on all the qubits, $\hat{H}_M = \sum_j \hat{X}_j$, and the initial state is chosen as a uniform superposition over all computational basis states $|+\rangle^{\otimes n}$. The parametric ansatz is once again in the form of a regular QAOA (as in [3]), now with the SK Ising Hamiltonian [5] as the cost Hamiltonian. In similar fashion to MaxCut, when we optimize the parameters for this QAOA,

we obtain a wavefunction which, when measured in the computational basis, will yield a bit string corresponding to a spin configuration with a relatively low energy.

Let us now outline the methods used to generate the training and testing data for the RNN specializing in the optimization of Ising QAOA ansatz parameters. To generate random instances of Ising QAOA, we sampled random values of J_{jk} , h_j and n . For both the training and testing data, after drawing a value for n , the parameters J_{ij} and h_i were drawn from independent Gaussian distributions with zero mean and unit variance. Finally, we constructed the corresponding Ising QAOA QNN ansatz of the form [3] with $P = 3$ for the sampled Hamiltonian. For the training instances, we sampled the number of qubits uniformly from $n \in [6, 8]$, yielding QNN system sizes of at most 8 qubits. The size of the training set was of 10000 instances from the above described distribution. For testing, we drew 50 samples uniformly from $n \in [9, 11]$, thus testing was done with strictly larger instances than those contained in the training set.

B. Variational Quantum Eigensolvers

1. Hubbard Model VQE

Here we describe the variational quantum eigensolver (VQE) ansatz that were used to generate the results in Fig. 3. The specific class of VQE problems we chose to consider were for variational preparation of ground states of Hubbard model lattices [39]. The Hubbard model is an idealized model of fermions interacting on a lattice. The 2D Hubbard model has a Hamiltonian of the form $\hat{H} = \hat{T}_h + \hat{T}_v + \hat{V}$, where \hat{T}_h and \hat{T}_v are the horizontal and vertical hopping terms and \hat{V} a spin interaction term, more explicitly,

$$\hat{H} = -t \sum_{\langle i,j \rangle, \sigma} (\hat{a}_{i,\sigma}^\dagger \hat{a}_{j,\sigma} + \hat{a}_{j,\sigma}^\dagger \hat{a}_{i,\sigma}) + U \sum_i \hat{a}_{i,\uparrow}^\dagger \hat{a}_{i,\uparrow} \hat{a}_{i,\downarrow}^\dagger \hat{a}_{i,\downarrow} \quad (6)$$

where the $\hat{a}_{j,\sigma}$ and $\hat{a}_{j,\sigma}^\dagger$ are annihilation and creation operators on site j with spin $\sigma \in \{\uparrow, \downarrow\}$. The goal of the VQE is to variationally learn a parametrized circuit which prepares the ground state of the Hubbard Hamiltonian from [6], or at least an approximation thereof.

Our variational ansatz to prepare these approximate ground states is based on the Trotterization of the time evolution under the Hubbard model Hamiltonian, it is of the form

$$\hat{U}(\boldsymbol{\theta}) = \prod_{j=1}^P e^{-i\theta_h^{(j)} \hat{T}_h} e^{-i\theta_v^{(j)} \hat{T}_v} e^{-i\theta_U^{(j)} \hat{V}} \quad (7)$$

where $\boldsymbol{\theta} = \{\theta_h, \theta_v, \theta_U\}$ are the variational parameters for the P Trotter steps. The exponentials at each step are done using a single fermionic swap network [39]. This is similar to the ansatz used in [6] but corresponds to a different order of simulation of the terms.

Let us provide more details as to our choices of parameters used to generate the results from Figure 3. We used an ansatz consisting of $P = 5$ steps, where each step introduced 3 parameters. For our initial state, we use an eigenstate of the kinetic term with the correct particle number and the same total spin as the ground state, and we study the model at half-filling. We set $t = 1.0$ for all instances, and this defines our units of energy. Our training data consists of 10000 instances with the lattice system size chosen to be either $n = 2 \times 2$ or $n = 3 \times 2$ with equal probability and with U chosen from a uniform distribution on the domain of $[0.1, 4.0]$. After training, we tested the neural network on instances with system size $n = 4 \times 2$, again strictly larger than our training set.

C. Meta-learning Methods & Results

In this section we present the main results of our quantum-classical meta-learning experiments, displayed in Figure 3, and discuss some additional details of our methods used to produce these results. We trained and tested a set of long short-term memory (LSTM) recurrent neural networks (RNN) to learn to optimize the variety of QNN instances discussed in sections III A and III B, namely, MaxCut QAOA, Ising QAOA and Hubbard VQE. For each of the three problem classes, the RNN was trained using 10000 problem instances. This training of the RNN was executed over a maximum of 1000 epochs, each with a time horizon of 10 iterations. Hence, training required the simulation of inference for at most 1 million quantum neural networks. In most cases the meta-training was stopped well before these 1000 epochs were completed, following standard early-stopping criteria [68].

The quantum circuits used for training and testing the recurrent neural network were executed using the Cirq quantum circuit simulator [69] running on a classical computer. The VQE ansatz were built using OpenFermion-Cirq [67]. Neural network training and inference was done in TensorFlow [70], using code adapted from previous work by DeepMind [34].

For both testing and training, we squashed the read-out of the cost function by a quantity which bounds the operator norm of the Hamiltonian. This was done to ensure a normalized loss signal for our RNN across various problem instances. In classical machine learning, normalizing data variance is well-known to accelerate and ameliorate training [71]. In the same spirit, we fed the RNN a cost function squashed according to the Pauli coefficient norm, denoted $\|\cdot\|_*$. Recall that for a Hamiltonian \hat{H} with a decomposition as a linear combination of Paulis of the form $\hat{H} = \sum_j \alpha_j \hat{P}_j$, then $\|\hat{H}\|_* \equiv \|\alpha\|_1 = \sum_j |\alpha_j|$. The squashed cost function is then simply the regular expectation value of the Hamiltonian, divided by the Pauli coefficient norm, $f(\boldsymbol{\theta}) = \langle \hat{H} \rangle_{\boldsymbol{\theta}} / \|\hat{H}\|_*$. As all Paulis have a spectrum of $\{\pm 1\}$ we are guaranteed that the squashed cost function $f(\boldsymbol{\theta})$ has its range contained in $[-1, 1]$. In

Treat these substitutions separately

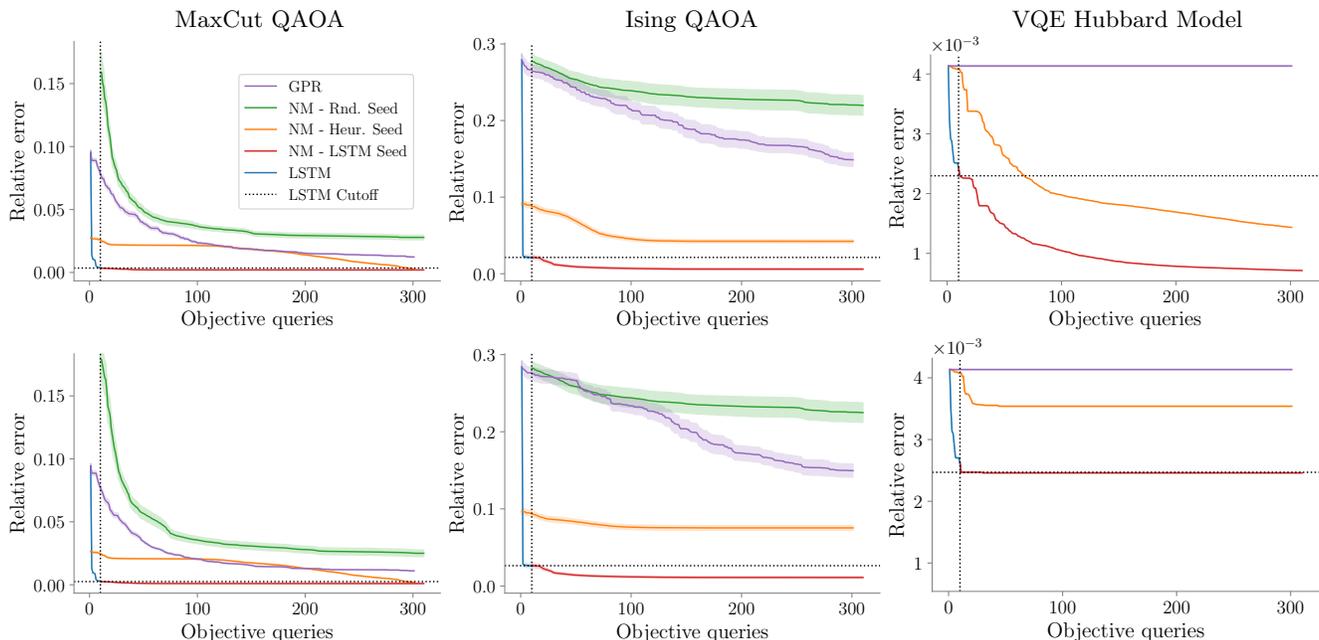


Figure 3. Displayed above are the average relative errors with respect to the number of objective function queries during the training of 50 random problem instances for the three classes of problems of interest, QAOA for MaxCut (left), QAOA for Ising models (middle), and VQE for the Hubbard model (right), for various choices of optimizers and initialization heuristics. The problem instances were sampled from the testing distribution described in sections III A and III B. These include a Gaussian Process Regression (GPR) optimizer [62], and Nelder-Mead (NM) [41] with various initialization heuristics. The first of these initialization heuristics was the best of 10 random guesses seed (Rnd. Seed). Also presented is NM initialized with application-specific heuristic seeds (Heur. Seed), which consisted of the adiabatic heuristic for VQE [67], and the mean optimal parameters of the training set for QAOA [63], and finally the seed from our meta-learned neural optimizer (LSTM). We cut off the LSTM after 10 iterations, as it is used mainly as an initializer for other optimizers. Note that we have not included overhead of the meta-training in this plot, see the main text for a breakdown of the overhead for the training of the LSTM. The top row is for noiseless readout of the expectation, while the bottom row has some Gaussian noise with variance 0.05 added to the expectation value readouts, thus emulating approximate estimates of expectation values. For reference, for the set of testing instances, given a QPU inference repetition rate of 10 kHz, the necessary wall clock time per objective query to achieve this variance [6] in the cost estimate is at most (70 ± 40) seconds for MaxCut QAOA, (2.3 ± 0.6) seconds for Ising QAOA, and (30 ± 20) seconds for Hubbard VQE. Note that the relative error is the difference in the squashed cost function relative to the squashed global optimum found through brute force methods, i.e., $f_{\text{rel}}(\theta) = (f(\theta) - \min_{\theta} f(\theta))$. Error bars represent the 95% confidence interval for the random testing instances from the distribution of problems described in sections III A and III B.

Figure 3 we plot the *relative error*, which is the difference in the squashed cost function relative to the globally optimal squashed cost function value found through brute force methods, $f_{\text{rel}}(\theta) = (f(\theta) - \min_{\theta} f(\theta))$. The brute force optimization methods were basin hopping for the QAOA instances [72], and exact diagonalization for the VQE instances.

For the testing of the trained LSTM, we used randomly sampled instances from the distributions of ansatz described in sections III A and III B. In all cases, the testing instances were for larger-size systems than those used for training, while keeping the number of variational parameters of the ansatz fixed. Note that as all the ansatz considered in this paper were QAOA-like, one can thus scale the size of the system while keeping the same number of parameters. This is an important feature of this class of ansatz as our LSTM is trained to optimize ansatz of a fixed parameter space dimension.

For all instances, the LSTM was trained on a time

horizon of $T = 10$ quantum-classical iterations, using the observed improvement (2) as the meta-learning loss function. We trained the LSTM on noiseless quantum circuit simulations in Cirq [69]. Note that training of each of the three LSTM networks already required the simulation of 1 million quantum circuit executions with the chosen time horizon of 10 iterations, and that the number of quantum circuit simulations scales linearly with the time horizon. Additionally, gradient-based training required backpropagation through time for the temporal hybrid quantum-classical computational graph, which added further linearly-scaling overhead. Thus, we chose a short time horizon to minimize the complexity of the training. For reference, 10 iterations is a significantly smaller number of quantum-classical optimization iterations than what is typically seen in previous works on QNN optimization [46]. The typical number of iterations required by other optimizers is usually on the order of hundreds to possibly thousands to reach a comparable

optimum of the parameter landscape.

Although the LSTM reaches a good approximate optimum in these 10 iterations, some applications of QNN's such as VQE require further optimization as a high-precision estimate of the cost function is desired. Thus, instead of simply using the LSTM as an optimizer for an extended time horizon, we used the LSTM as a few-iteration initializer for Nelder-Mead (NM). This was done to minimize the complexity of training the RNN and avoid the instabilities of longer training horizons where the RNN would most likely learn a local method for fine-tuning its own initial guess. A longer time horizon would thus most likely not have provided a significant gain in performance, all the while substantially increasing cost of training.

We tested the robustness of the RNN optimizer by comparing its performance to other common optimizers, both in the cases where Gaussian noise was added to the cost function evaluations, and for a noiseless readout idealized case. This additional Gaussian noise can be interpreted as a means to emulate the natural noise of quantum expectation estimation with a finite number of measurement runs [9]. Figure 3 allows for comparison of noisy and noiseless inference (QNN optimization) for the trained LSTM versus alternative optimization and initialization heuristics. For the noisy tests, the expectation samples obeyed a normal distribution of variance 0.05, thus the cost function estimates were drawn according to $y_t \sim \mathcal{N}(\langle \hat{H} \rangle_{\theta_t}, 0.05)$ for the results presented in Figure 3. For the testing instances used to generate the results presented in Figure 3, following a standard prescription [6] for the number of repetitions required to guarantee an upper bound to the variance of 0.05, the number of repetitions (QNN inference runs) should be of $(7 \pm 4) \times 10^5$ repetitions for MaxCut QAOA, $(2.3 \pm 0.6) \times 10^4$ repetitions for Ising QAOA, and $(3 \pm 2) \times 10^5$ repetitions for Hubbard VQE. In terms of wall clock time, assuming that the QPU can execute 10000 repetitions (consisting of a quantum circuit execution, multi-qubit measurement, and qubit resetting) per second, for the distribution of testing instances, the total time needed for the LSTM to perform its 10 optimization steps is in the range of (700 ± 400) seconds for MaxCut QAOA, (23 ± 6) seconds for Ising QAOA, and (300 ± 200) seconds for Hubbard VQE. Note that the standard deviation here is due to the variations in the Pauli norm of the Hamiltonians for the sampled instances of the test set.

Apart from this added cost function noise, the simulated quantum circuit executions were simulated without any other form of readout or gate execution noise. Plotted in Figure 3 are the 95% confidence intervals for the optimization of the 50 testing instances which were sampled according to the testing distributions described in Sec. IIIA and Sec. IIIB. Our results show that the neural optimizer learns initialization heuristics for the QAOA and VQE parameters which generalize across problem sizes. We discuss these results in further detail in the following section.

Let us provide a description of the alternative optimization and initialization heuristics used to generate Figure 3. First alternative strategy was a Bayesian Optimization using Gaussian processes [62], here the initial parameters are set to nil, same as the was the case for the LSTM optimizer. Next, in order to compare the LSTM to other initialization heuristics, we compared the initialization of Nelder-Mead (NM) at parameter values found from the best of 10 random guesses (Random Seed), NM initialized using some state of the art heuristics for QAOA and VQE (Heuristic Seed), and NM initialized after 10 iterations of the LSTM (LSTM Seed). The application-specific heuristic seeds (Heur. Seed) were the adiabatic heuristic for VQE [67], where the variational parameters are scaled in a similar fashion to an adiabatic interpolation across the 5 steps, while for the QAOA the parameters were initialized at the mean value of the optimal parameters for the training set distribution of problem instances. As was shown in [63], as there is a concentration of the cost function for fixed parameters, one can thus expect the distribution of optimal parameters of the QAOA to be concentrated around some mean.

In Figure IIIC, we compare the Euclidean distance in parameter space between the output of the 10 iterations of the LSTM versus other initialization heuristics. We clearly see that the LSTM optimizer initializes the QNN parameters closer to the optimal parameters of each test instance, on average, as compared to other methods. We see that in the case of the QAOA, the constant fit heuristic [63] for the training instances yields a cluster of parameters that is not clustered around the optimal parameters of the larger instance, while the LSTM output parameters are significantly closer to the globally optimal parameters found by brute force. This shows a clear separation between the parameters obtained from a constant fit of the training set versus the LSTM's adaptive scheme for optimizing parameters in few iterations.

IV. DISCUSSION

As shown in Figure 3, our trained neural optimizer reaches a higher-quality approximate optimum of the parameters in 10 iterations than other optimizers can manage in hundreds, both for noisy and noiseless readout. Most evident in the case of VQE, where the local optimizers can have severe difficulty optimizing parameters when given noisy evaluations of the cost function. Of all alternatives to the neural optimizer, the probabilistic approach of Bayesian optimization via Gaussian process regression was the best performer.

In all six settings, the LSTM rapidly finds an approximate minimum within its restricted time horizon of 10 iterations. The neural optimizer needs to initialize the parameters in a basin of attraction of the cost function landscape so that a local optimizer can then easily converge to a local optimum in fewer iterations and more consistently. As we can see across all cases, the Nelder-

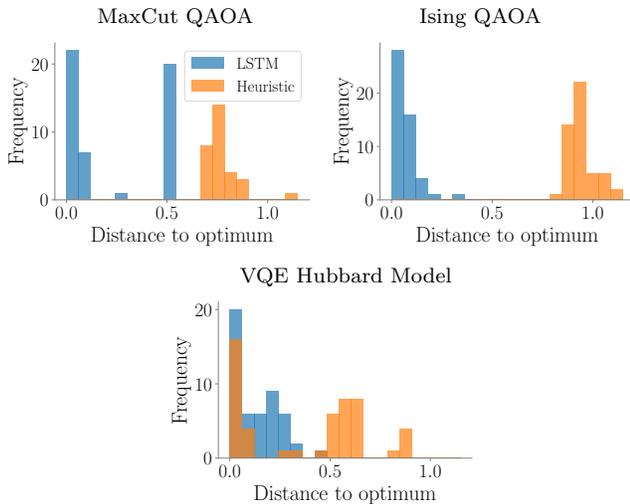


Figure 4. The above histograms represent the parameter space Euclidean distance to the true optimum $d(\theta) = \|\theta - \theta^*\|_2$, immediately after initialization, for both the LSTM (after 10 iterations) and alternative initialization heuristics. For each of the three cases, 50 samples of the test set were used. For the QAOA, this alternative initialization heuristic was identical to [63], whereas for VQE, this was the Adiabatic Heuristic [67]. The histograms were collected from the test set problem instances, which were described in Sections III A and III B, and were used to generate the results presented in Figure 3. We see that the LSTM initializes parameters significantly closer to the optimum on average as compared to alternative heuristics.

Runs that are initialized by the LSTM rather than other initialization heuristics tend to reach the highest quality optimum with much lower variance in performance. These results show that the LSTM initializes the parameters in a good basin of attraction near the optimum. Although the LSTM initialization helps in all cases, we can see that for the noisy VQE case in Figure 3, the Nelder-Mead approach struggles to improve upon the guess of the LSTM. While not the focus of this work, these results point towards the need for further investigation into better local optimization techniques which are robust to noise [73].

When looking at the optimal parameters found by the RNN (see Fig. III C), we observed a certain degree of concentration in their values, similar to what has been observed in previous works [32] [63]. The MaxCut QAOA was the most concentrated, which corroborates recent observations [63]. Similarly, the optimal parameters of the Ising QAOA were also observed to have a degree of concentration, as was also observed in previous work [32]. Finally, the VQE had the least amount of concentration of the three problem classes, but still exhibited some degree of clustering in the optimal values.

The concentration of parameters is not surprising given the connections between QAOA-like ansatzes and gradient descent/adiabatic optimization [28] [60]. In a sense, these QAOA-like QNNs are simply variational methods to de-

scend the energy landscape, and the variational parameters can be seen as energetic landscape descent hyperparameters, akin to gradient descent parameters. Similar to how the classical meta-learning approaches to gradient-based optimization converged onto methods comparable to best-practices for hyperparameter optimization (e.g., comparable to performance of AdaGrad and other machine learning best-practice heuristics), the neural optimizer in our case found a neighborhood of optimal hyperparameters and learned a heuristic to quickly adjust these parameters on a case-by-case basis.

Although it may seem that this meta-learning method is costly due to its added complexity over regular optimization, one must remember that, for the time being, classical computation is still much cheaper than quantum computation. As the optimization scheme generalizes across system sizes, one may imagine training an LSTM to optimize a certain ansatz for small system sizes by quantum simulation on a classical computer, then using the LSTM to rapidly initialize the parameters for a much larger instance of the same class of problem on a quantum computer. This approach may be well worth the added classical computation, as it can reduce the number of required runs to get an accurate answer on the QPU by an order of magnitude or more.

V. CONCLUSION & OUTLOOK

In this paper, we proposed a novel approach to the optimization of quantum neural networks, namely, using meta-learning and a classical neural network optimizer. We tested the performance of this approach on a set of random instances of variational quantum algorithm optimization tasks, which were the Quantum Approximate Optimization applied to MaxCut and Sherrington-Kirkpatrick Ising spin glasses, and a set of Variational Quantum Eigensolver ansatzes for preparation of ground states of Hubbard models.

The neural network was used to rapidly find a global approximate optimum of the parameters, which then served as an initialization point for other local search heuristics. This combination yielded optima of the quantum neural network landscape which were of a higher quality than alternatives could produce with orders of magnitude more quantum-classical optimization iterations. Furthermore, the neural network exhibited generalization capacity across problem sizes, thus opening up the possibility of classical pre-training of the neural optimizer for inference on larger instances requiring quantum processors.

Two significant challenges of quantum neural network optimization in the NISQ era are finding optimization methods that allow for precise fine-tuning of the parameters to hone in on local minima despite the presence of readout noise and to find good initialization heuristics to allow for more consistent convergence of these local optimizers. Given the results presented in this paper, we

believe that this first challenge has been mitigated by our quantum-classical meta-learning approach, while the second challenge remains open for future work. ✓

In terms of possible extensions of this work, the meta-learning approach could be further improved in several ways. One such way would be to use more recent advances in meta-learning optimizer neural networks [74] which can scale to arbitrary problems and number of parameters. This would extend the capabilities of our current approach to optimizing the parameters of arbitrary QNN's beyond Trotter-based/QAOA-like ansatzes with variable numbers of parameters across instances. Another possible extension of this work would be to meta-learn an optimizer for Quantum Dynamical Descent [28], a quantum generalization of gradient descent which takes the form of a continuous-variable QAOA. As our neural optimizer was tested on various QAOA problems successfully, one could imagine applying it to the optimization of the Quantum Dynamical Descent hyperparameters. The latter could be considered *learning to learn with quantum dynamical descent with classical gradient descent*. This would also be a way to generalize the applicability of

our approach to arbitrary QNN optimization tasks. Finally, as a NISQ-oriented alternative to the latter, one could *meta-learn to optimize the hyperparameters for the stochastic quantum circuit gradient descent algorithm* recently proposed by Harrow et al. [31]. We leave the above proposed explorations to future work.

VI. ACKNOWLEDGEMENTS

Circuits and neural networks in this paper were implemented using a combination of Cirq [69], OpenFermion-Cirq [67], and TensorFlow [70]. The authors would like to thank Yutian Chen and his colleagues from DeepMind for providing code for the neural optimizer [34] which was adapted for this work, as well as Edward Farhi, Li Li, and Murphy Niu for their insights, observations, and suggestions. MB and GV would like to thank the team at the Google AI Quantum lab for the hospitality and support during their respective internships where this work was completed. GV acknowledges funding from NSERC.

* Both authors contributed equally to this work.

- [1] J. Preskill, arXiv preprint arXiv:1801.00862 (2018).
- [2] E. Farhi and H. Neven, arXiv preprint arXiv:1802.06002 (2018).
- [3] E. Farhi, J. Goldstone, and S. Gutmann, arXiv preprint arXiv:1411.4028 (2014).
- [4] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. OBrien, *Nature communications* **5**, 4213 (2014).
- [5] N. Killoran, T. R. Bromley, J. M. Arrazola, M. Schuld, N. Quesada, and S. Lloyd, arXiv preprint arXiv:1806.06871 (2018).
- [6] D. Wecker, M. B. Hastings, and M. Troyer, *Phys. Rev. A* **92**, 042303 (2015).
- [7] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, *Nature* **549**, 195 (2017).
- [8] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, arXiv preprint arXiv:1812.01041 (2018).
- [9] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, *New Journal of Physics* **18**, 023023 (2016).
- [10] S. Hadfield, Z. Wang, B. O’Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, arXiv preprint arXiv:1709.03489 (2017).
- [11] E. Grant, M. Benedetti, S. Cao, A. Hallam, J. Lockhart, V. Stojevic, A. G. Green, and S. Severini, *npj Quantum Information* **4**, 65 (2018).
- [12] S. Khatri, R. LaRose, A. Poremba, L. Cincio, A. T. Sornborger, and P. J. Coles, *Quantum* **3**, 140 (2019).
- [13] M. Schuld and N. Killoran, *Physical review letters* **122**, 040504 (2019).
- [14] S. McArdle, T. Jones, S. Endo, Y. Li, S. Benjamin, and X. Yuan, arXiv preprint arXiv:1804.03023 (2018).
- [15] M. Benedetti, E. Grant, L. Wossnig, and S. Severini, *New Journal of Physics* **21**, 043023 (2019).
- [16] B. Nash, V. Gheorghiu, and M. Mosca, arXiv preprint arXiv:1904.01972 (2019).
- [17] Z. Jiang, J. McClean, R. Babbush, and H. Neven, arXiv preprint arXiv:1812.08190 (2018).
- [18] G. R. Steinbrecher, J. P. Olson, D. Englund, and J. Carolan, arXiv preprint arXiv:1808.10047 (2018).
- [19] M. Fingerhuth, T. Babej, *et al.*, arXiv preprint arXiv:1810.13411 (2018).
- [20] R. LaRose, A. Tikku, É. O’Neel-Judy, L. Cincio, and P. J. Coles, arXiv preprint arXiv:1810.10506 (2018).
- [21] L. Cincio, Y. Subaşı, A. T. Sornborger, and P. J. Coles, *New Journal of Physics* **20**, 113022 (2018).
- [22] H. Situ, Z. Huang, X. Zou, and S. Zheng, *Quantum Information Processing* **18**, 230 (2019).
- [23] H. Chen, L. Wossnig, S. Severini, H. Neven, and M. Mohseni, arXiv preprint arXiv:1805.08654 (2018).
- [24] G. Verdon, M. Broughton, and J. Biamonte, arXiv preprint arXiv:1712.05304 (2017).
- [25] Y. LeCun, Y. Bengio, and G. Hinton, *nature* **521**, 436 (2015).
- [26] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, Vol. 1 (MIT press Cambridge, 2016).
- [27] J. Schmidhuber, *Neural networks* **61**, 85 (2015).
- [28] G. Verdon, J. Pye, and M. Broughton, arXiv preprint arXiv:1806.09729 (2018).
- [29] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, *Nature Communications* **9** (2018), 10.1038/s41467-018-07090-4
- [30] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, arXiv preprint arXiv:1811.11184 (2018).
- [31] A. Harrow and J. Napp, arXiv preprint arXiv:1901.05374 (2019).
- [32] Z.-C. Yang, A. Rahmani, A. Shabani, H. Neven, and C. Chamon, *Physical Review X* **7**, 021027 (2017).

- [33] E. Grant, L. Wossnig, M. Ostaszewski, and M. Benedetti, arXiv preprint arXiv:1903.05076 (2019).
- [34] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas, arXiv preprint arXiv:1611.03824 (2016).
- [35] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, in *Advances in Neural Information Processing Systems* (2016) pp. 3981–3989.
- [36] B. Zoph and Q. V. Le, arXiv preprint arXiv:1611.01578 (2016).
- [37] C. Finn, P. Abbeel, and S. Levine, arXiv preprint arXiv:1703.03400 (2017).
- [38] M. Long, Y. Cao, J. Wang, and M. I. Jordan, arXiv preprint arXiv:1502.02791 (2015).
- [39] I. D. Kivlichan, J. McClean, N. Wiebe, C. Gidney, A. Aspuru-Guzik, G. K.-L. Chan, and R. Babbush, *Phys. Rev. Lett.* **120**, 110501 (2018).
- [40] Z. Jiang, K. J. Sung, K. Kechedzhi, V. N. Smelyanskiy, and S. Boixo, *Physical Review Applied* **9**, 044036 (2018).
- [41] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, *SIAM Journal on optimization* **9**, 112 (1998).
- [42] G. Nannicini, *Physical Review E* **99**, 013304 (2019).
- [43] G. G. Guerreschi and M. Smelyanskiy, arXiv preprint arXiv:1701.01450 (2017).
- [44] J. C. Spall, *IEEE Transactions on aerospace and electronic systems* **34**, 817 (1998).
- [45] M. J. Powell, Cambridge NA Report NA2009/06, University of Cambridge, Cambridge, 26 (2009).
- [46] G. Nannicini, arXiv preprint arXiv:1805.12037 (2018).
- [47] J. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, *New Journal of Physics* **18**, 023023 (2016).
- [48] In general, one may relay the raw measurement results to the classical processing unit, which can then compute the expectation value of the cost function. For the purposes of this paper, we assumed the classical optimizer only has access to (noisy) estimates of the expectation value of the cost Hamiltonian.
- [49] A. Y. Kitaev, A. Shen, M. N. Vyalyi, and M. N. Vyalyi, *Classical and quantum computation*, 47 (American Mathematical Soc., 2002).
- [50] N. C. Rubin, R. Babbush, and J. McClean, *New Journal of Physics* **20**, 053020 (2018).
- [51] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, in *Proceedings of the 1988 connectionist models summer school*, Vol. 1 (CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988) pp. 21–28.
- [52] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *nature* **323**, 533 (1986).
- [53] J. Romero and A. Aspuru-Guzik, arXiv preprint arXiv:1901.00848 (2019).
- [54] A. Nichol and J. Schulman, arXiv preprint arXiv:1803.02999 (2018).
- [55] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, arXiv preprint arXiv:1807.05960 (2018).
- [56] C. Audet and M. Kokkolaras, “Blackbox and derivative-free optimization: theory, algorithms and applications,” (2016).
- [57] Z. C. Lipton, J. Berkowitz, and C. Elkan, arXiv preprint arXiv:1506.00019 (2015).
- [58] R. Pascanu, T. Mikolov, and Y. Bengio, in *International Conference on Machine Learning* (2013) pp. 1310–1318.
- [59] S. Hochreiter and J. Schmidhuber, *Neural computation* **9**, 1735 (1997).
- [60] A. Bapat and S. Jordan, arXiv preprint arXiv:1812.02746 (2018).
- [61] G. Verdon, J. M. Arrazola, K. Brádler, and N. Killoran, arXiv preprint arXiv:1902.00409 (2019).
- [62] C. E. Rasmussen, in *Advanced lectures on machine learning* (Springer, 2004) pp. 63–71.
- [63] F. G. Brandao, M. Broughton, E. Farhi, S. Gutmann, and H. Neven, arXiv preprint arXiv:1812.04170 (2018).
- [64] M. Suzuki, *Physics Letters A* **146**, 319 (1990).
- [65] T. Kadowaki and H. Nishimori, *Physical Review E* **58**, 5355 (1998).
- [66] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, *Physical Review X* **8**, 021050 (2018).
- [67] J. R. McClean, I. D. Kivlichan, D. S. Steiger, Y. Cao, E. S. Fried, C. Gidney, T. Häner, V. Havlíček, Z. Jiang, M. Neeley, *et al.*, arXiv preprint arXiv:1710.07629 (2017).
- [68] L. Prechelt, in *Neural Networks: Tricks of the trade* (Springer, 1998) pp. 55–69.
- [69] M. LLC, “Cirq: A python framework for creating, editing, and invoking noisy intermediate scale quantum circuits,” (2018).
- [70] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*
- [71] S. Ioffe and C. Szegedy, arXiv preprint arXiv:1502.03167 (2015).
- [72] D. J. Wales and J. P. Doye, *The Journal of Physical Chemistry A* **101**, 5111 (1997).
- [73] P. Omalley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding, *et al.*, *Physical Review X* **6**, 031007 (2016).
- [74] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. de Freitas, and J. Sohl-Dickstein, in *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (JMLR. org, 2017) pp. 3751–3760.