Deep recurrent networks predicting the gap evolution in adiabatic quantum computing

Naeimeh Mohseni[1,2], Carlos Navarrete-Benlloch [*3,4,1], Tim Byrnes [*5,6,7,8,9], and Florian Marquardt[1,2]

[1]Max-Planck-Institut für die Physik des Lichts, Staudtstrasse 2, 91058 Erlangen, Germany
[2]Physics Department, University of Erlangen-Nuremberg, Staudtstr. 5, 91058 Erlangen, Germany
[3]Wilczek Quantum Center, School of Physics and Astronomy, Shanghai Jiao Tong University, Shanghai 200240, China
[4]Shanghai Research Center for Quantum Sciences, Shanghai 201315, China
[5]New York University Shanghai, 1555 Century Ave, Pudong, Shanghai 200122, China
[6]State Key Laboratory of Precision Spectroscopy, School of Physical and Material Sciences,East China Normal University, Shanghai 200062, China
[7]NYU-ECNU Institute of Physics at NYU Shanghai, 3663 Zhongshan Road North, Shanghai 200062, China
[8]Center for Quantum and Topological Systems (CQTS), NYUAD Research Institute, New York University Abu Dhabi, UAE
[9]Department of Physics, New York University, New York, NY 10003, USA

In adiabatic quantum computing finding the dependence of the gap of the Hamiltonian as a function of the parameter varied during the adiabatic sweep is crucial in order to optimize the speed of the computation. Inspired by this challenge, in this work we explore the potential of deep learning for discovering a mapping from the parameters that fully identify a problem Hamiltonian to the aforementioned parametric dependence of the gap applying different network architectures. Through this example, we conjecture that a limiting factor for the learnability of such problems is the size of the input, that is, how the number of parameters needed to identify the Hamiltonian scales with the system size. We show that a long short-term memory network succeeds in predicting the gap when the parameter space scales linearly with system size. Remarkably, we show that once this architecture is combined with a convolutional neural network to deal with the spatial structure of the model, the gap evolution can even be predicted for system sizes larger than the ones seen by the neural network during training. This provides a significant speedup in comparison with the existing exact and approximate algorithms in calculating the gap.

1 Introduction

Machine learning based on neural networks has demonstrated significant predictive capability for many challenging problems [18, 29]. In particular, its application in studying quantum many-body systems has attracted significant attention in the last few years [13]. A few notable successes include the use of neural networks in identifying quantum phases of matter and learning phase transitions [6, 12, 51, 53, 54], quantum state tomography [17, 50], enhancing quantum Monte Carlo methods [9, 25], solving optimization problems [2, 32], quantum control [10], the efficient representation of quantum states [11, 47], and tackling quantum many-body dynamics [24, 35, 47, 52].

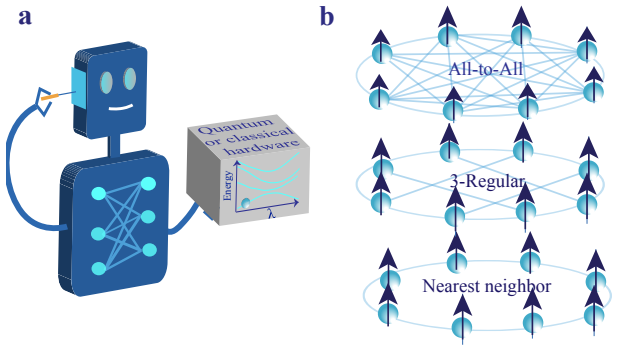


Figure 1: (a) A neural network learns the parametric gap by observing the data generated by a classical computer or a quantum computer. For the purposes of this work, we train the network on the data generated by a classical computer. (b) Spin models studied in this work that are taken to be the problem Hamiltonian in AQC. Spins are connected with random couplings.

Adiabatic quantum computing (AQC) is an example of a quantum many-body dynamical process and was proposed as an approach for solving optimization problems [15, 36]. Starting from the ground state of a reference Hamiltonian, a parameter of the system is varied adiabatically until reaching a target or problem Hamiltonian, whose ground state encodes the solution of the optimization problem. The sweep time in AQC for which adiabaticity can be achieved is proportional to a negative power of the minimum energy gap between the two lowest energy levels of the inter-

mediate Hamiltonians that the system goes through during the sweep [1, 30, 42]. Therefore, one way of optimizing the computation time is by spending the majority of the evolution time in the vicinity of any anti-crossings. Creating such an annealing schedule requires prior knowledge of the dependence of the gap on the parameter that is swept, which we call the *parametric gap*, and which is known to be as hard as solving the original problem [5]. There are a handful of cases for which analytical expressions for the gap exist [26, 42, 46, 48], but mostly the gap can only be determined numerically either based on exact diagonalization or approximate algorithms [4, 34, 56] which are limited to relatively small system sizes. Therefore, in general there is no intuition about how the parametric gap is related to the structure of the problem Hamiltonian.

It remains an interesting open problem to characterize the complexity of learning the parametric gap by applying deep learning, where one can think of two scenarios: training the network on either the data generated by a quantum computer or numerical simulations (Fig. 1 (a)). In the former case, we have a hybrid quantum-classical algorithm. Such algorithms have recently attracted a lot of interest and have been examined in different contexts [23, 35, 39], as they can be applied for regimes for which the run-time of the exact numerical simulations is prohibitive, while the network can still be trained on the data generated via experiment. It is also interesting to explore how the difficulty of the task is related to the complexity of the problem Hamiltonian and what are the models for which a neural network can assist in approximating the gap.

In this work, we address these open questions by exploring the performance of different network architectures trained on the parametric gap for many different realizations of a random problem Hamiltonian. For the purposes of this work, we train the neural network on the data generated from numerical simulations rather than quantum hardware. However, our methods could equally be applied in a quantum-classical hybrid scenario, given the availability of the desired hardware.

As for the problem Hamiltonian in AQC, we consider spin models as test cases. We find that LSTM networks, which are typically used for sequence processing and prediction, including audio signal analysis [19] and language translation [33] excel in predicting the gap evolution in certain scenarios. These architectures are known to be good at capturing both long-term and short-term dependencies in sequences. This characteristic is extremely useful as it gives the LSTM network the power to handle complex dynamics. Remarkably, we demonstrate combining LSTM with a convolutional network called CONVLSTM [55] allows extrapolation to larger system sizes in some settings. This architecture combines a convolutional neural network (CNN) to deal with the spatial structure of the input with the LSTM that tracks the time evolution.

While throughout this work we concentrate on the particular task of gap prediction in AQC, our study provides insight into more general many-body dynamics. We conclude that an important limiting factor for the learnability of such dynamics is the way in which the number of parameters is needed to identify the Hamiltonian scales with the system size.

2 Problem definition

In this section, we define the models that we explore in this work. We also introduce the neural network architectures that we apply and explain how we train them.

2.1 Model

We consider the AQC Hamiltonian defined as

$$H(\lambda) = (1-\lambda)H_0 + \lambda H_{\mathrm{p}}, \tag{1}$$

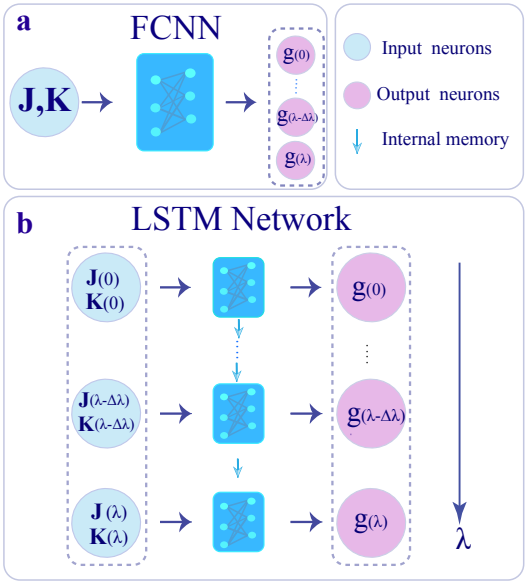
with

$$H_{\mathrm{p}} = \sum_{i,j=1}^{M} J_{ij}\sigma_i^z\sigma_j^z + \sum_{i=1}^{M} K_i\sigma_i^z, \tag{2a}$$

$$H_0 = -\sum_{i=1}^{M} \sigma_i^x, \tag{2b}$$

where σ_i^α with $\alpha = x, z$ are Pauli operators, and J_{ij} and K_i are random coefficients that identify the problem Hamiltonian. In practice, AQC considers a time-varying parameter λ that is swept from 0 to 1, following some schedule $\lambda(t)$. However, this specific function of time is irrelevant for our purposes, since our goal is finding a mapping from the parameters (J_{ij}, K_i), which we collect in a matrix $\boldsymbol{J}$ and a vector $\boldsymbol{K}$, to the gap between the ground and first excited states of $H(\lambda)$ parametrized as a function of λ. In the following we denote this parametric gap by $g(\lambda)$. We study in particular two limiting cases for the problem Hamiltonian: i) all-to-all connected spin models and ii) 1D nearest neighbor connected models and iii) 3-Regular graphs shown in Fig. 1 (b). Note that all parameters are dimensionless since they are normalized to the energy scale of H_0, which we take as the reference.

Let us emphasize that while we are here considering the problem of parametric gap prediction in AQC, the same approach can be applied to find a mapping from the (possibly time-dependent) parameters of any many-body Hamiltonian, to the temporal dynamics of some observable. One simply has to train the network on trajectories of the observable of interest, taking time as a parameter that would play the role of λ in our AQC example.



h

Figure 2: Schematic representation of the neural networks considered in this work. (a) Fully connected neural network (FCNN) with the bare parameters $\boldsymbol{J}$ and $\boldsymbol{K}$ as the input and the parametric gap as the output. (b) The long short-memory (LSTM) network with $\boldsymbol{J}(\lambda) = \lambda\boldsymbol{J}$ and $\boldsymbol{K}(\lambda) = \lambda\boldsymbol{K}$ as the input and the gap as the output denoted by $g(\lambda)$. Arrows between the modules (dark blue cells) indicate the content of the internal neural memory being passed from a given value of λ to the next considered value, which we take $\Delta\lambda$ apart.

2.2 Neural network architectures

We apply three neural network architectures for our goal: an FCNN, an LSTM network, and a CONVLSTM network. For the FCNN architecture, we feed into the neural network the parameters $\boldsymbol{J}$ and $\boldsymbol{K}$ that fully identify the problem Hamiltonian and the neural network provides as output the parametric gap (Fig. 2 (a)), which we signify by

$$(\boldsymbol{J}, \boldsymbol{K}) \xrightarrow{\text{FCNN}} g(\lambda). \tag{3}$$

In the LSTM architecture, the input is not just the bare parameters that identify the problem Hamiltonian, but the effective contribution of these parameters during the whole sweep, namely $\lambda\boldsymbol{J}$ and $\lambda\boldsymbol{K}$ (Fig. 2 (b)). These coefficients effectively identify the full adiabatic Hamiltonian during the sweep. In this case, then we have

$$(\lambda\boldsymbol{J}, \lambda\boldsymbol{K}) \xrightarrow{\text{LSTM}} g(\lambda). \tag{4}$$

The most important difference between these two architectures is that the LSTM receives as input the effective contribution of the parameters for each value of λ up to the one it wants to compute, eventually working its way sequentially through the whole $\lambda \in [0, 1]$ interval. In contrast, the FCNN has to work out the full parametric gap at once. In addition, as shown in Fig. 2(b), the LSTM architecture is composed of modules (dark blue cells). Each module is made of a few gates (see Sec. II of the Supplemental Material for more details) which decide on the flow of information in and out of each module at each time.

To study the possibility of extrapolating the predictions to system sizes beyond what the neural network has been trained on we also apply a CONVLSTM network [55]. This architecture is designed for data with spatio-temporal input [49]. It combines a CNN to deal with the spatial structure of the input with the LSTM that tracks the "evolution" in λ. CNNs can be applied for variable input size therefore helping to scale up the predictions to larger sizes [18, 38] (See Supplemental Material Secs. I and II for more details). Inspired by this feature and using an appropriate preparation of the input such that we can present properly the spatio-temporal structure of the input to the network, we explore the power of the CONVLSTM in predicting the gap for larger system sizes than that it has been trained on. The input and the output of the network in this case are signified as

$$(\lambda\boldsymbol{J}(x), \lambda\boldsymbol{K}(x)) \xrightarrow{\text{CONVLSTM}} g(\lambda), \tag{5}$$

where $\boldsymbol{J}(x)$ and $\boldsymbol{K}(x)$ should properly present the spatial structure of the problem Hamiltonian as we explain in more detail in Sec. 4.

2.3 Training

To train the neural network, we generate a set of random parameters $\boldsymbol{J}$ and $\boldsymbol{K}$ for the particular system size of interest. We then diagonalize the Hamiltonian for the desired values of λ and calculate the first two eigenvalues, whose difference provides the parametric gap. All these parameters are taken from a uniform distribution in the interval $[-1, 1]$. To improve the performance of the neural network at regions where the gap is smaller, we train it on the $\log(g(\lambda) + 1)$ rather than $g(\lambda)$.

Out of the generated random instances, we keep a set to evaluate the neural network, which we call the test set, and a set for validation. The validation set is used to fine-tune the hyperparameters of the neural network, but no training occurs on this set. The remaining instances are used for training. To evaluate the performance of the neural network, we calculate the mean square error MSE on our test set defined as

$$\text{MSE} = \langle |\log(g_{\text{true}}(\lambda) + 1) - \log(g_{\text{predict}}(\lambda) + 1)|^2 \rangle_{n,\lambda}, \tag{6}$$

where the g_{true} and the g_{predict} denote the true gap calculated by diagonalizing the Hamiltonian and the predicted gap obtained by the neural network, respectively. The average is taken over the number of instances n and the parameter λ.

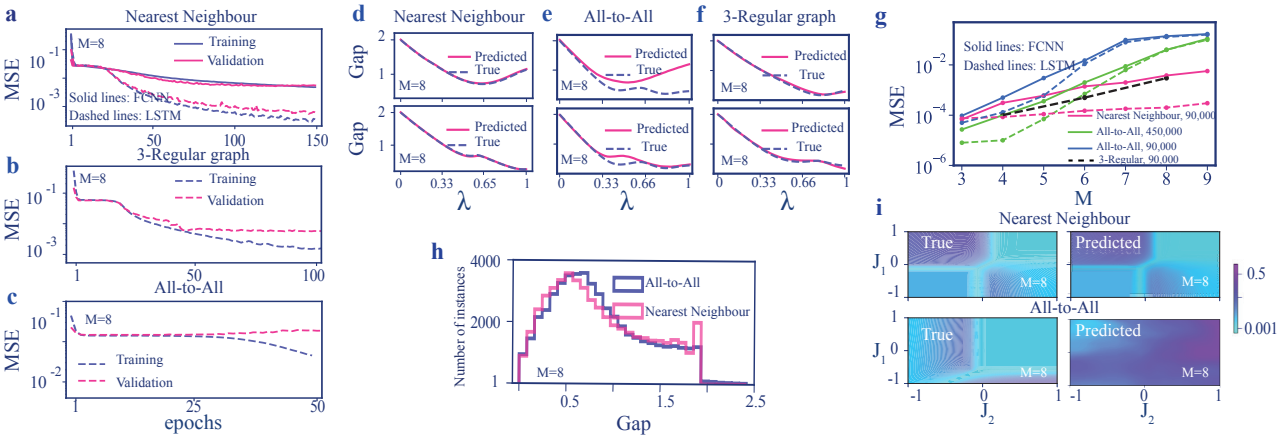


Figure 3: Comparing the performance of the LSTM network and FCNN in predicting the parametric gap for the all-to-all model, the nearest neighbor connected model, and 3-regular graph. Validation MSE versus epochs for system size $M = 8$ applying the FCNN (solid lines) and LSTM network (dashed lines) for (a) the nearest neighbor, (b) 3-regular graph, (c) all-to-all connected models. 90,000 samples are used to train the neural network and 10,000 samples are used for the validation. Predicted gap (solid line) and exact gap (dashed line) in terms of λ with $\tau = 30$ for a few typical problem instances for the (d) nearest neighbor connected model, (e) all-to-all connected model, and (f) 3-regular graph. (g) The error MSE versus system size on the test set (containing 1000 samples for each system size) for all models applying both FCNN (solid curves) and LSTM network (dashed curves). The numbers on the legend denote the training set size. The shown points are average over 5 attempts for training. (h) Histogram of the gap during the whole sweep for both the nearest neighbor and all-to-all connected models. (i) The true and predicted minimum gap versus couplings J_1 and J_2, where all other couplings and local terms are fixed.

3 Gap evolution learnability

In this section, we study the learnability of the gap by applying the different neural network architectures that we discussed above. We first study the performance of our neural network architectures in predicting the gap for spin models in two extreme limits: either all spins are connected or only nearest neighbors in 1D shown in (Fig. 1 (b)). In addition, we study the learnability of the gap for 3-regular graphs. We inspect how the prediction accuracy scales with system size for these models applying our different network architectures.

In Fig. 3(a), (c), we show the MSE over training and validation sets in terms of epochs for the nearest neighbor and all-to-all connected models. The number of epochs indicates the number of times that the network has seen all the training instances. For the nearest neighbor connected model, it can be seen that for both LSTM network and FCNN the error decreases with the number of epochs. However, it is evident that the LSTM network performs better (Fig. 3(a)). In contrast, for the all-to-all connected model, the network overfits (Fig. 3(c)). For this model, we just showed the results using LSTM network as we observed the same behavior applying the FCNN as well. Investigating different factors such as the network size and the training set size, we learned that the latter is the main bottleneck in this case.

In Figs. 3(d) and (e), we show the predicted and the true gap for a few typical instances applying the LSTM network. It is clear that our network fails to predict the gap for the all-to-all connected instances while successfully predicts the gap with high accuracy for the nearest neighbor-connected instances.

In Fig. 3 (g), we study how the error scales with the system size for a fixed number of training samples specified in the legend. The shown errors are found by averaging over 1000 test instances. Let us focus first on the nearest neighbor-connected model. It is obvious that the LSTM network has a considerably higher precision and the error in prediction scales more favorably in terms of system size for this architecture in comparison with the FCNN (compare the pink solid line with the pink dashed line). We attribute this to the fact that the LSTM architecture has memory and is able to record both long and short-term dependencies. This architecture has the built-in notion of causality, while the FCNN needs to learn causality on its own as it has no notion of time. Applying the same number of training samples for the all-to-all connected model (blue lines), the error grows more dramatically with system size in comparison with the nearest neighbor-connected model. The error can be decreased by increasing the training set size, but even a factor of 5 (green lines) is not enough to achieve reasonable accuracy for larger system sizes (say $M > 6$). This implies that the number of samples required to train the network explodes with the system size when aiming at a given error in the predictions. As a consequence of this explosion, the LSTM network does not seem to show a better performance in comparison with the FCNN for $M > 5$ in the figure when 90,000 instances are used for training (compare dashed and solid blue lines).

We have explored some reasons why the network fails for the all-to-all connected model, while it succeeds for the nearest neighbor model when for a given system size M the Hilbert space dimension is the same for both models. We conjecture the main reason is due to the way in which the parameter space size scales with the system size for each model: linearly for the nearest neighbor model and quadratically for the all-to-all connected one. Note that by parameter space size we mean the number of parameters that identify the model. One may think that another reason might be that the nearest neighbor connected model can be also simpler in terms of the gap size and complexity of the gap trajectories. To investigate this, we compare the gap size during the sweep for both the nearest neighbor and the all-to-all connected models for system size $M = 8$ in Fig. 3 (h). The plot is made for one random instance of each model, but similar figures are found for other instances. It is apparent that, in general, the typical size of the gap is similar for both models. Ideally, one would wish to investigate whether the gap in the fully connected case is more feature-rich compared with the nearest neighbor case by analyzing the dependence of the minimum gap with all parameters $\boldsymbol{J}$ and $\boldsymbol{K}$. Unfortunately, this is computationally unfeasible. Therefore, we consider a simpler analysis, in which we analyze the dependence of the minimum gap with two of the parameters only, e.g. J_1 and J_2, for different random instances of all other parameters. For all random instances, we find a dependence similar to that shown in Fig. 3 (i). As can be seen, the pattern of the minimum gap in the all-to-all connected model does not appear to be more complicated than the nearest neighbor connected model. However, the network fails in predicting the minimum gap for the all-to-all connected model. This seems to further support the idea that the all-to-all connected model is not more feature-rich than the nearest-neighbors one. Another potential reason is that the local nature of the connectivities in the nearest neighbor model can make it easier for the network to learn the dynamics of the gap. If that were to be true, then one should expect an improvement by encoding the all-to-all connected model into a local model. In Sec. 5, we introduce such an encoding through the so-called Lechner-Hauke-Zoller (LHZ) mapping [28], and find the same scaling of the error with the system size.

To further explore the validity of our conjecture — that the success of the network is connected with the way that parameter space size scales with system size — we also studied 3-regular graphs for which each spin is connected to three other spins. In Fig. 3(b), we show the MSE over training and validation sets in terms of epochs for the LSTM network. In Fig. 3(f), we show the predicted and true gap for two typical instances. In Fig. 3(g), the black curve shows the performance of our LSTM network in predicting the gap for this model where for each system size we averaged over 1000 realizations. In overall, the precision in learning the gap on this model is lower in comparison with the nearest neighbor connected model. Nevertheless, the network is able to make predictions with reasonable accuracy. The reason for the lower precision is that for the 3-regular graph, the number of connectivities scales as $3M/2$ while it scales as M for the nearest neighbor connected model. Therefore, to get comparable accuracy for the predictions, one needs to provide more samples to train the 3-regular graph model.

Note that our conjecture that the learnability of the network is connected with the parameter space size, i.e. the number of connectivities, is consistent with well-known aspects of optimization problems. For optimization problems, the complexity of the problem increases by increasing the density of the problem, namely increasing the connectivities of the problem Hamiltonian [7, 40, 41].

As a last remark of this section, one may argue that it might be easier for the network to learn the times for which the gap becomes small, rather than the full parametric dependence of the gap. We have also investigated this and observed that the network still fails for the all-to-all connected model. Our conjecture is that this is again a consequence of the quadratic scaling of the parameter space size with the system size.

4 Extrapolation

In this section, we explore the possibility of the network to predict the gap for system sizes beyond those in which it is trained. As indicated already, the potential architectures for this goal are CNNs, which can be applied for an input with variable size. These architectures are known to be good for extracting features on local models. Therefore, we apply to the nearest neighbor connected model the CONVLSTM architecture which is designed for sequence prediction problems with spatial structure. This network extends the fully connected LSTM architecture to have a convolutional structure in both the input-to-module (dark blue sheet Fig. 4(a)) and module-to-module (light blue sheet Fig. 4(a)) transitions.

In Fig. 4(a), we show the input of the network with a 1D spatial structure and containing three features at each site. The first feature represents the corresponding local parameter $\boldsymbol{K}$ at each site. The second and third ones represent the couplings $\boldsymbol{J}$, arranged so as to emphasize that each site is connected to its two nearest neighbors. See Supplemental Material Sec. III for more details on the technical implementation and the layout of the network.

We train the network on system sizes $M \in [3, 9]$, and then evaluate it on the test samples with system sizes $M \in [3, 22]$. We observed that our CONVLSTM network succeeds in predicting the gap both for the system sizes that it has been trained on and larger sys-

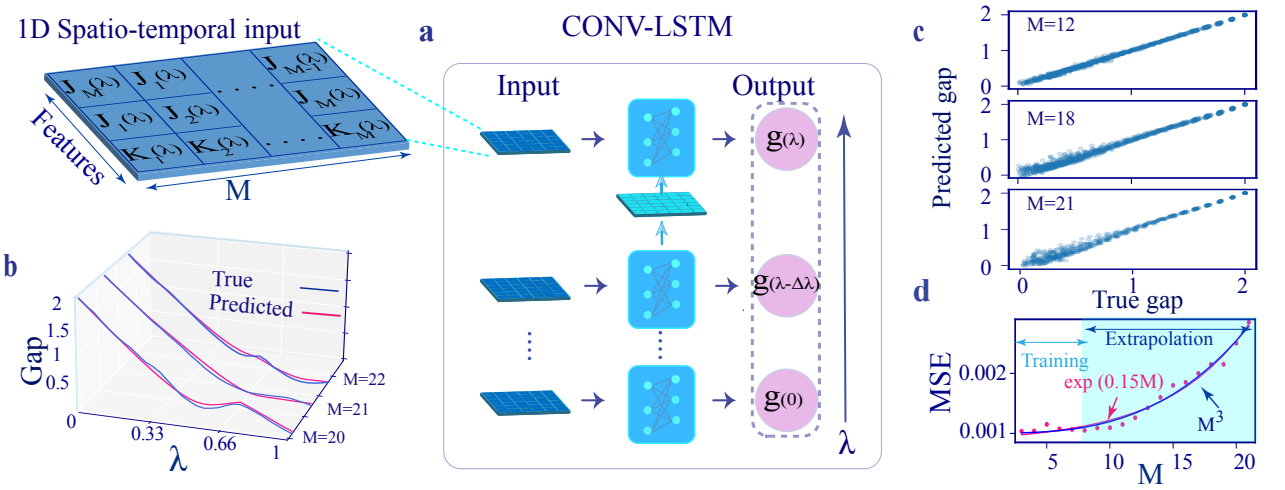


Figure 4: Extrapolating the gap to the larger system sizes applying 1D-CONVLSTM network for the nearest neighbor connected model. (a) Schematic representation of the CONVLSTM with spatio-temporal inputs (light blue sheets). Arrows between the modules (dark blue cells) indicate the content of the internal neural memory being passed one value of λ to the next $\lambda + \Delta\lambda$ and the dark blue sheet denotes a convolutional structure for module-to-module transitions. (b) Predicted (solid line) and true gap (dashed line) in terms of λ with $\Delta\lambda = 1/30$ for three typical problem instances. (c) Predicted gap versus the true gap for all values of λ and 20 random problem instances for each system size. (d) The MSE error in predicting the gap versus system size. Network is trained on system sizes with $M \in [3, 9]$ and is evaluated on test samples with $M \in [3, 22]$. The shown errors are averaged over 200 random problem instances for each system size. The highlighted region marks the system sizes that the network has not been trained on. 90,000 samples are used to train the network.

tem sizes than that. In Fig. 4 (b), we show the predicted gap (solid line) and the true gap (dashed line) as a function of λ for three typical problem instances with $M = 20, 21, 22$. As can be seen, the network is able to approximate the gap during the evolution with good precision. We also find that there is a correlation between the size of the gap and the accuracy of the network. In Fig. 4 (c), we show the true gap against the predicted gap for 20 test instances and all considered values of λ. It is clear that close to the regions where the gap is smaller, the performance of the network is less accurate, even on an absolute level.

In Fig. 4 (d), we show how the prediction error in prediction scales with the system size. The highlighted region marks the system sizes that the network has not been trained on. We have not been able to conclude whether the error scales polynomially or exponentially, since both functions fitted relatively well.

In the previous section, we observed for the all-to-all connected model, using 90,000 samples for training, the network is able to learn the gap on small system sizes with $M < 7$. We are interested to study whether a CONVLSTM network that is trained on that small sizes can still extrapolate the predictions to the larger system sizes. To apply the CONVLSTM network for the all-to-all connected model, we need to map it to a local model. This is because CNNs are good at extracting features on the local models as neurons of different layers are locally connected. In the next section, we map the all-to-all connected model to a local model and explore the potential of CONVLSTM for extrapolating the predictions to larger system sizes.

5 Mapping the all-to-all connected model to a local model

In this section, we first investigate whether the highly non-local nature of the all-to-all connected model is one of the reasons that makes it hard for the network to predict the gap. To understand this, we apply the LHZ mapping [28] to encode this model into a model with only local connectivities and check if the network performs better. LHZ maps the graph of the M all-to-all connected logical qubits (Fig. 5 (a)) onto a planar graph with $N_p = M(M-1)/2$ physical qubits and only local connectivities (Fig. 5 (b)). Within this mapping, the original problem Hamiltonian (2a) can be encoded into the following Hamiltonian in the physical qubit basis

$$H_{\mathrm{p}}^{\mathrm{LHZ}} = \sum_{k=1}^{N_p} J_k \sigma_k^z - C \sum_{\langle i,j,k,l \rangle} \sigma_i^z \sigma_j^z \sigma_k^z \sigma_l^z, \qquad (7)$$

where $\langle i, j, k, l \rangle$ denotes sum over nearest neighbor spins. Energy penalties in the second term which involve $N_p - M + 1$ four-body interactions (the four qubits around each small pink circle shown in Fig. 5 (b)) are introduced to ensure that in the $C \to \infty$ limit the low-energy sector of $H_{\mathrm{p}}^{\mathrm{LHZ}}$ reproduces the spectrum of the original Hamiltonian (in practice, we have checked that $C = 3$ suffices to observe the convergence of the ground state in all the cases we have

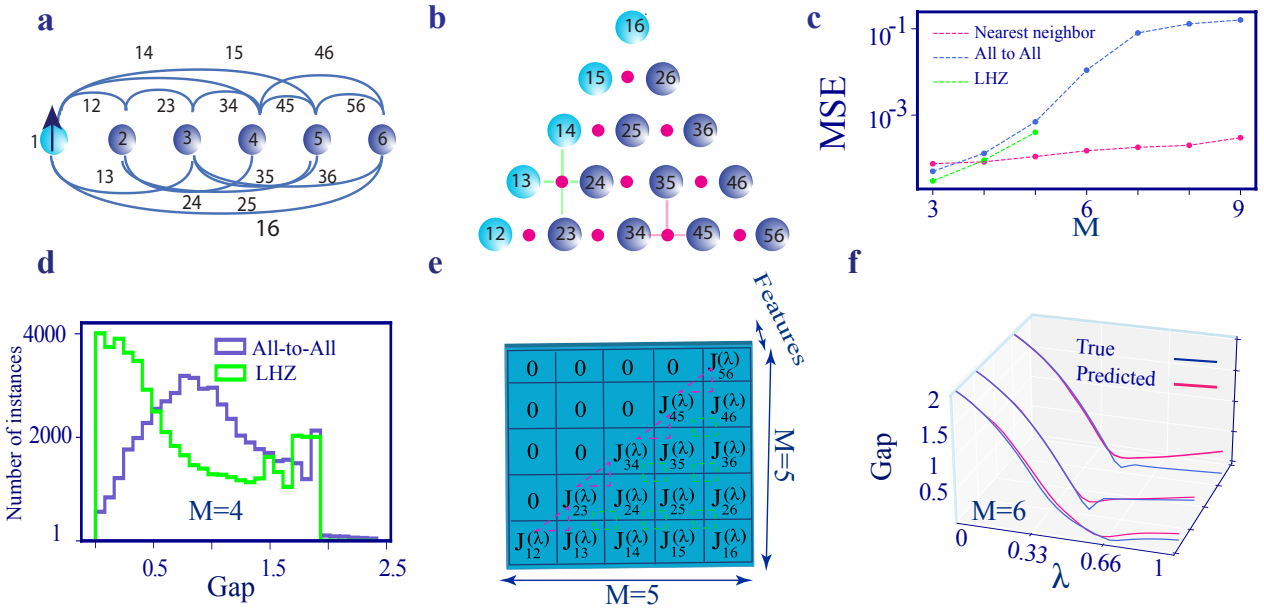


Figure 5: The power of the LSTM network in predicting the parametric gap for the LHZ model with $C = 3$. (a) An all-to-all connected model with system size $M = 5$ where the light blue circle shows an auxiliary logical qubit (fixed to 1) to implement single-body terms. (b) The LHZ architecture where the light blue circles show physical qubits that encode single-body terms. The four qubits around each small pink circle (plaquette) consist of four-body constraints. (c) The error for predicting the parametric gap averaged over 1000 realizations for all three investigated models versus system size applying LSTM network. 90,000 instances are used to train the network for all the shown models and system sizes. (d) Histogram of the gap during the sweep for the all-to-all connected and LHZ models for system size $M = 4$. (e) Spatio-temporal input of the 2D-CONVLSTM containing one feature at each site which presents the couplings. The couplings are arranged on the shown square such that they respect the connectivities in (b). The dashed squares and three-angles denote connections between qubits that identify the three-body and four-body constraints on LHZ model. The square at the top left shows the second feature which identifies for the network where a qubit exists. (f) Exact gap and predicted gap (applying 2D-CONVLSTM) in terms of λ with $\Delta\lambda = 1/30$ for three particular problem instances with size $M = 6$ where the network has not been trained on.

studied). In the bottom row only three qubits are connected, for which three-body constraints are introduced instead.

Single-body terms in the original model, which correspond to a local field acting on the logical qubits, can be also implemented by adding an auxiliary logical qubit (light blue qubit in Fig. 5 (a)) fixed to state $|0\rangle$, which is an eigenstate of σ_z with eigenvalue $+1$. Interaction of this auxiliary logical qubit with the rest of the qubits implements these local terms, which correspond to M extra physical qubits (light blue qubits in Fig. 5 (b)) in the LHZ architecture [28].

In Fig. 5 (c), we compare the performance of the LHZ model with the originally all-to-all connected model applying the standard LSTM network. 90,000 instances are applied to train the network for each model and system sizes up to $M = 5$, for which the LHZ model already contains 15 physical qubits. We also show the results for the nearest neighbor connected model for the purpose of comparing the scaling. We observe that for the LHZ model the error scales with the system size in a similar way to the original model. While we have not been able to analyze for larger M values, if locality played a strong role, we see no reason why it should not be visible even for small system sizes. Therefore overall, these results suggest that the locality of the connectivities should not improve the way that error scales with the system size.

In Fig. 5 (d), we compare the typical size of the gap for both models. As can be seen, the LHZ model has substantially more instances with smaller gap sizes as compared to the all-to-all connected model, but still, the network achieves comparable or even better precision. This confirms our conjecture that the main bottleneck is not the size of the gap, but rather the way in which the parameter space size scales with the system size.

Now we explore whether mapping the all-to-all connected model to a local model and applying a CONVLSTM network helps to extrapolate the predictions to larger system sizes. For this case, we need a 2D-CONVLSTM as the LHZ model has a 2D spatial structure. In Fig. 5 (e), we show the spatio-temporal input of our 2D-CONVLSTM for the case with $M = 5$. Each site contains one feature which represents the coefficients $\boldsymbol{J}$. These couplings are arranged on the shown square such that they respect the connectivities in the LHZ model. Dashed squares and triangles mark qubits around each small pink circle in Fig. 5 (b). We

train the network on system sizes $M \in \{3, 4, 5\}$ and evaluate it on system size $M = 6$ (Fig. 5 (e)), which is the largest size (includes 21 physical qubits) that we are able to generate a few samples for. As can be seen, the network is still able to predict the gap for a larger system size that it has not been trained on, but not with a high accuracy such as in the nearest neighbor connected model. Due to the numerical limitations, we have not been able to evaluate our network for larger system sizes, but considering the low precision for $M = 6$, we expect the network to fail for the larger sizes. We attribute this partially again to the fact that the quadratic scaling of the parameter space with system size necessitates exponentially growing resources for training. However, we expect our 2D CONVLSTM succeeds in predicting the dynamics of 2D models for which the parameter space size scales more favorably with the system size.

6 Speedup

In this section, we discuss the potential speedup that can be achieved by employing the network for the nearest-neighbor model, for which we have shown the network succeeds in making predictions and in extrapolating.

Supervised training of neural networks can be motivated in many ways, e.g. an explicit algorithm to turn the input into the desired output may not even be known in principle (as in image classification). However, for a scenario like the one discussed here, a numerically exact algorithm exists. One can then think of at least three reasons why training and deploying a network might still be beneficial: (i) the run-time of the network is so much shorter than the run-time of the exact algorithm, and we will use it for prediction on so many problem instances, that it is worth to invest the numerical effort needed to generate a large number of training examples in the first place; (ii) the run-time of the algorithm scales very unfavorably with system size and the network is able to make reasonably accurate predictions also for larger sizes, even if it was not trained on those; (iii) there is a regime where the run-time of the exact algorithm is prohibitive, but data may be generated in another way (e.g. via experiments), and the network can still be made to learn in this regime and be used for prediction. We will discuss the last point (training from experiments) in the following subsection and focus on the numerical speedup here.

A real advantage will be obtained if the eventual number of problem instances N_{use} that we intend to apply the network to is sufficiently large. Specifically, if N_{train} is the number of training samples we needed to achieve good accuracy, we have to fulfill the inequality

$$N_{\text{train}}(\tau_{\text{Alg}} + \tau_{\text{train}}) + N_{\text{use}}\tau_{\text{NN}} < N_{\text{use}}\tau_{\text{Alg}},$$

where τ_{Alg} is the run-time of the algorithm itself and τ_{NN} is that of the neural network, for one problem instance. τ_{train} denotes training time spent per one training sample during all epochs. If a network trained on small systems can also be applied to larger system sizes, where the algorithm run-time becomes $\tau_{\text{Alg,Large}}$, we need to insert that larger value on the right-hand-side, making application of the neural network more favorable (even if its own run-time might also increase somewhat).

With this in mind, let us provide some illustrative numbers with all the caveats regarding the dependence on computer hardware and algorithm. We trained the network on small system sizes, which is inexpensive, taking just a few hours to generate $N_{\text{train}} = 90,000$ samples, plus about one day to train the network. The largest system size for which we have been able to generate a few tens of instances to evaluate the neural network is $M = 22$. For this size, it takes one and a half hours to calculate the gap evolution for a single instance when applying the Lanczos algorithm. In contrast, once the neural network is trained, it is able to predict the gap evolution for this system size in 5ms. Assuming a scenario with these illustrative numbers, we can use the formula above to conclude that the application of the neural network would become favorable if it were deployed on $N_{\text{use}} > 20$ actual problem instances which is notably less than the number of training samples, due to the performance gains via extrapolation. Any application on more instances would yield strong time (and memory) savings in comparison to direct use of the algorithm.

7 Hybrid algorithm

Our protocol is adaptable for training neural networks on larger system sizes using data generated from a quantum annealer. In this section, we discuss the near-term feasibility and associated costs of a hybrid implementation. We caution that there are caveats to consider and highlight. In general, measuring the gap using a quantum annealer is currently not straightforward. However, there are proposals for achieving this in the future. Therefore, the ideas presented in this section should be considered rough estimations and only applicable when such proposals are implemented.

Let us start commenting on the feasibility and cost of measuring the gap in experiments. This can be done by applying different methods. Spectroscopic techniques are widely used for this purpose, for example using Ramsey-like interferometry [31, 43]. In all methods, for each value of λ we want, one needs to

measure an observable by repeating the experiment n times, resulting in an error that scales as $\sim 1/\sqrt{n}$, set by the projection noise. Including the number N of samples that we need to prepare for training, overall we then need $nN_\lambda N$ runs, where N_λ stands for the number of values of λ that we take. The number of training samples depends on the system size M we want to train the neural network on. Since in this work we have trained the network on sizes up to $M = 9$, it is not easy to see a clear scaling for the number of samples required to achieve a given accuracy. Therefore, we roughly estimate $N \sim 2 \times 10^5$ from our experience assuming one trains the network on the problem instances with $M = 50$. Then, taking $N_\lambda \sim 50$ and $n \sim 10^4$, one would require around 10^{11} runs. This implies that it seems feasible to train the network on data generated from quantum annealers implemented on platforms for which each run takes up to a few microseconds, which would then require about one day to generate data to train the network. This is indeed the case for superconducting circuit platforms [3], which provide a leading experimental candidate where our ideas should be feasible.

In the future, it will be an interesting challenge to explore how much a network can deal with more noisy measurement data, reducing the need to accumulate statistics. Some efficiency improvements are possible, e.g. one might allow for a larger statistical error (smaller n) if the time points are closely spaced because the network will then effectively try to interpolate smoothly through the noisy observations.

8 Conclusion and outlook

In this work, we explored the power of deep learning in discovering a mapping from the parameters that fully identify a problem Hamiltonian to the parametric gap of the corresponding AQC Hamiltonian. We observe the CONVLSTM network succeeds in predicting the gap on models for which the parameter space size scales linearly with the system size. For such models, our CONVLSTM network is even able to predict the gap for system sizes larger than that it has been trained on and may provide speedup in comparison with the existing exact and approximate algorithms. While during this work we concentrated on the gap prediction in AQC, our study can provide insight for more general many-body dynamics. We conjecture that one of the limiting factors for the learnability of such dynamics applying supervised learning is the way that parameters identifying the model scale with the system size.

Our study supports the promise of CONVLSTM networks in predicting the dynamics of inhomogeneous many-body systems and their potential for extrapolating the dynamics beyond what the neural network is trained on [8, 37, 45]. Our scheme can also be applied in the context of quantum approximation optimization algorithms as it can be viewed as a trotterized version of AQC with parametrized annealing pathway. In this context, it can in particular be integrated with "divide and conquer" methods where one splits problems with hundreds of qubits to subproblems with a few tens of qubits and the dynamics of desired observables of each sub-problem can be learned and predicted by the network [16, 20, 44].

We acknowledge the publication of a paper [21] that recently confirmed our research findings in this work.

Acknowledgments

N.M would like to thank the Erwin Schrödinger International Institute for Mathematics and Physics(ESI), University of Vienna (Austria) for the opportunity to participate in the Thematic Programme "Quantum Simulation - from Theory to Application" where part of this work has been accomplished and for the support given. T. B. is supported by the National Natural Science Foundation of China (62071301); State Council of the People's Republic of China (D1210036A); NSFC Research Fund for International Young Scientists (11850410426); NYU-ECNU Institute of Physics at NYU Shanghai; the Science and Technology Commission of Shanghai Municipality (19XD1423000); the China Science and Technology Exchange Center (NGA-16-001). CNB acknowledges sponsorship from the Yangyang Development Fund, as well as support from a Shanghai talent program and from the Shanghai Municipal Science and Technology Major Project (Grant No. 2019SHZDZX01).

* **Corresponding authors:** derekkorg@gmail.com, tim.byrnes@nyu.edu

Supplemental Material

In this Supplemental Material, we provide a brief review of the convolutional neural network and a particular type of convolutional recurrent neural network called convolutional long short-term memory. We also provide details related to the layout of the network architectures that we applied.

1 Convolutional neural networks

Convolutional neural networks (CNNs) are specific types of networks with a grid-structured topology [18]. They are composed of multiple layers of neurons, including convolutional layers, pooling layers, and some-

times fully connected layers. The convolutional layers apply a series of filters (also known as kernels) to the input image, each of which is responsible to detects specific features. The pooling layers then downsample the output of the convolutional layers by taking the maximum or average of small groups of neurons, reducing the dimensionality of the data and improving computational efficiency. A convolutional neural network (CNN) uses data that contains a few features at each point of the spatial grid. For instance, a 2D-CNN takes an input with shape (w, h, c) where w and h represent the height and width of the spatial structure of the input, and c represents the number of features at each point of the spatial grid. In our case, the features at each site of the grid are identified by the coefficients that describe the spatial structure of our problem Hamiltonians, as shown in Fig. 3 (a) and Fig. 4 (e). .

CNNs have built-in affine invariance so they can recognize patterns that are shifted or tilted in the input. One known benefit of CNNs is that they can be applied for input with varying spacial structures helping to scale up the predictions to larger sizes. Due to this feature, they do not use the standard matrix multiplication but convolution instead. These are the main features that we exploited in this work to extrapolate the prediction of our network to the large sizes. The type of affine invariance and the locality that is required for these architectures to make best of them all exist in our nearest neighbour or the encoded version of the all-to-all connected models.

2 Convolutional Recurrent Neural Networks

In this section, we provide a brief review of the recurrent neural networks (RNNs) and a particular type of that called long short-term memory (LSTM). Then we explain an extended version of that called convolutional LSTM (CONVLSTM) network which we applied in this study.

RNNs are built of a chain of repeating modules of neural networks. Such a network introduces a feedback loop such that the output of the network at the current time depends on the current input (x_t), called the external input, and also on the perceived information from the past, called the hidden input (h_{t-1}) [38]. Note that in our AQC example λ plays the role of time in this architecture. Such a network is able to record the history for – in principle – arbitrary long times, since weights are not time-dependent and therefore the number of trainable parameters does not grow with the time interval. For training such a network, the gradient of the cost function needs to be backpropagated from the output towards the input layer, as in feedforward networks, and also along backward along the time axis.

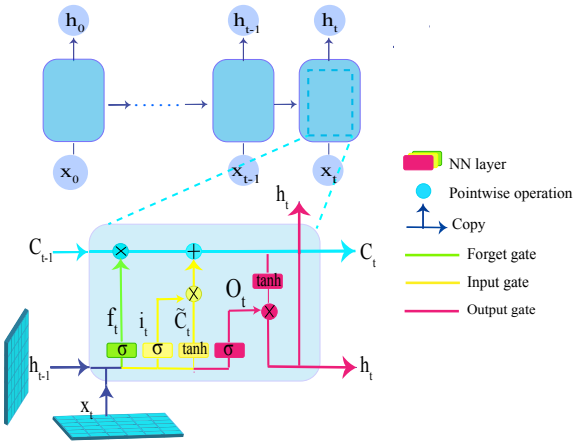


Figure S1: An CONVLSTM network made of a chain of repeating modules, where each module includes three gates. Both the input-to-state and module-to-module transitions have a convolutional structure.

However, RNNs are prone to run into a fundamental problem, the "vanishing/exploding gradient problem", i. e., that the size of the gradient decreases (or sometimes increases) exponentially during backpropagation. In principle, this problem can also occur in traditional feedforward networks, especially if they are deep. However, this effect is typically much stronger for RNNs since the time series can get very long. This seriously limited the trainability of early RNN architectures, which were not capable of capturing long-term dependencies. This problem led to the development of RNNs with cleverly designed gated units (controlling memory access) to avoid the exponential growth or vanishing of the gradient, and therefore permitting to train RNNs that capture both long and short-term dependencies. The first such architecture is called LSTM, developed by Hochreiter and Schmidhuber in the late 90s [22]. As an RNN architecture, standard LSTM is also built of a chain of repeating modules, as is shown in Fig. S1, where the repeating modules have a more complicated structure than in a simple recurrent network. Each module includes three gates, where each gate is composed out of a sigmoid network layer, together with the point-wise multiplication on top of it. Next, we explain step by step how these three gates together control how the memory needs to be accessed. We label weights w and biases b by subscripts according to the name of the corresponding layer.

- Forget gate layer: this gate uses the hidden state h_{t-1} from the previous time step and the external input x_t at a particular time step t (with the bias b_f and the weight w_f) to decide whether to keep the memory, or to discard the information that is of less importance, applying a sigmoid activation.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right). \tag{S1}$$

σ denotes sigmoid function and the dot stands for matrix multiplication. Eventually, the output of the forget gate is multiplied with the module state (C_t).

- Input gate layer: the operation of this gate is a three-step process,
  - first a sigmoid layer decides which data should be stored (very similar to the forget gate)

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right). \tag{S2}$$

  - hidden state and current input also will be passed into the tanh function to push values between -1 and 1 to regulate the network and stored in $\tilde{C}_t$.

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right). \tag{S3}$$

  - the outcome of the two previous steps will be combined via multiplication operation and then this information is added to the module state ($f_t * C_{t-1}$).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t. \tag{S4}$$

  Here the $*$ denotes element-wise multiplication.

- Output gate layer: the operation of this gate which decides the value of the next hidden input can be decomposed into two steps,
  - run a sigmoid layer on the previous hidden state and the current input, which decides what parts of the module state are going to be carried

$$O_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right). \tag{S5}$$

  - passing the module state through tanh to squish the values to be between -1 and 1, and finally multiply it by the output of the sigmoid gate so that we only pass to the next module some selected parts

$$h_t = O_t * \tanh\left(C_t\right). \tag{S6}$$

When data besides temporal structure have also spatial structure the extended version of this architecture called CONVLSTM can be applied. In this case there is convolutional structure in both the input-to-module and module-to-module transitions [49] (shown with blue sheets in Fig. S1) and the internal matrix multiplications are exchanged with the convolution operations.

3 Neural networks layout

In this section we present the layout of the architectures that we applied for the gap prediction task. We have specified and trained all these different architectures with Keras [14], a deep-learning framework written for Python. We systematically tested different values for hyperparameters for different architectures that we applied here. For example for activation function, learning rate, batch size, the kernel size of the convolutional network, etc. After conducting several rounds of experiments, we identified the combination of hyperparameters that produced the best results on our validation set. We report these parameters below.

3.1 Fully connected neural network

In Table. 1, we summarize the details related to the layout of our fully connected neural network (FCNN) for different system sizes and all the models that we explored. The training set size for all models and system sizes is 90,000. Since for the all-to-all connected model, the number of samples required for training explodes with the system size, the network fails in predicting the gap for $M > 7$ using 90,000 samples for training.

We use the rectified linear activation function (ReLU) for all layers except the final one, which utilizes a linear activation function. Additionally, we use the popular “adam" optimizer with a default learning rate of 0.001.

FCNN	All-to-All		NN		LHZ	
System size	# HL	#N/L	# HL	#N/L	# HL	#N/L
M=5	5	500	5	500	3	500
M=6	7	700	5	500	4	500
M=7	Fails		6	700	-	-
M=8	Fails		6	700	-	-
M=9	Fails		6	700	-	-

Table 1: The layout of the FCNN for different system sizes. # HL and # N/L denote the number of hidden layers and the number of neurons per layer, respectively.

3.2 Convolutional long short-term memory

In this section, we present the layout of the 1D and 2D CONVLSTM architectures that we applied in the main text.

1D-CONVLSTM In Table. 2, we present the layout of our 1D-CONVLSTM network applied in Sec. IV. CONVLSTM layers capture the temporal-spatial dependencies of the input. TimeDistributed is a wrapper that applies a layer to every temporal slice of an input. We use this wrapper together with the global max pooling to transfer the input with the temporal-spatial structure to the output with temporal structure.

To push the network to succeed in extrapolation, we did a pre-processing of the input data, as we explain

next. Assume that we want to train the network on sizes $M \in [3, M_T]$ with N training samples for each system size, finally evaluating it on $M \in [3, M_E]$, with $M_E > M_T$ the largest extrapolation we explore. We prepare a four-dimensional array with size $N(M_T - 3) \times N_t \times M_E \times 3$ filled with zeros. Here $N(M_T - 3)$ is the total number of training samples, N_t denotes the number of time steps, and 3 denotes the number of features as we explained in the main text. Now for each sample and time step, we place the string with M_T coupling coefficients randomly within the M_E zeros in this third dimension of the array.

Layers	Filters	Kernel size
CONVLSTM1D	20	3
CONVLSTM1D	40	3
CONVLSTM1D	60	3
CONVLSTM1D	40	3
CONVLSTM1D	20	3
TimeDistributed(Global max pooling)		

Table 2: The layout of the 2D-CONVLSTM

Note that in a CNN by default, a filter starts at the left of the input with the left-hand side of the filter placed on the far left pixels of the input. The filter is then stepped across the input until the right-hand side of the filter is placed on the far right pixels of the image. This means the edge of the input is only exposed to the edge of the filter. However, starting the filter outside the frame of the image can give the pixels on the border of the image more of an opportunity for interacting with the filter and therefore more of an opportunity for features to be detected by the filter, and in turn, an output feature map that has the same shape as the input image. This needs to add the border of input some pixels which is called padding. But to make it clear for the network where exactly the edge of the input starts we add a row of features that are filled with ones for the range of pixels that input starts and ends identifying where a qubit exists and where not.

2D-CONVLSTM In Table. 3, we present the layout of our 2D-CONVLSTM network applied in Sec. V. We prepare the input of this network also with the same pre-processing instruction that we explained for our 1D-CONVLSTM. The only difference is that here we need to insert randomly a square array with size $(M_t - 1, M_t - 1)$ inside the input array of the network with size $(M_e - 1, M_e - 1)$.

4 Training data

As we pointed out in the main text we train the network on small system sizes $M < 8$. To generate data for training, we use qutip [27]. Regarding the time steps for which to calculate the gap, we empirically inspected the gap behavior for different system sizes

Layers	Filters	Kernel size
CONVLSTM2D	20	(2,2)
CONVLSTM2D	40	(2,2)
CONVLSTM2D	60	(2,2)
CONVLSTM2D	40	(2,2)
CONVLSTM2D	20	(2,2)
TimeDistributed(Global max pooling)		

Table 3: The layout of the 2D-CONVLSTM applied in Sec. III C.

under different resolutions. From our observations, we found that a ratio of $\frac{g(\lambda+\Delta\lambda)-g(\lambda)}{g_{Max}} < 0.05$ is a good choice. We acknowledge that this number may not be suitable for all scenarios.

References

[1] Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 20–29, 2003. DOI: 10.1145/780542.780546.

[2] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh. Accelerating quantum approximate optimization algorithm using machine learning. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 686–689. IEEE, 2020. DOI: 10.5555/3408352.3408509.

[3] Tameem Albash and Daniel A Lidar. Demonstration of a scaling advantage for a quantum annealer over simulated annealing. *Physical Review X*, 8(3):031016, 2018. DOI: 10.1103/PhysRevX.8.031016.

[4] Boris Altshuler, Hari Krovi, and Jérémie Roland. Anderson localization makes adiabatic quantum optimization fail. *Proceedings of the National Academy of Sciences*, 107(28):12446–12450, 2010. DOI: 10.1073/pnas.1002116107.

[5] MHS Amin and V Choi. First-order quantum phase transition in adiabatic quantum computation. *Physical Review A*, 80(6):062326, 2009. DOI: 10.1103/PhysRevA.80.062326.

[6] Matthew JS Beach, Anna Golubeva, and Roger G Melko. Machine learning vortices at the kosterlitz-thouless transition. *Physical Review B*, 97(4):045207, 2018. DOI: 10.1103/PhysRevB.97.045207.

[7] Giulio Biroli, Simona Cocco, and Rémi Monasson. Phase transitions and complexity in computer science: an overview of the statistical physics approach to the random satisfiability problem. *Physica A: Statistical Mechanics and its Applications*, 306:381–394, 2002. DOI: 10.1016/S0378-4371(02)00516-2.

[8] Alex Blania, Sandro Herbig, Fabian Dechent, Evert van Nieuwenburg, and Florian Mar-

quardt. Deep learning of spatial densities in inhomogeneous correlated quantum systems. *arXiv preprint arXiv:2211.09050*, 2022. DOI: 10.48550/arXiv.2211.09050.

[9] Troels Arnfred Bojesen. Policy-guided monte carlo: Reinforcement-learning markov chain dynamics. *Physical Review E*, 98(6):063303, 2018. DOI: 10.1103/PhysRevE.98.063303.

[10] Marin Bukov, Alexandre G. R. Day, Dries Sels, Phillip Weinberg, Anatoli Polkovnikov, and Pankaj Mehta. Reinforcement learning in different phases of quantum control. *Phys. Rev. X*, 8:031086, Sep 2018. DOI: 10.1103/PhysRevX.8.031086.

[11] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017. DOI: 10.1126/science.aag2302.

[12] Juan Carrasquilla and Roger G Melko. Machine learning phases of matter. *Nature Physics*, 13(5): 431–434, 2017. DOI: 10.1038/nphys4035.

[13] Juan Carrasquilla and Giacomo Torlai. Neural networks in quantum many-body physics: a hands-on tutorial. *arXiv preprint arXiv:2101.11099*, 2021. DOI: 10.48550/arXiv.2101.11099.

[14] François Chollet et al. Keras. `https://keras.io`, 2015.

[15] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000. DOI: 10.48550/arXiv.quant-ph/0001106.

[16] Keisuke Fujii, Kaoru Mizuta, Hiroshi Ueda, Kosuke Mitarai, Wataru Mizukami, and Yuya O. Nakagawa. Deep variational quantum eigensolver: A divide-and-conquer method for solving a larger problem with smaller size quantum computers. *PRX Quantum*, 3:010346, Mar 2022. DOI: 10.1103/PRXQuantum.3.010346.

[17] Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D. Rodriguez, and J. Ignacio Cirac. Neural-network quantum states, string-bond states, and chiral topological states. *Phys. Rev. X*, 8:011006, Jan 2018. DOI: 10.1103/PhysRevX.8.011006.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

[19] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013. DOI: 10.1109/ICASSP.2013.6638947.

[20] Gian Giacomo Guerreschi. Solving quadratic unconstrained binary optimization with divide-and-conquer and quantum algorithms. *arXiv preprint arXiv:2101.07813*, 2021. DOI: 10.48550/arXiv.2101.07813.

[21] Pratibha Raghupati Hegde, Gianluca Passarelli, Giovanni Cantele, and Procolo Lucignano. Deep learning optimal quantum annealing schedules for random ising models. *arXiv preprint arXiv:2211.15209*, 2022. DOI: 10.48550/arXiv.2211.15209.

[22] S Hochreiter and J Schmidhuber. Long short-term memory neural computation. DOI: 10.1162/neco.1997.9.8.1735.

[23] Hsin-Yuan Huang, Richard Kueng, and John Preskill. Information-theoretic bounds on quantum advantage in machine learning. *Phys. Rev. Lett.*, 126:190505, May 2021. DOI: 10.1103/PhysRevLett.126.190505.

[24] Hsin-Yuan Huang, Richard Kueng, Giacomo Torlai, Victor V Albert, and John Preskill. Provably efficient machine learning for quantum many-body problems. *science*, 2021. DOI: 10.1126/science.abk3333.

[25] Li Huang and Lei Wang. Accelerated monte carlo simulations with restricted boltzmann machines. *Phys. Rev. B*, 95:035105, Jan 2017. DOI: 10.1103/PhysRevB.95.035105.

[26] Marko Žnidarič and Martin Horvat. Exponential complexity of an adiabatic algorithm for an np-complete problem. *Phys. Rev. A*, 73:022329, Feb 2006. DOI: 10.1103/PhysRevA.73.022329.

[27] J Robert Johansson, Paul D Nation, and Franco Nori. Qutip: An open-source python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760–1772, 2012. DOI: 10.48550/arXiv.1110.0573.

[28] Wolfgang Lechner, Philipp Hauke, and Peter Zoller. A quantum annealing architecture with all-to-all connectivity from local interactions. *Science advances*, 1(9):e1500838, 2015. DOI: 10.1126/sciadv.1500838.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Exploring deep reinforcement learning with multi q-learning. *nature*, 521(7553):436–444, 2015. DOI: 10.1038/nature14539.

[30] Daniel A Lidar, Ali T Rezakhani, and Alioscia Hamma. Adiabatic approximation with exponential accuracy for many-body systems and quantum computation. *Journal of Mathematical Physics*, 50(10):102106, 2009. DOI: 10.1063/1.3236685.

[31] Yuichiro Matsuzaki, Hideaki Hakoshima, Kenji Sugisaki, Yuya Seki, and Shiro Kawabata. Direct estimation of the energy gap between the ground state and excited state with quantum annealing. *Japanese Journal of Applied Physics*, 60(SB):SBBI02, 2021. DOI: 10.1088/0305-4470/15/10/028.

[32] Matija Medvidović and Giuseppe Carleo. Classical variational simulation of the quantum approx-

imate optimization algorithm. *npj Quantum Information*, 7(1):1–7, 2021. DOI: 10.1038/s41534-021-00440-z.

[33] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010. DOI: 10.21437/Interspeech.2010-343.

[34] Naeimeh Mohseni, Marek Narozniak, Alexey N Pyrkov, Valentin Ivannikov, Jonathan P Dowling, and Tim Byrnes. Error suppression in adiabatic quantum computing with qubit ensembles. *npj Quantum Information*, 7(1):1–10, 2021. DOI: doi.org/10.1038/s41534-021-00405-2.

[35] Naeimeh Mohseni, Thomas Fösel, Lingzhen Guo, Carlos Navarrete-Benlloch, and Florian Marquardt. Deep learning of quantum many-body dynamics via random driving. *Quantum*, 6:714, 2022. DOI: 10.22331/q-2022-05-17-714.

[36] Naeimeh Mohseni, Peter L McMahon, and Tim Byrnes. Ising machines as hardware solvers of combinatorial optimization problems. *Nature Reviews Physics*, 4(6):363–379, 2022. DOI: 10.1038/s42254-022-00440-8.

[37] Naeimeh Mohseni, Junheng Shi, Tim Byrnes, and Michael Hartmann. Deep learning of many-body observables and quantum information scrambling. *arXiv preprint arXiv:2302.04621*, 2023. DOI: 10.48550/arXiv.2302.04621.

[38] Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, 2015.

[39] Murphy Yuezhen Niu, Andrew M Dai, Li Li, Augustus Odena, Zhengli Zhao, Vadim Smelyanskyi, Hartmut Neven, and Sergio Boixo. Learnability and complexity of quantum samples. *arXiv preprint arXiv:2010.11983*, 2020. DOI: 10.48550/arXiv.2010.11983.

[40] Asier Ozaeta, Wim van Dam, and Peter L McMahon. Expectation values from the single-layer quantum approximate optimization algorithm on ising problems. *Quantum Science and Technology*, 7(4):045036, 2022. DOI: 10.1088/2058-9565/ac9013.

[41] Boris Pittel, Joel Spencer, and Nicholas Wormald. Sudden emergence of a giantk-core in a random graph. *Journal of Combinatorial Theory, Series B*, 67(1):111–151, 1996. DOI: 10.1006/jctb.1996.0036.

[42] Jérémie Roland and Nicolas J Cerf. Quantum search by local adiabatic evolution. *Physical Review A*, 65(4):042308, 2002. DOI: 10.1103/PhysRevA.65.042308.

[43] A. E. Russo, K. M. Rudinger, B. C. A. Morrison, and A. D. Baczewski. Evaluating energy differences on a quantum computer with robust phase estimation. *Phys. Rev. Lett.*, 126:210501, May 2021. DOI: 10.1103/PhysRevLett.126.210501.

[44] Zain H Saleem, Teague Tomesh, Michael A Perlin, Pranav Gokhale, and Martin Suchara. Quantum divide and conquer for combinatorial optimization and distributed computing. *arXiv preprint arXiv:2107.07532*, 2021. DOI: 10.48550/arXiv.2107.07532.

[45] N. Saraceni, S. Cantori, and S. Pilati. Scalable neural networks for the efficient learning of disordered quantum systems. *Phys. Rev. E*, 102:033301, Sep 2020. DOI: 10.1103/PhysRevE.102.033301.

[46] Gernot Schaller. Adiabatic preparation without quantum phase transitions. *Phys. Rev. A*, 78:032328, Sep 2008. DOI: 10.1103/PhysRevA.78.032328.

[47] Markus Schmitt and Markus Heyl. Quantum many-body dynamics in two dimensions with artificial neural networks. *Phys. Rev. Lett.*, 125:100503, Sep 2020. DOI: 10.1103/PhysRevLett.125.100503.

[48] Ralf Schützhold. Dynamical quantum phase transitions. *Journal of Low Temperature Physics*, 153 (5-6):228–243, 2008. DOI: 10.1007/s10909-008-9831-5.

[49] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf.

[50] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Neural-network quantum state tomography. *Nature Physics*, 14(5):447–450, 2018. DOI: 10.1038/s41567-018-0048-5.

[51] Evert PL Van Nieuwenburg, Ye-Hua Liu, and Sebastian D Huber. Learning phase transitions by confusion. *Nature Physics*, 13(5):435–439, 2017. DOI: 10.1038/nphys4037.

[52] Filippo Vicentini. Machine learning toolbox for quantum many body physics. *Nature Reviews Physics*, 3(3):156–156, 2021. DOI: 10.1038/s42254-021-00285-7.

[53] Lei Wang. Discovering phase transitions with unsupervised learning. *Phys. Rev. B*, 94:195105, Nov 2016. DOI: 10.1103/PhysRevB.94.195105.

[54] Sebastian J Wetzel. Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders. *Physical Re-*

view E, 96(2):022140, 2017. DOI: 10.1103/PhysRevE.96.022140.

[55] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf.

[56] A. P. Young, S. Knysh, and V. N. Smelyanskiy. Size dependence of the minimum excitation gap in the quantum adiabatic algorithm. *Phys. Rev. Lett.*, 101:170503, Oct 2008. DOI: 10.1103/PhysRevLett.101.170503.