Quantum Neural Networks

A Hamiltonian Complexity Approach

The goal of this work is to build a quantum neural network, train it on some actual data and then test its performance on a classification task. More precisely, the idea of this submission is to use Microsoft's quantum simulator, LIQUi|⟩*, to construct local Hamiltonians that can encode trained classifier functions in their ground states, and which can be probed using simulated annealing. I propose a training scheme which is completely closed-off to the environment and which does not depend on external measurements, avoiding unnecessary decoherence during the annealing procedure. For a network of size n, the trained network can be stored as a list of $\mathcal{O}(n)$ coupling strengths. Using* LIQUi|⟩-Learn*, my quantum neural network library, I address the question of which interactions are most suitable for the classification task, developing a number of qubit-saving optimizations for the simultation. Furthermore, a small neural network to classify colors into red vs. blue is trained and tested.*

Entry to the first Microsoft Quantum Challenge 2016 by **Johannes Bausch** | jkrb2@cam.ac.uk, DAMTP, University of Cambridge, UK | April 21, 2016.

Git repository at `https://liquidlearn-guest@bitbucket.org/rumschuettel/liquidlearn.git` (password `quantum2016`).

1 Introduction

The authors of [1] summarize past efforts to extend the notion of classical neural networks to the quantum world. This task is not straightforward: measurements have to be included to introduce nonlinearity, which interferes with the coherence of the model, or postselection is used to collapse a superposition of networks. A system of interacting quantum dots is suggested, capable of simulating network dynamics. However, experimentally, it is unclear how this is easier than building an analogue quantum computer.

Adiabatic quantum annealing [2] is the only area of experimental quantum computing claiming significant qubit counts. By tuning the couplings between qubits, one hopes to encode a computation into the ground state of the Hamiltonian. Cooling and measuring the system then reveals the computation output. The interactions, however, are typically limited to Ising-type couplings, and the physical interaction graph is fixed: logical interactions have to be crafted onto this underlying structure. Furthermore, the coupling strengths have to be hand-tuned to represent a problem of interest, or calculated on classical hardware.

Despite these limitations, specialized hardware such as the *D-Wave*—with their underlying *Chimera* interaction graph—has one significant advantage over a full-fledged analogue programmable quantum computer: we can build them today.

I want to introduce a new notion of quantum neural networks, based on this idea of quantum annealing on a fixed underlying graph, but addressed from a Hamiltonian complexity perspective. Furthermore, I propose a training algorithm which could be completely incorporated into such an annealing device, avoiding the need for any classical optimization or repeated measurement to introduce non-linearity, and reducing unnecessary decoherence. I construct a few such neural networks with LIQUi|⟩-Learn—an F# library based on LIQUi|⟩ which can be used to build, train and test quantum classifiers—demonstrating that the quantum training algorithm works as intended. I finally address a few related questions, such as which interactions are most suitable for the network, and put it to the "real-world" classification task of distinguishing red and blue colors.

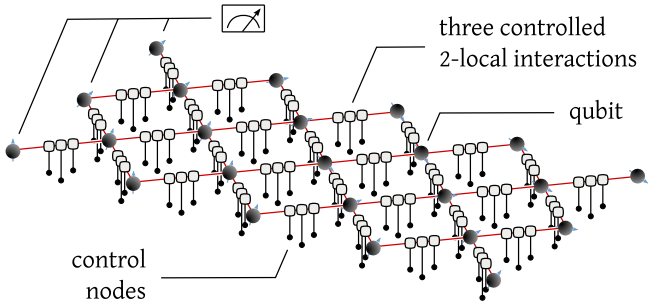


Figure 1: Schematic of a quantum neural network with controlled local interactions between adjacent qubits. The control nodes are used for training the network, as well as fixing the coupling strengths for classification tasks. Any number of qubits can act as output, which is constrained to the training data during training and measured jointly for classification.

The field of Hamiltonian complexity has seen a lot of interesting contributions over the past decade. Based on Feynman's idea of embedding computation into a local Hamiltonian [3] and starting with Kitaev's original proof of QMA-hardness of the ground state energy problem of 5-local Hamiltonians, the physical relevance of hardness results has been improved successively [4–7]. More recent results include using history state and tiling Hamiltonian embeddings to prove undecidability of the spectral gap [8] and size-driven phase transitions [9], both of which embed highly complex computation into relatively simple local couplings. This suggests that local Hamiltonians are well-suited for computational tasks; but which problems can we answer, and what is the overhead? Hamiltonians constructed from local interactions have been classified into the complexity classes P, StoqMA, NP and QMA in [10, 11], and it is a well-known fact that almost all Hamiltonian interactions are uni-

versal in the sense that they can be used to approximate any 2-qubit unitary operation [12].

There are a series of implications which we are able to draw from these findings. First of all, we know that local Hamiltonians can be used to perform both classical and quantum computation. There are two fundamentally different embedding constructions available.

A so-called *History State* construction is a local Hamiltonian allowing state transitions, based on Feynman's idea of embedding quantum computation into the ground state of the Hamiltonian. Given some Hilbert space $\mathcal{H}$ and two states $|a\rangle, |b\rangle \in \mathcal{H}$, we can encode a transition $a \leftrightarrow b$ by a local interaction of the form $(|a\rangle - |b\rangle)(\langle a| - \langle b|)$. The ground state of such a Hamiltonian will be a superposition of all states connected by transition rules. The most extensive such construction to date can be found in [13], and the spectral gap of such History State Hamiltonians is known to close as $1/\text{poly}$ in the system size.

Tiling problems, on the other hand, regard a set of square, edge-colored tiles, and ask whether it is possibe to tile the infinite plane with matching colors, where we are not allowed to rotate the tiles. This problem is known to be undecidable, and is the foundation of various undecidability or uncomputability results on spectral properties of local Hamiltonians. Computation in these constructions is done similar to cellular automata, and in particular *not* a superposition in-place: e.g. in [9], we embed a Turing Machine into a local Hamiltonian and use one spatial direction of the system for its evolution.

One fundamental difference to History State constructions is that with a Tiling Hamiltonian, we get a constant spectral gap. The problem with such Tiling Hamiltonians, however, is that it is not clear how to perform quantum computation with it, and a no-go theorem for embedding unitary gates into a Tiling Hamiltonian [14] as well as the implications of such a gap amplification (we could solve the *Quantum PCP Conjecture* if we could amplify to a constant gap, cf. [15]) sound discouraging—but also challenging, as there is no known result forbidding this possibility.

I believe that there are a lot of open problems in this field of research. In particular, both the set of local interactions for History State constructions—sums of Hadamard-like operations—and Tiling interactions—diagonal projectors—only span a low-dimensional subspace over all possible interactions, and while interactions in general have been classified, it is not clear whether there exists a more optimal one than History States, or not. A lot of research has been done to improve unitary gate usage in various contexts [16, 17], but we know little to nothing about reducing the overhead of embedding computation into the ground space of a Hermitian operator. For the hardness results, the constructed embeddings follow the usual *Solovay-Kitaev* Karp reduction from a language L—complete for e.g. NP or BQEXP—which results in a poly computation overhead, but the constructions themself usually perform computation in a very localized fashion, meaning that there is some notion of heads, similar to a Turing Machine, while most of the space of the underlying Hilbert space remains a passive computation tape. This is not efficient: if we were able to more directly embed quantum circuits into a local Hamiltonian, we could drastically reduce the number of qubits needed to perform the calculation.

Neural networks are an alternative computational model, which has gained significant popularity over the past few years. Due to the underlying graph's high connectivity, computation is naturally performed in a highly parallel manner. This does not necessarily work for quantum simulators, where low connectivity and geometric locality is preferred. We therefore want to model a quantum neural network as a geometrically local hypergraph, where the edges represent interactions from a fixed set and hope that quantum effects—entanglement and long-range correlations—will step in to replace the lack of non-local connections in the interaction graph. Training will be done by varying the coupling strengths between adjacent qubits.

In this work, I want to focus on the problem of *classification*: starting from a dataset consisting of YES and NO instances, we train the model to accept everything that is a YES instance with high probability, and analogously reject for NO—corresponding to a low and high energy ground state of the simulated Hamiltonian, respectively.

To start out, take a hypergraph $G = (\mathrm{V}, \mathrm{E})$ with a set of vertices V labelling qubits, and hyperedges E encoding interactions between them. Assign the qubit Hilbert space $\mathbb{C}^2$ to each vertex, such that the overall Hilbert space becomes $\mathcal{H} = (\mathbb{C}^2)^{\otimes|\mathrm{V}|}$. For a local operator h acting only on a subset of vertices $\mathrm{U} \subseteq \mathrm{V}$ and as zero everywhere else, we write $\mathrm{h}^{(\mathrm{U})}$. For every hyperedge, we are given a list of local interactions $(\mathrm{S}_e)_{e\in\mathrm{E}}$. The overall Hamiltonian on $\mathcal{H}$ can then be written as

$$\mathrm{H} = \sum_{e\in\mathrm{E}} \sum_{\mathrm{h}\in\mathrm{S}_e} a_{\mathrm{h},e}\mathrm{h}^{(e)}, \tag{1}$$

where the $a_{\mathrm{h},e}$ are the coupling constants to train. We label some of the vertices of the hypergraph as ouptut vertices $\mathrm{V}_o \subseteq \mathrm{V}$, and call the complement V_o^c *hidden*: $\mathrm{tr}_h(\mathrm{H})$ then denotes the Hamiltonian H, where we traced out this hidden part of $\mathcal{H}$. This setup allows us to define the following optimization problem.

Definition 1 (Classification Problem) *Let* L *be a decision problem with instances of binary size n. Given a hypergraph G with n output vertices, and a list of local interactions* $(\mathrm{S}_e)_{e\in\mathrm{E}}$*, find the optimal coupling constants in equation* (1) *such that the two discrete distributions* $\mathcal{D}_{\mathrm{YES}} := \{\langle l|\,\mathrm{tr}_h(\mathrm{H})\,|l\rangle : l \in \mathrm{L}_{YES}\}$*, and analogously* $\mathcal{D}_{\mathrm{NO}}$*, are maximally distinguishable.*

Given $\mathcal{G}_{\mathrm{YES/NO}}$, we can calculate *soundness*, defined as the fraction of $l \in \mathcal{D}_{\mathrm{YES}}$ for which $\mathbb{P}(l \in \mathcal{G}_{\mathrm{YES}}) > \mathbb{P}(l \in \mathcal{G}_{\mathrm{NO}})$, as well as *completeness*, defined by flipping the roles of YES and NO. We further require the mean of $\mathcal{G}_{\mathrm{YES}}$ to lie below the mean of $\mathcal{G}_{\mathrm{NO}}$, in analogy to the hardness results summarized in the introduction. The definition deliberately leaves room for interpretation, i.e. what precisely *maximally distinguishable* means. For our purposes, we distinguish between the two distributions using a linear logistic regression.

2 Simulation, Training and Testing

Introduction. We want to train a neural network, which we model as a connected hypergraph $G = (V, E)$, where the hyperedges E connect one or more vertices, each of which labels a qubit. The size of the edge corresponds to the maximum locality of the interaction. As quantum computation is intrinsically reversible, we regard the subset of vertices $V_o \subseteq V$ simultaneously as in- and output vertices.

I will give a walk-through on how to create a simple quantum classifier using LIQ$Ui|\rangle$-Learn. As an example, let us create a path graph of three vertices, the middle of which is hidden.

```
let edges = [ O"1" --- V"h" ; V"h" --- O"2" ]
let graph = new Hypergraph(edges)
```

With `O"1"` and `O"2"`, we have two input slots to perform classification on. We first define the dataset to train via

```
let trainingSet = {
  Yes = List.map Data.FromString ["00"; "11"]
  No  = List.map Data.FromString ["01"; "10"]
}
```

We then decide between the type of interactions to use—e.g. only projectors, a complete Pauli basis or some random gate set—and train the dataset on our graph.

```
let model = (
  new SimpleControlledTrainer(graph, Sets.Pauli())
).Train(trainingSet)
```

In this example, `Sets.Pauli()` is a factory type returning a list of interactions that apply to every edge: for our 2-local graph, every edge gets assigned the possible interaction set $\{\sigma_i \otimes \sigma_j : 0 \leq i, j \leq 3\}$.

Under the hood, the trainer's constructor now augments every edge with a list of control vertices, one for each interaction. This new "controlled" graph then has the 32 3-local hyperedges

```
[
  O"1" --- V"h" --- C(id="1"; interactions=["00"])
  O"1" --- V"h" --- C(id="2"; interactions=["01"])
  O"1" --- V"h" --- C(id="3"; interactions=["02"])
  //...
  V"h" --- O"2" --- C(id="32"; interactions=["33"])
]
```

where `C` labels a special control vertex with an id and a list of interactions[1].

As a next step, the trainer creates a controlled gate for every edge, which allows it to vary the corresponding interaction's coupling strength in equation (1). As an example, consider the interaction $\sigma_1 \otimes \sigma_3$ between the first output vertex `O"1"` and the hidden vertex `V"h"`. Sorting the control qubit Hilbert space to the front, i.e. working on $\mathcal{H}_{\text{control}} \otimes (\mathcal{H}_1 \otimes \mathcal{H}_h)$,

[1] Technically, each interaction label also carries a list of the vertices where it acts nontrivially, which is used for the vertex optimization later. This is left out here for brevity.

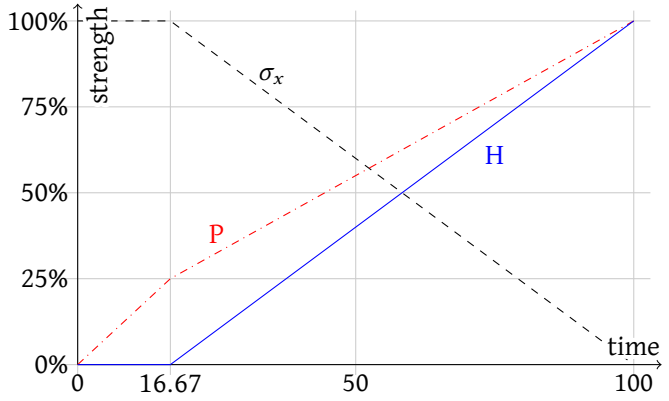


Figure 2: Annealing schedule for the LIQ$Ui|\rangle$ simulation. The dashed line is the σ_x interaction, the dash-dotted red line is the output penalty terms P and the solid blue line is the local coupling H which we want to train.

we create a controlled `Liquid.Gate` with unitary matrix

$$\exp\left(it\begin{pmatrix}0 & 0\\ 0 & \sigma_1\otimes\sigma_3\end{pmatrix}\right) = \begin{pmatrix}\mathbb{1}_4 & 0\\ 0 & \mathrm{U}(t)\end{pmatrix},$$

where $\mathrm{U}(t) = \exp(it\sigma_1 \otimes \sigma_3)$ is the time-dependent exponentiated interaction used for annealing. If the control qubit is now mostly in $|0\rangle$, the coupling is weak, and if it is mostly in $|1\rangle$, the interaction is strong.

In order to train the model, LIQ$Ui|\rangle$ allocates a `Ket` with one qubit per unique graph vertex, using `Liquid.Spin`, and we run the `Spin.Test` annihilation, using the qubit interactions defined above as custom `SpinTerms`. Internally, we run the simulator twice: once for the YES instances and once for NO. In the YES run, we assign an energy penalty to the output qubits in any state specified by the NO training dataset (here `"01"` and `"10"`), utilizing a new gate based on the exponentiated projector

$$\exp(it\,|01\rangle\langle 01| + it\,|10\rangle\langle 10|)_{1,2} \otimes (\mathbb{1}_4)_{\text{h,control}}, \qquad (2)$$

and vice versa on the YES dataset for the second run. Together with the controlled interactions we perform annealing, following a schedule as shown in fig. 2.

After each run, we perform a joint measurement on the control qubits. Depending on whether a control qubit tends to a state $|0\rangle$—meaning the interaction between the controlled qubits is the zero interaction $\exp(it0) = \mathbb{1}_4$—or to $|1\rangle$—meaning the interaction is $\mathrm{U}(t)$—we know whether this particular interaction either frustrates due to the penalty, or is not in conflict, i.e. allows a global ground state with little penalty overlap. We collect and normalize all coupling strengths $\{a_{h,i}\}$ and store them in the model.

In order to test the model on the training data, we call

```
model.Test trainingSet // or some other test set
```

The classifier creates a `Liquid.Spin` Hamiltonian of our original graph, but with the couplings in equation (1) hard-coded into the `SpinTerms`. For every data point $l \in L$ given, we now calculate $\langle l|\,\mathrm{tr}_h\,\mathrm{H}\,|l\rangle$, using `Spin.EnergyExpectation`. The result is a list of energies and standard deviations for each test data point.

Optimizations. Since we have to work within the limitations imposed by hard- and software[2], we cannot simulate a graph with 32 hyperedges and 35 qubits. So instead of having one controlled edge per interaction, we group interactions on similar vertices together into fewer, bigger multiedges: instead of training $U(t)$ on a controlled interaction with block-diagonal matrix $\mathrm{diag}(\mathbb{1}_4, U)$, we instead train a whole series of interactions simultaneously with a matrix $\mathrm{diag}(\mathbb{1}_4, U_1, \ldots, U_n)$, utilizing $\mathcal{O}(\log n)$ control qubits. If we bin a multiedge with `0"1"` and `V"h"` together with one including `0"2"`, however, we have to increase the underlying interaction size to 3-local instead of 2-local. This is done automatically, and the maximum number of controlled and controlling qubits can be given as parameter to `SimpleControlledTrainer`. By default, we group at most 3 qubits together, and use at most 4 qubits as control: this allows us to train a maximum of 15 interactions, shared between 3 qubits, reducing the number of qubits necessary to train the example at hand from 35 to 13 (!). How this optimization performs on other graphs can be seen in table 1.

This optimization works in a similar fashion to how LIQ*Ui*|⟩ clusters gates together, but with the additional knowledge of knowing how to merge single-control gates into multi-control gates.

A Real World Classification. We would like to assess our setup with a more relevant, "real-world" problem. Unfortunately, even the smallest examples from well-known machine learning datasets require way too many qubits as input to run on the quantum simulator, so we have to construct a somewhat artificial dataset which works within the hardware and software limitations at hand.

We consider the task of classifying colors into the classes *red* and *blue*. Our training dataset will consist of a list of 9 bit, little-endian color codes in RGB order, e.g. 001 100 111, which would represent a light blue tone (little red, half green, full blue). This is an interesting, non-trivial task, as there are a lot of edge cases, as well as some colors which are neither considered red or blue. So let us see what our quantum classifier thinks.

The graph we will train this example on can be seen in fig. 3. We use the `Projectors` interaction set, i.e. 2-local projectors on computational basis states. Ten red and blue shades are chosen randomly to train the network, and we test it on the full possible 512 color spectrum available with 9 bits. The output is plotted in fig. 4, sorted from most red to most blue. The error bar shows the standard deviation as returned by the test measurement utilizing LIQ*Ui*|⟩'s `Spin.EnergyExpectation`.

We see that the classification task was indeed able to distinguish between red and blue colors. It is interesting to see that the network correctly identifies turquoise as blue and pink or orange tones as red. It also leaves most of the other colors—green and purple—at an intermediate energy level, which is particularly visible when looking at the hue-sorted spectrum in fig. 4.

[2] 23 qubits for the public version of LIQ*Ui*|⟩.

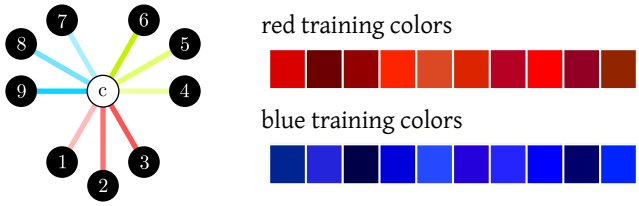


Figure 3: Training graph for 9 bit color classification problem. Each component encodes its color in 3 bits. We need 22 qubits to train the `Projectors` interactions on an optimized graph, where we limit the gate size to 4 contol qubits and 4 controlled qubits. The right hand side shows ten randomly chosen red and blue shades used for training the quantum classifier.

Benchmarking Interactions. As outlined in the introduction, one interesting research questions is which type of interaction is most suited for a quantum classifying Hamiltonian, in the sense that it requires few qubits to train while giving a good separation between the YES and NO instances.

In this work, we regard five sets of interactions:

```
Sets.Projectors() // projectors on comp. basis
Sets.Paulis() // Pauli product matrices
Sets.Ising() // Ising-type interactions
Sets.Heisenberg() // Heisenberg-type interactions
Sets.Random() // set of 7 sparse random Hermitians
```

Pauli matrices give a full basis for Hermitian matrices, as we can expand any Hermitian as a linear combination of Pauli product matrices. Ising-type interactions are relevant, as they play a fundamental role in many real-world physical systems and are the interactions used in a *D-Wave*, so it is interesting to see how they compare to others. Heisenberg interactions have been proven to be QMA-complete in [10], and the random matrices are picked as a benchmark for the others.

In order to quantify how these interaction sets fare, we try to train all possible datasets[3] on small quantum networks, cf. table 1. The intuition is that if an interaction type is able to correctly train a lot of combinations on a small scale, assembling a bigger graph from these interaction types results in a more space-efficient network. For the network with six inputs, we only pick a few random training sets, as the number of all possible sets scales factorially (even when excluding symmetries) and due to our limited computing power. Table 1 summarizes the benchmark results.

À priori, it is not clear which interactions will perform best in a specific scenario. The underlying graph plays a big role, as well as overfitting, which the Pauli matrices seem to suffer from: their marginally better energy split as compared to other interaction types might be a bad tradeoff in terms of runtime. Surprisingly, the randomly chosen sparse matrices fare better than expected, and Heisenberg and Ising interactions are usually outperformed.

[3] For a network with two inputs, we have the possible data points 00, 01, 10 and 11. We build all data sets comprising two YES and two NO instances, but disregard any equivalent sets under symmetries, e.g. redefining $0 \leftrightarrow 1$, or permuting the order by the underlying graph's symmetries—this is implemented in the `AllDataSets` function.

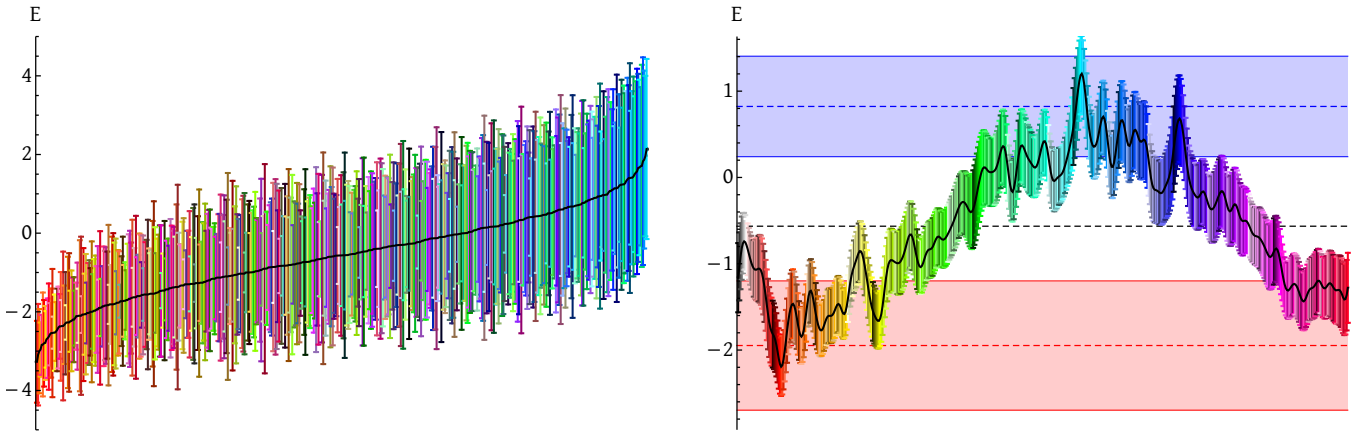


Figure 4: Testing outcome of trained QNN from fig. 3. The left figure shows the expected energy and uncertainty for all possible 512 9-bit colors, sorted by measured energy from `Spin.EnergyExpectation`. The right figure shows a moving average over the energy with window size 10, sorted by the color's actual hue. Blue and turquoise tones have the highest energy and can be clearly assigned the class "blue": they lie within one sigma of the average energy when tested on the training dataset, which is shown as blue stripe around $E \approx 0.8$. Violet and green lie neither close to the blue or red region, so could be identified as "neither". Note that orange, yellow and warm reds have the lowest energy and lie clearly within the identified "red" region around $E \approx -2$. This is to be expected: the QNN has no training data to classify these colors, so it extrapolates—i.e. yellow being green and red without blue, so the presence of red *and* absence of blue amplify to this outcome.

3 Conclusion and Outlook

In this work, I have demonstrated the feasibility of a quantum neural network, implemented as a ground state energy problem of a local Hamiltonian. The LIQ*Ui*|⟩-Learn toolset has been used to simulate an array of such networks: a color classifier has been trained and tested, and various interaction types have been benchmarked against each other.

It would be interesting to see this line of research performed on a greater scale. The networks I could test on machines accessible to me—a few standard i7 8-core desktop computers at the CMS, and a few hours on an 8 Xeon CPU cluster at the Cambridge CS department—were very limited: if ≈ 30 qubits were accessible, I believe a lot more interesting interaction geometries could be assessed, and with greater computing power a more exhaustive search over optimal interaction types could be performed.

As a PhD student in quantum information theory I would also like to see this research put on a more rigorous theoretical foundation. While the training algorithm has been shown to perform well on this small scale, nothing is known about its convergence behaviour, its complexity, the gap behavior, or if there is a more efficient algorithm which e.g. does not rely on potentially high-locality projectors in equation (2). It would, for example, be interesting to see if we can modify the algorithm to work serially: instead of preparing all training data in superposition, the model could be fed each data point subsequently, annihilated, and the separate outcomes combined at the end. This way, only product states would need to be prepared as input, which is much more feasible experimentally.

Furthermore, an interesting question is the capacity of such networks. For a network of n nodes, we only need to store $\mathcal{O}(n)$ coupling constants, which is much lower than the $\Omega(n^2)$ needed entries in transition matrices for many classical neural nets. Does this restriction of only having local connectivity restrict the amount of information we can store in a quantum network, or do entanglement and long-range correlations make up for it?

Of course this idea is a great candidate to be run on an actual quantum annihilating device, such as the D-Wave. Not only would this allow us to create neural networks on a much larger scale, but it would also be one step closer towards an implementation as specialized classifier hardware, given a controlled interaction can be implemented efficiently in an experimental setup. If this is possible, a chip containing a generic network as in fig. 1 could be put in training mode to learn a dataset, and then prompted afterwards with test data to get a quick answer. Multiple chips of this type could be used to build a larger, more capable network, that can handle a lot of classification tasks in parallel.

I would like to thank the organizers of the Microsoft Quantum Challenge for their effort, and for making LIQ*Ui*|⟩ openly accessible. It provided me with the perfect excuse to learn F# and to dive deeper into neural networks and machine learning, a topic I have been fascinated by ever since I started programming over 10 years ago. I will be finishing my PhD in Cambridge next summer, and I am positive that I want to continue working on this interdisciplinary research around Hamiltonian complexity, a subject which I find gains ever more relevance—and applicability—with every new problem that is addressed.

In order to run the code described in this submission, clone the LIQ*Ui*|⟩-Learn repository (see link below abstract) with password `quantum2016`. Branch `master` is for Windows and compiles on VisualStudio 2015, .NET 4.6.1, branch `Linux` was used to run all simulations outlined above, and compiles on mono 4.2.3. Read `readme.md` for instructions on how to set up LIQ*Ui*|⟩-Learn, or contact me (jkrb2@cam.ac.uk) if there are any problems. I will release the code publicly at some point, but for now the repository is only accessible to Microsoft as part of this submission.

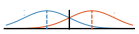
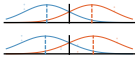
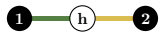
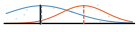
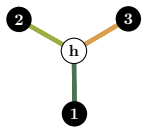
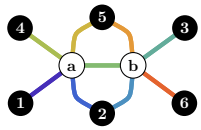
Graph	Interactions	Data	Simulation			Simulation Result		
			Qubits	Qubits (opt)	Time (avg, s)	Spectrum	Energy Δ	Fidelity (%)
	Pauli	3	18	7	360		2.0	88
	Projectors	3	6	5	11		2.0	100
	Random	3	9	5	11		0.64	65
	Heisenberg	3	5	4	1		1.0	75
	Ising	3	5	4	1		5.58	75
	Pauli	3	35	13	1.0×10^3		1.72	75
	Projectors	3	11	7	260		0.97	88
	Random	3	17	7	310		1.10	75
	Heisenberg	3	9	6	57		0.0	50
	Ising	3	8	6	57		0.0	50
	Pauli	35	52	19	15×10^3		0.68	56
	Projectors	35	16	8	690		0.84	75
	Random	35	25	9	8.7×10^3		1.19	72
	Heisenberg	35	13	8	840		0.0	50
	Ising	35	11	7	120		0.0	50
	*Projectors*	10	44	20	35×10^3		0.40	63
	Random	6	71	22	n/a*		0.48	57
	Heisenberg	10	35	16	11×10^3		0.0†	54
	Ising	10	25	14	4.4×10^3		0.0†	52

Table 1: Interaction benchmark results. The best-performing interaction set for each graph is highlighted in italics. It is interesting to see that the best interaction type varies greatly with the underlying graph. To quantify the fidelity, we calculate the percentage of instances that can be classified correctly with the given graph and interaction type, where we sum over all used training sets. For the most basic two-vertices graph, projectors come out on top—projectors give a sharp splitting and require few qubits to train. Ising and Heisenberg interactions struggle with instances with symmetry, e.g. yes as $\{|01\rangle, |10\rangle\}$ and no as $\{|00\rangle, |11\rangle\}$, which is to be expected. For three vertices on a path graph, projectors still fare best. Interestingly, random matrices and Paulis are both runner-ups despite the former requiring far fewer qubits to train. Neither Heisenberg and Ising type interactions can be trained on this type of graph. The four vertices graph shows that random matrices overtake Paulis, while saving a lot of qubits. They do almost as well as projectors while providing a larger gap. The last graph was too big to run with the full set of Pauli matrices, but again we see that projectors and randomly chosen matrices do much better than Heisenberg or Ising-type interactions. The ten randomly chosen datasets were not necessarily the same for all interaction types.

† Small nonzero splitting, the resulting fidelity > 50 might therefore be a precision artifact in this case.

⋆ Due to lack of computation time, this test had to be interrupted after six test sets.

References

[1] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. "The quest for a Quantum Neural Network". In: *Quantum Information Processing* 13.11 (2014), pp. 2567–2586. arXiv: 1408.7005.

[2] Catherine C. McGeoch. "Adiabatic Quantum Computation and Quantum Annealing: Theory and Practice". In: *Synthesis Lectures on Quantum Computing* 5.2 (2014), pp. 1–93.

[3] Richard P. Feynman. "Quantum mechanical computers". In: *Foundations of Physics* 16.6 (1986), pp. 507–531.

[4] Julia Kempe, Alexei Yu. Kitaev, and Oded Regev. "The Complexity of the Local Hamiltonian Problem". In: *SIAM Journal on Computing* 35.5 (2006), pp. 1070–1097. arXiv: 0406180 [quant-ph].

[5] Roberto Oliveira and Barbara M. Terhal. "The complexity of quantum spin systems on a two-dimensional square lattice". In: *Quantum Information & Computation* (2005), pp. 1–23. arXiv: 0504050 [quant-ph].

[6] Dorit Aharonov, Daniel Gottesman, Sandy Irani, and Julia Kempe. "The power of quantum systems on a line". In: *Communications in Mathematical Physics* 287.1 (2009), pp. 41–65. arXiv: 0705.4077.

[7] Daniel Gottesman and Sandy Irani. "The Quantum and Classical Complexity of Translationally Invariant Tiling and Hamiltonian Problems". In: *Theory of Computing* 9.1 (2013), pp. 31–116. arXiv: 0905.2419.

[8] Toby S. Cubitt, David Perez-Garcia, and Michael M. Wolf. "Undecidability of the spectral gap". In: *Nature* 528.7581 (2015), pp. 207–211. arXiv: 1502.04573.

[9] Johannes Bausch, Toby S. Cubitt, Angelo Lucia, David Perez-Garcia, and Michael M. Wolf. "Size-Driven Quantum Phase Transitions". In: (2015). arXiv: 1512.05687.

[10] Toby S. Cubitt and Ashley Montanaro. "Complexity classification of local Hamiltonian problems". In: (2013), p. 51. arXiv: 1311.3161.

[11] Stephen Piddock and Ashley Montanaro. "The complexity of antiferromagnetic interactions and 2D lattices". In: (2015), p. 35. arXiv: 1506.04014.

[12] Andrew M. Childs, Debbie Leung, Laura Mančinska, and Maris Ozols. "Characterization of universal two-qubit Hamiltonians". In: (2010). arXiv: 1004.1645.

[13] Johannes Bausch, Toby S. Cubitt, and Maris Ozols. "The Complexity of Translationally-Invariant Spin Chains with Low Local Dimension". In: *QIP 2016*. To appear on the arXiv end of April, 2016, p. 52.

[14] Toby S. Cubitt. *Part III: Advanced Quantum Information Theory*. 2014.

[15] Thomas Vidick and John Watrous. "Quantum Proofs". In: *Foundations and Trends in Theoretical Computer Science* 11.1-2 (2016), pp. 1–215.

[16] Alfred V. Aho and Krysta M. Svore. "Compiling Quantum Circuits using the Palindrome Transform". In: *Proceedings of the ERATO Conference on Quantum Information Sciences (EQIS)*. 2003. arXiv: 0311008 [quant-ph].

[17] Nabila Abdessaied, Mathias Soeken, and Rolf Drechsler. "Quantum Circuit Optimization by Hadamard Gate Reduction". In: 2014, pp. 149–162.