Hamiltonian Simulation using Quantum Autoencoders

Burak Mete, Irene Lopez Gutierrez, Christian Mendl

April 16, 2021

1 Introduction

Quantum machine learning is a very broad area that comprises two different main fields, depending on the algorithm type. First, the use of existing classical machine learning algorithms that are aiming to explore quantum systems, and secondly the quantum algorithms, which makes use of quantum properties like superposition and entanglement to explore either classical or quantum data. Exploiting the benefits of quantum computing might be used to perform a routine that solves an existing classical problem, with the motivation of achieving a considerable speedup, or a better accuracy in the learning task compared to classical methods. One could also potentially use the quantum algorithms, to explore quantum systems. Figure 1, also summarizes those different utilization of QML approaches[1].

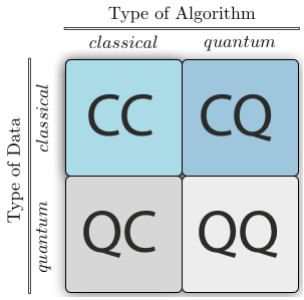

Figure 1: Different types of QML [1]

In this project we are interested in the latter, namely quantum circuits that encode machine learning algorithms for classical data, such as discriminative or generative neural networks. That is, the main goal is to obtain a

circuit that can be run on a quantum hardware. Ideally, a speedup would be provable compared to the classical counterpart (at least for the noiseless case). Since the hardware is not mature enough yet and for some heuristic algorithms providing provable theoretic speedups is hard, we will at least require to have an intuition of why a speedup would be expected.

With this work, we are aiming to generate an efficient solution for a subset of Hamiltonian Simulation problems, with making use of Quantum Autoencoders, and the latent space representations of the quantum system.

In the Section 2, we will try to give an overview on the Quantum Autoencoders, and present you our implementation of Romero, Olson and Aspuru-Guzik's paper [2], using the classical MNIST [3] dataset, and we will share our insights and results. We will continue with giving a background on the Hamiltonian Simulation problem, and the complexities of the state of art techniques in solving the problem. In Section 3, we are presenting the details of our model, which creates a pipeline for simulating the time evolution of a system with a given Hamiltonian. That will be followed by the Section 4, where we will give the results of our model and give a benchmark analysis for the other solutions.

For the implementation, we have used Pennylane [2] as the backend of the quantum circuit simulator, and PyTorch [4] for the classical optimization of the parameters of the quantum gates.

Also the code is publicly available at:
https://github.com/Bmete7/QuantumNeuralNet

2 Background

2.1 Quantum Autoencoders

2.1.1 Architecture

In the classical learning setting, autoencoders are a specific type of neural networks, which have been proven to be useful in various aspects. The architecture of the model is designed to take an N dimensional input, and try to reconstruct the same data in the output layer. The significance of the model is that, while reconstructing the output, it creates a bottleneck in the middle of the process by mapping the data in a lower dimensional *latent space*, which is later on used to reconstruct the input. The latent representation is learned through a function $f(x)$, which is referred to as the encoder network. That means the network is able to represent the same data, only by using a function $f'(x)$, which is called the decoder network,

and also the latent space representation of the data. The encoding/decoding process and the latent state representations can be seen in the Figure 2.

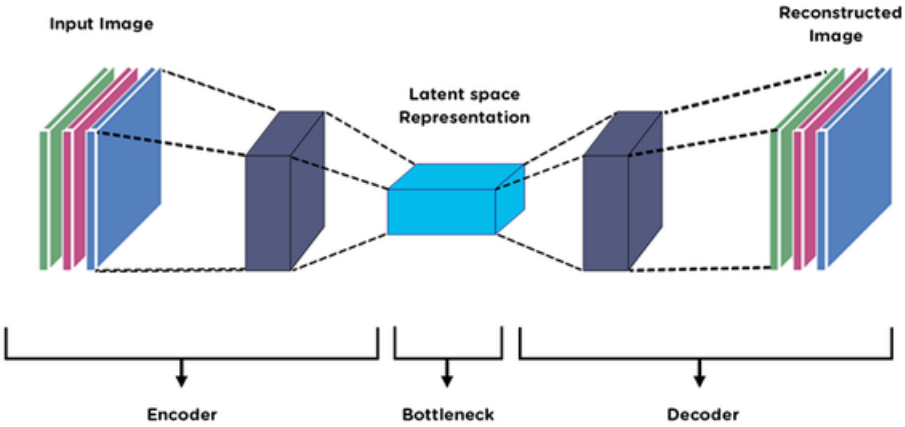

Figure 2: Autoencoder Architecture [5]

First of all, Autoencoders have been widely used for dimensionality reduction [6, 7], with the motivation of preserving the latent representation to reduce space complexity to store the actual data. If the network is trained well enough, that means the original input can always be reconstructed, using the decoder network. Secondly, the latent space representation, brings new ideas for creating a generative model[8], by instead of finding a latent space from the model, finding a set of distribution parameters, which will be sampled to generate a novel latent code. Autoencoders can also be used as a feature extractor, with the motivation of latent space having important insights about the original data.

Quantum Autoencoders, also emerged with two distinct motivations [2, 9]. First is to perform the same task that the classical models have been doing, and trying to reach a speed-up in the learning scheme, and second is to reach a better accuracy, driven by the idea of exploiting the quantum properties(i.e entanglement, superposition, fidelity measure), that might help to learn the patterns in the data in such way that classical algorithms can not capture. Another important aspect worth mentioning is, while working on a quantum data, a quantum algorithm can have an exponential reduction in the memory space required.

2.1.2 Model implementation

The model is implemented with the variational quantum circuit approach. In this method, there are quantum gates, whose optimal parameters to be found in the learning process. To be able to create such gates, there has to be unitaries with $2^n \, x \, 2^n$ dimensions, that creates an n-qubit unitary gate. However, this leads to having exponential number of parameters w.r.t number of qubits, which makes the optimization process intractable. Nevertheless, we can use the programmable circuit approach[10], basically decoding the large unitary into single qubit rotation gates and CNOTs. An exemplary programmable circuit for 4-qubit quantum system can be seen in the Figure 3.

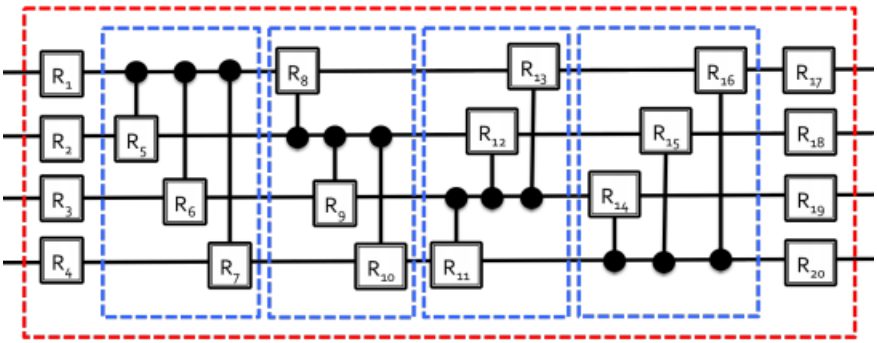

Figure 3: a Programmable circuit consisted of single qubit rotation gates and 2-qubit controlled gates [2], red dotted area represents the so called "unit-cell", which can be applied to the system several times to increase the expressiveness of the network.

Once the programmable circuit is selected, the backbone of the architecture is complete, and the defined model will work as an encoder. On the contrary to the classical autoencoders, the decoder does not have to be learned from scratch, since the unitary matrices can be inverted efficiently, the decoder can be the inverse of the encoder. Therefore, If we define the encoder network as $U^{\vec{p}}$, where $\vec{p}$ refers to the optimal parameters of the network, the decoder network will be equal to $(U^{\vec{p}})^{\dagger}$, since the whole encoder will generate a unitary.

After applying the encoder to the system, the goal of the method is to divide the quantum system into two subsystems, namely subsystem A and subsystem B. In the subsystem A, the encoder generates a "latent code" which later on can be used to reconstruct the input itself. Whereas in the

subsystem B, the goal is to generate a so-called "reference state", which is ideally the same for each and every possible data instances. If the encoding is done with a significant accuracy, since the model is be able to create one subsystem that yields identical results for any data instance, the same reference qubits can be added to the latent space to reconstruct the output. For simplicity, in our implementation, the reference state is selected as $|00\ldots0\rangle$, whose number of qubits may depend on the size of the latent space. Therefore, after the encoder, desired functionality is that the subsystem A would be consisted of the latent code, while subsystem B would generate the state $|00\ldots0\rangle$, for any input values.

One way to realize this effect is, to apply a set of swap gates between the subsystem B and B', that consists of the reference state ansatz. This means, if the network is able to create a latent space, then it also can generate the input, using the fixed reference state swapped into subsystem B. The overview of the model can be seen in Figure 4.

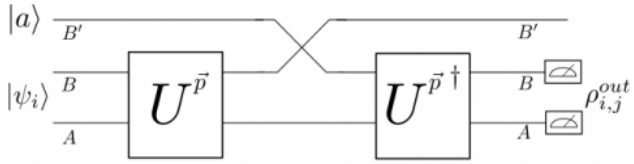

Figure 4: The network architecture. The programmable circuit overall defines $U^{\vec{p}}$ [2],

The final fragment of the model is to come up with a loss function to train the variational circuit. Influenced by the classical loss function for autoencoders, which is the $L2$ Norm of the input and the output, can be translated into QAE such as:

$$C_1(\vec{p}) = \sum_i p_i . F(|\psi_i\rangle, \rho^{out}_{i,\vec{p}}) \tag{1}$$

$$F\big(\,|\psi_i\rangle_{AB} \otimes |a\rangle_{B'}\,,\, U^{\dagger}_{AB} V_{BB'} U_{AB}\,|\psi_i\rangle_{AB} \otimes |a\rangle_{B'}\,\big) \tag{2}$$

Here, $\rho^{out}_{i,\vec{p}}$ defines the density matrix of output at the subsystems A and B, given the parameterized unitaries, $|a\rangle$ the reference state, and V stands for the unitary of the SWAP gate.

Equation (1) defines the cost function, as the fidelity between the original input $|\psi\rangle$ and the output, which is a reconstruction of the input. At the final stage, only subsystems A and B have to be measured, while subsystem B' is being traced out. Fidelity of quantum systems, is a metric to measure how similar those states are. If the states are pure, the fidelity can be found by taking the inner product of them. This can be seen from the Equation (3).

$$\begin{aligned} &F(\rho, \sigma) = |\langle\psi_A|\psi_B\rangle|^2 \\ &where \\ &\rho = |\psi_A\rangle\langle\psi_A| \, , \, \sigma = |\psi_B\rangle\langle\psi_B| \end{aligned} \tag{3}$$

If we make further simplification to the cost function (1), we get:

$$C_2(\vec{p}) = \sum_i p_i . F(Tr_A\big[\vec{U}|\psi_i\rangle\langle\psi_i|_{AB}(\vec{U})^\dagger\big] \, , \, |a\rangle_B) \tag{4}$$

The simplified cost version yields the same result as the first cost function, and it suggests that to find the fidelity between the reference state, and the expected value of the subsystem B after the encoder. Tracing out the subsystem A can be simply done by not measuring the qubits which are in the subsystem A. The measured part are also referred to as the "trash state" since, it is expected to be as the same as the fixed reference state for every possible input state. The remaining qubits that are traced out, yields the "compressed state" or the "latent space", which can be stored or used for some inference or learning tasks later on. While doing the testing, instead of measuring the subsystem A, we can measure the subsystem B and get the latent space state.
Since the compressed state might be entangled, it would be more meaningful to implement the desired process sequentially after the encoding instead of measuring the subsystem, in order not to alter the state irreversibly.

2.2 Hamiltonian Simulation

Hamiltonian, is the operator, that gives the total energy of the system, which is the summation of the potential and the kinetic energy of that system. Their eigenvalues, specifies the possible outcomes after a measurement. And the eigenspaces refer to the energy spectrum, or the state that the system will be in, after measuring its corresponding eigenvalue.

$$\hat{H}|\psi(t)\rangle = \delta|\psi(t)\rangle \tag{5}$$

Hamiltonians also govern the time evolution of quantum systems. The equation describes the evolution of a quantum system/wave function over time is called "the Schrödinger's Equation", which can be seen from the Equation (6)

$$\hat{H}|\psi(t)\rangle = i\hbar\frac{d}{dt}|\psi(t)\rangle \tag{6}$$

Given the initial state of a quantum system, if the Hamiltonian is independent of time, the state at any time t can be found by the following Equation (7).

$$|\psi(t)\rangle = e^{-iHt/\hbar}|\psi(0)\rangle = U(t)|\psi(0)\rangle \tag{7}$$

This shows that exponentiation of the Hamiltonian gives a time-dependent unitary operator, which defines the evolution itself. The problem of simulating the evolution is that the Hamiltonian grows exponentially w.r.t number of qubits, therefore making the exponentiation intractable for large number of qubits [11]. There are couple of ways to solve this issue, by finding an approximate $U'(t)$ such that $||U'(t) - e^{-iHt}|| < \epsilon$

The most common techniques to solve the Hamiltonian Simulation problem are:

- Taylor Series Expansion: $e^{-iHt} = \sum_{n=0}^{\infty} \frac{(-iHt)^n}{n!} = I - iHt + \frac{H^2t^2}{2} + \dots$
- Trotter-Suzuki: $e^{(A+B)t} = \lim_{n\to\infty}\left(e^{iAt/n}e^{iBt/n}\right)$ for $H = \sum_n a_n H_n$ by simulating each local Hamiltonian in small gaps of time periods
- Quantum Walk

Algorithm	Gate Complexity	Query Complexity
Taylor Series	$O\left(\frac{tlog^2(\frac{t}{\epsilon})}{loglog\frac{t}{\epsilon}}\right)$	$\mathrm{O}\left(\frac{d^2\|\|H\|\|_{max}log\frac{d^2\|\|H\|\|_{max}}{\epsilon}}{loglog\frac{d^2\|\|H\|\|_{max}}{\epsilon}}\right)$
Trotter Suzuki	$O\left(\frac{t^2}{\sqrt{\epsilon}}\right)$	$O\left(d^3t\left(\frac{dt}{\epsilon}\right)^{\frac{1}{2}}k\right)$
Quantum Random Walk	$O\left(\frac{t}{\sqrt{\epsilon}}\right)$	$O\left(d^2\|\|H\|\|_{max}\frac{t}{\sqrt{\epsilon}}\right)$

Table 1. Gate and Query complexities of different Hamiltonian Simulation methods

3 Implementation of the Hamiltonian Simulator using QAE

3.1 Problem Definition

The goal of the model is, given an Hamiltonian and its ground state, along with a time step t, finding the time evolution of the system with the use of quantum machine learning. The motivation of the method is to find a quantum mapping in the latent space that corresponds to the time evolution in the actual quantum system. Therefore, without having to exponentiate the Hamiltonian, our model proposes creating the approximation of the actual time evolution, after decoding the results of the time evolution in the latent space.

First part of the method is to have a trained Quantum Autoencoder, whose details has been given at the Section 2.1.2. The details on the dataset which is used, consists of the input and ground truth values, along with the Hamiltonians that are acting on the input states, has been explained in detail in Section 3.2.

The model for the Hamiltonian simulator, is a hybrid model, consisted of classical and a quantum part. First layer of the model is a fully connected(FC) layer, that accepts 2^n features and creates $6(n-k)+3(n-k)(n-k-1)/2$ features. Here, $n-k$ refers to the number of qubits in the latent space, and the total number of parameters created in the output of the FC layer corresponds to all the parameters in the parametrized quantum circuit. Therefore, the parameters of the variational quantum circuit is scalable and grows linearly to the number of qubits. The second layer is a parameterized quantum circuit, which has the same architecture as in the Figure 3. As explained before, the parameters of the programmable circuit are coming from the output of the previous layer- which is the fully connected layer.Therefore, the parameters of the quantum circuit is defined by the classical FC layer, and thus the optimization will be done classically. As it can also be seen at the model structure in the Figure 3, the rotation gates and the controlled rotation gates all require 3 real parameters, hence the model has $6(n-k)+3(n-k)(n-k-1)/2$ parameters for an $n-k$-qubit system.

Again, the goal of the circuit is to approximate the unitary $U(t) = e^{-iHt}$. Repeated iterations of the defined circuit means, applying the approximated unitary once more, and it will also correspond to the time evolution of the system with differing time steps.

Hence, after applying the quantum circuit to the latent space of the in-

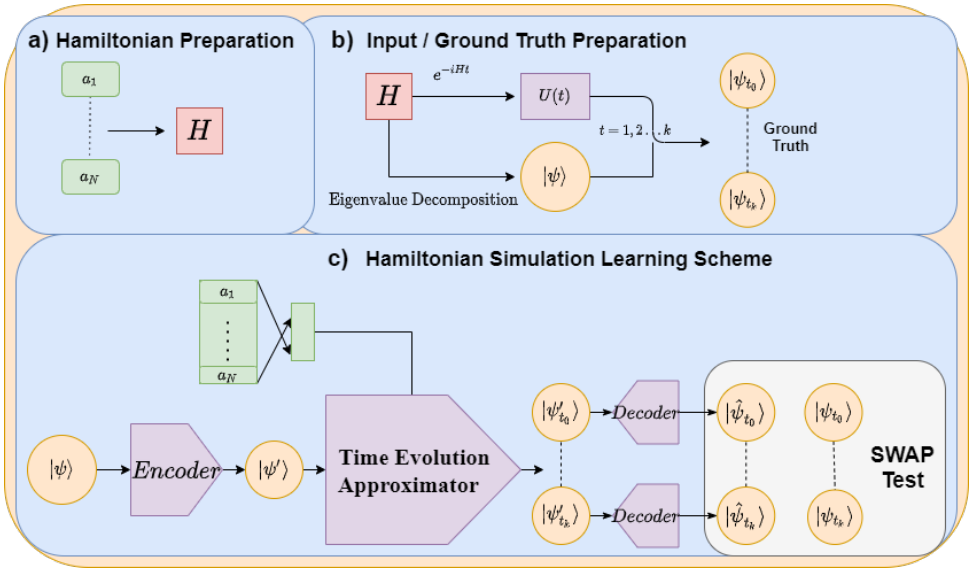

Figure 5: Learning Scheme of the Time Evolution Approximator, the section a and b refers to the preparation of the datasets, which are thoroughly explained at the Section 3.2, before running the actual Hamiltonian Simulation Approximator, there has to be an QAE, which is trained and be able to encode-decode any general n-qubit quantum state. The green entities refers to real variables, which are optimized classically, while purple refers to the parameterized quantum circuits/gates, and the orange refers to quantum states.

put once, the desired output should be close to the state which corresponds to the outcome of the time evolution, with the time step $t = 1$. When the time step parameter of the evolution is an integer, it can be visualized as applying the unitary e^{-iH} to the system t times. Therefore, the approximator should create more than one output, that coincides with the different time steps of the evolution. This is crucial because the approximated unitary for only one step in the time evolution, can be completely different than the exponentiated Hamiltonian, however the network would still find a solution that will mimic a similar operation. Thus, for this project, 3 time steps are chosen in the network architecture, also considering the number of qubits required in the simulator. Since we are still at the latent space, the decoder can be used to reconstruct the actual data in the original vector space. The accuracy of the model can be determined by, how close the approximation of a specific time evolution is, to the actual ground truth data. An exemplary

learning scheme of the model can be seen at the Figure 5

An additional aspect of the project is, since the unitaries defining the time evolution can be approximated in the latent space, a specific Hamiltonian that corresponds to that time evolution can also be found, since $\hat{H} = \frac{i.\ln \hat{U}}{t}$. The found Hamiltonian, which acts on the latent space, can also be used to analyze the actual system, or infer knowledge from it.

3.2 Preparing a Dataset

To be able to analyze systems where there is a Hamiltonian governing the time evolution, we need to have a dataset that comprises the input states, the Hamiltonian. For the ground truth values, the time evolved quantum states with different time steps can be used. Since a Hamiltonian is a Hermitian matrix, any n-Hermitian matrix can be constructed from the linear combinations of product states of Pauli matrices and the identity matrix, with non-negative and real parameters [12]. Therefore as an exemplary 2-qubit Hamiltonian can be prepared as in the Equation (8), by the randomly generated real-valued coefficients

$$H = a_{11} I \otimes I + a_{12} I \otimes \sigma^x + \cdots + a_{44} \sigma^z \otimes \sigma^z \tag{8}$$

As a reverse operation, the coefficients for the Pauli product states for a n-Hamiltonian, can also be found as:

$$a_{ij} = \frac{1}{2^n} tr\big((\sigma^i \otimes \sigma^j) H\big) \tag{9}$$

Using the methodology at Equation (8), with random non-negative real coefficients, random Hamiltonians can be prepared. The input states are chosen to be the ground state for a particular Hamiltonian. In order to find the ground state, eigenvalue decomposition has to be applied onto the Hamiltonian, and the eigenvector which corresponds to the lowest eigenvalue equals to the ground state of that particular Hamiltonian. Even though finding the ground state for a specific Hamiltonian is a costly operation which requires exponential time, this is not a huge issue since it is only done for preparing the dataset, and not the learning scheme itself. For the labels, there are 3 different time evolved states are taken into consideration, coincides for the time steps, $t = 1, 2, 3$. Therefore:

$$|\psi_{t_k}\rangle = e^{-iH_i t_k} |\psi_0\rangle \tag{10}$$

where $|\psi_0\rangle$ constructs the input and the time-evolved states $|\psi_{t_k}\rangle$ generate the ground truth values. For the embedding of the randomly generated data into the quantum circuits, Amplitude Embedding [13] method is used.

3.3 Loss Function

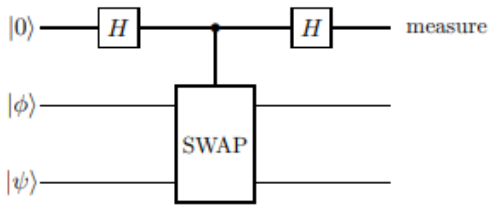

Figure 6: Implementation of the SWAP Gate [14]

The cost metric of the model is the fidelity between the trash state and the reference state. As it was explained in Section 2.1.2 successfully encoding scheme for a QAE means achieving a fix reference state at one of the subsystems for each input sample. Hence, the similarity between those states can be calculated via measuring their Fidelities. Since the reference state $|a\rangle$, and the trash state $|\psi_B^{'}\rangle$, are both pure, the fidelity between them can be defined as:

$$F(|a\rangle, |\psi_B^{'}\rangle) = |\langle a|\psi_B^{'}\rangle|^2 \tag{11}$$

To find the fidelity between two systems, SWAP Test method [14] can be used. SWAP Test suggests a solution with a fixed quantum circuit, where there is an ancillary qubit which measures the similarity between the two qubits. As it can be seen at the Equation (12), measuring the ancillary has a direct correlation with the fidelity between two states. Therefore the cost function of the whole circuit will be:

$$\begin{aligned}
|\psi_0\rangle &= |0, \phi, \psi\rangle \\
|\psi_1\rangle &= \frac{|0, \phi, \psi\rangle + |1, \phi, \psi\rangle}{\sqrt{2}} \\
|\psi_2\rangle &= \frac{|0, \phi, \psi\rangle + |1, \psi, \phi\rangle}{\sqrt{2}} \\
|\psi_3\rangle &= \frac{|0, \phi, \psi\rangle + |1, \phi, \psi\rangle + |0, \psi, \phi\rangle - |1, \psi, \phi\rangle}{2} \\
&= \Big(|0\rangle\big(|\phi, \psi\rangle + |\psi, \phi\rangle\big) + |1\rangle\big(|\phi, \psi\rangle - |\psi, \phi\rangle\big)\Big) \\
\therefore p(0) &= \frac{1}{2}\big(1 + |\langle\phi|\psi\rangle|^2\big) \\
\therefore |\langle\phi|\psi\rangle|^2 &= 2.\langle\sigma^Z|\phi_{\text{ancilla}}|\sigma^Z\rangle - 1
\end{aligned} \tag{12}$$

3.4 Training Scheme for Hamiltonian Simulator

4 Results

As mentioned before, for the quantum circuits PennyLane [15] and for the classical optimization [4] is used. The experiments are done on both 2 and 4-qubits separately. For the optimization of the parameters, Adam optimizer [16] is being used.

The average fidelities of the quantum autoencoder with different set of hyperparameters are shown in the Table 4. Also, the average fidelity graph for the 4-qubit experiment, with 80 different input states, along with their 3 different time evolved states, can be seen in the Figure 7

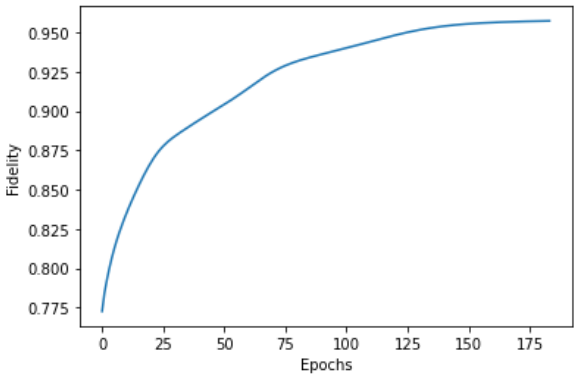

Figure 7: Average fidelity graph for the 4-qubit experiment, with 80 different input states,

Number of Qubits	Latent Space Size	Average Fidelity
2	1	0.972
4	3	0.991
4	2	0.958
6	5	0.998
6	5	0.990
6	3	0.896

Table 2. Training Accuracies for different set of hyper parameters in the training, with 250 epochs and the learning rate of 0.03

The next step in the learning pipeline is to use the encoded latent state representations to learn a time evolution in the lower dimensional space,

using the real Pauli product state coefficients which are used to generate the Hamiltonian of the actual system. Although the encoding/decoding part is running with a convincing accuracy, the results of the Hamiltonian simulator are not comparable enough to the current state-of-art methods of the Hamiltonian Simulation problem. The model seems to be on a "barren plateau" and got stuck after several iterations. A potential reason why the network does not converge well in this model, could be that the coefficients of the Pauli product states does not contain a suitable or enough information to learn the time evolution in the latent space. The reason why the Pauli product state coefficients are used instead of learning a Hermitian matrix to approximate the Hamiltonian acting on the latent space, is that the automatic differentiation with the linear algebraic operations with complex numbers using state-of-art frameworks do not yield the most optimal results. However, the concrete next steps would comprise two steps. First, is to make sure the existing model, that is using Pauli product states coefficients as input, does not generally solve a viable solution, even by choosing different set of hyper-parameters (learning rate, optimizer, different parameterized circuit architectures, longer epochs). If this is the case, the next meaningful step would be, to create a module for the automatic differentiation with complex numbers in the quantum circuit approach, and instead of having the Pauli product states as input, randomly initialized Hermitian matrices can be used. After exponentiating the Hamiltonian in the latent space (note that this is computationally tractable, since we are on a much lower dimensional space) the unitary which approximates the time evolution in the latent space with a significant accuracy can be found.

Another reason could be that the model runs on a very low number of qubits, since the simulation of large systems on a classical system can be quite complex. Although the model is supposed to generate a better time complexity for the solution to the problem, the intermediate steps, such as the preparation of the dataset can be computationally quite expensive and not tractable.

References

[1] Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Machine learning in a quantum world. pages 431–442, 2006.

[2] Jonathan Romero, Jonathan P Olson, and Alan Aspuru-Guzik. Quantum autoencoders for efficient compression of quantum data. *Quantum Science and Technology*, 2(4):045001, 2017.

[3] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[5] Classical Autoencoder Architecture. `https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f`. Accessed: 2021-03-25.

[6] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, pages 4–11, 2014.

[7] Wei Wang, Yan Huang, Yizhou Wang, and Liang Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 490–497, 2014.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[9] Kwok Ho Wan, Oscar Dahlsten, Hlér Kristjánsson, Robert Gardner, and MS Kim. Quantum generalisation of feedforward neural networks. *npj Quantum information*, 3(1):1–8, 2017.

[10] Paulo BM Sousa and Rubens Viana Ramos. Universal quantum circuit for n-qubit quantum gate: A programmable quantum gate. *arXiv preprint quant-ph/0602174*, 2006.

[11] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[12] Vladimir Privman, Dima V Mozyrsky, and Steven P Hotaling. Hamiltonians for quantum computing. In *Photonic Quantum Computing*, volume 3076, pages 84–96. International Society for Optics and Photonics, 1997.

[13] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning. *arXiv preprint arXiv:2001.03622*, 2020.

[14] Harry Buhrman, Richard Cleve, John Watrous, and Ronald De Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001.

[15] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, M Sohaib Alam, Shahnawaz Ahmed, Juan Miguel Arrazola, Carsten Blank, Alain Delgado, Soran Jahangiri, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[16] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.