Approximating the ground state energy of the molecular Hamiltonian

Thesis by

Amir Azzam

In Partial Fulfillment of the Requirements for the
Degree of
International Computer Engineering

LA SALLE UNIVERSITAT RAMON LLULL
Barcelona, Spain

2022

ACKNOWLEDGEMENTS

I would like to acknowledge and thank my supervisors, Dr. Parfait Atchadé Adelomou and Dr. Xavier Vilasís Cardona for all their help throughout this project, as this work would not have been possible without their guidance.

I would also like to thank all of my teachers and the people who have helped me during the past four years. Although it was hard at times, they helped me navigate the different subjects and manage the pressure of university.

Last but not least, I would like to thank my family and friends for being there for me during my studies. This wouldn't have been possible without all of their love and support.

ABSTRACT

Currently the drug discovery process is very slow and expensive. A lot of research has been done to accelerate this process. However, all the classical computational methods that tried to solve this issue struggle due to the complexity of the molecules. The process of simulating one molecule currently can take months to complete. As a result, a lot of the research has shifted to tools like artificial intelligence and quantum computing.

Quantum computing has the computational capability to simulate these molecules in a short time, reducing the total time it takes to discover suitable drugs. However, current quantum systems are still in early stages of development and are not capable of tackling this problem alone. This is why multiple hybrid methods have emerged to try and use the best of both classical and quantum computers.

This project looks into integrating quantum computing with machine learning in order to provide a model that would help accelerate this process. The project trained an LSTM neural network using data generated by a quantum computer. This model was trained to predict the ground state energy of the molecule. This ground state energy helps reveal important properties about the molecule itself.

In order to generate this data, a circuit was executed on a quantum computer. The circuit is used to calculate the molecular Hamiltonian of the system, which represents the total energy of that system. As a result, the data generated mapped the circuit used by the quantum computer to the energy results obtained by that circuit.

This model returned good results when tested with data it already seen, and when tested with new molecules of similar complexity. The current model suffers from a generalizability problem due to the lack of diverse training data. However, the approach does seem promising and it could, with enough training data, provide accurate approximations.

Key words: ANN, Deep learning, Entanglement, LSTM, Molecular Hamiltonian, Observables, Quantum Computers, Qubits, RNN, Schrodinger equation, Superposition.

RESUMEN

Actualmente el proceso de descubrimiento de fármacos es muy lento y costoso. Se ha investigado mucho para acelerar este proceso. Sin embargo, todos los métodos computacionales clásicos que intentaron resolver este problema no han tenido en ninguna ocasión el camino fácil debido a la complejidad de las moléculas.

Actualmente, el proceso de simulación de una molécula puede tardar meses en completarse. Como resultado, gran parte de la investigación se ha orientado hacia herramientas como la inteligencia artificial y la computación cuántica. La computación cuántica tiene la capacidad computacional para simular estas moléculas en poco tiempo, lo que reduce el tiempo total que lleva descubrir fármacos adecuados. Sin embargo, los sistemas cuánticos actuales aún se encuentran en etapas tempranas de desarrollo y no son capaces de abordar este problema por sí solos. Esta es la razón por la que múltiples métodos híbridos han surgido para probar y utilizar lo mejor de las computadoras clásicas y cuánticas.

Este proyecto busca integrar la computación cuántica con el aprendizaje automático para proporcionar un modelo que ayudaría a acelerar este proceso. El proyecto propone una red neuronal LSTM utilizando datos generados por una computadora cuántica. Este modelo fue programado para predecir la energía del estado fundamental de la molécula. Esta energía del estado fundamental ayuda a revelar propiedades importantes sobre la propia molécula. Para generar estos datos, se ejecutó un circuito en una computadora cuántica. El circuito se utiliza para calcular el hamiltoniano molecular del sistema, que representa la energía total de dicho sistema. Como resultado, los datos generados asignaron el circuito utilizado por la computadora cuántica a los resultados de energía obtenidos por ese mismo circuito. Este modelo trajo consigo buenos resultados cuando se probó con datos con los que ya había trabajado anteriormente y cuando se probó con nuevas moléculas de una complejidad similar. El modelo actual sufre de un problema de generalización debido a la falta de datos de prácticas similares alternativas. Sin embargo, el enfoque parece prometedor y podría, con suficientes datos de prueba, proporcionar aproximaciones precisas.

Palabras clave: ANN, aprendizaje profundo, entrelazamiento, LSTM, hamiltoniano molecular, observables, computadoras cuánticas, qubits, RNN, ecuación de Schrödinger, superposición.

RESUM

Actualment el procés del descobriment de fàrmacs és molt lent i costós. S'han fet moltes investigacions per accelerar aquest procés. Tanmateix, tots els mètodes computacionals clàssics que intenten resoldre aquest problema ténen constants dificultats a causa de la complexitat de les molècules. El procés de simulació d'una molècula actualment pot trigar mesos a completar-se. Com a resultat, gran part de la investigació s'ha desplaçat cap a eines com la intel·ligència artificial i la computació quàntica. La computació quàntica té la capacitat computacional de simular aquestes molècules en poc temps, reduint el temps total que es necessita per descobrir fàrmacs adequats. Tanmateix, els sistemes quàntics actuals encara es troben en les primeres etapes de desenvolupament i no són capaços d'abordar aquest problema per si sols. És per això que múltiples mètodes híbrids han sorgit per intentar utilitzar el millor dels ordinadors clàssics i quàntics.

Aquest projecte estudia la integració de la computació quàntica amb l'aprenentatge automàtic per tal de proporcionar un model que ajudi a accelerar aquest procés. El projecte va entrenar una xarxa neuronal LSTM mitjançant dades generades per un ordinador quàntic. Aquest model va ser configurat per preveure l'energia de l'estat fonamental de la molècula. Aquesta energia de l'estat fonamental ajuda a revelar propietats importants sobre la pròpia molècula. Per tal de generar aquestes dades, s'ha executat un circuit en un ordinador quàntic. El circuit s'utilitza per calcular l'hammiltonià molecular del sistema, que representa la seva energia total. Com a resultat, les dades generades permeten construir el circuit utilitzat per l'ordinador quàntic amb els resultats d'energia obtinguts pel mateix circuit. Aquest últim model ha donat bons resultats quan es va provar amb dades ja vistes i amb noves molècules de complexitat similar. El model actual pateix un problema de generalització a causa de la manca de dades a les diverses proves. Tanmateix, l'exploració sembla prometedora i podria, amb les dades necessàries per a l'assaig, proporcionar aproximacions precises.

Paraules clau: ANN, Deep learning, Entanglement, LSTM, Hamiltonià Molecular, Observables, Computadors Quàntics, Qubits, RNN, Equació de Schrodinger, Superposició.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

ANN. Artificial Neural Network is an information processing system modeled after the nervous system used in deep learning.

Deep Learning. Deep learning is a form of machine learning that uses artificial neural networks.

Entanglement. Entanglement is the phenomenon where two particles share their state, and can not be expressed without knowing the state of the other.

LSTM. Long Short Term Memory is a type of recurrent neural network that maintains a long and short memory.

Molecular Hamiltonian. The molecular Hamiltonian is an operator that represents the total energy of a molecular system.

Observable. In quantum mechanics, an observable is an operator that reveals some properties of the quantum system .

Quantum Computers. These are computers that use qubits as their base computation unit.

Qubits. Qubits are the quantum bits and are the basic computation unit for a quantum computer.

RNN. Recurrent Neural Networks are a type of artificial neural network that processes the input sequentially.

Schrodinger equation. In quantum mechanics, the Schrodinger equation governs how a quantum system evolves over time.

Superposition. Superposition is the state when a particle is in a linear combination of all the possible states that particle can take.

Chapter 1

INTRODUCTION

In the past few years, we've seen the pandemic's impact on people's life. This pandemic made us understand the importance of vaccinations and the need to speed up their development process. Which would allow us to respond more quickly to new emerging diseases.

Generally speaking the drug discovery process is divided into five main stages [31], the first aiming at identifying potential molecules and compounds that would help fighting the disease. Molecules are made up of two or more atoms joined together to create a chemical bond. The atoms in the molecule share electrons, which allows each of them to become more stable by lowering the atom's overall energy [42].

In this stage of the drug discovery process, thousands of molecules need to be tested, and only few hundreds of these molecules moves to the pre-clinical testing stage [27]. This screening process takes many years to complete and does not guarantee that the selected molecules would be able to create an effective drug. Currently, computers are used to try and speed up this process. Some classical methods like molecular dynamics (MD) [61] simulations, density functional theory (DFT) [81], and artificial intelligence (AI) [86] are used to help discover these molecules.

MD and DFT try to approximate the values of the molecular ground state energies by simplifying the equations involved in calculating these values [61, 81]. The ground state energy is the lowest possible energy a molecule can take, which is when the molecule is in the most stable configuration [41]. Nevertheless, even after applying these approximations to reduce the computational complexity, some of these simulations still can take a long time to complete [29].

Many deep learning algorithms have recently emerged to address this issue. Some of these approaches are focused on predicting bioactivity or toxicity, target identification, and molecular optimization [86]. Other biomedical challenges, such as estimating ground state energy, have received greater attention in recent years. However, the lack of accurate training data in some cases poses a problem for the machine learning algorithms. Other approaches attempted to utilize machine learning as a supplement to current tools such as DFT and MD to achieve higher accuracy [13].

Quantum computers, in contrast to the approaches previously stated, are significantly better at modeling quantum systems. This is because they exploit the properties of quantum mechanics to reduce the system's computational complexity. A quantum computer could theoretically decrease the computational complexity of such a problem to a polynomial cost [6]. These systems, however, are still in the early stages of development, and the technology is not yet mature enough to address these major issues [70].

The fundamental idea of the project emerged from researching how quantum computers find this ground state energy. These systems usually create a formulation, which is represented as a quantum circuit [83]. These quantum circuits use quantum gates to perform operations, similar to the logic gates used by classical computers. Quantum circuits can be expressed as a series of coefficients each multiplied by a group of quantum gates (see section 3.2). The main question raised by this project is: would it be possible to learn the patterns in this formulation in order to predict the output without having to using a quantum computer?

In order to test this hypothesis, the project used a combination of machine learning and quantum computing. The quantum computer was used to generate the training and testing data. Then, this data was used to build a regression model that would be able to predict the ground state energy based on the formulation used to compute it.

The next chapter looks into current classical methods used to solve the problem addressed by the project. Moreover, the chapter will also look into similar work done in regard to this problem.

Then, chapter 3 will look at the current state of the art for all the technologies used in this project. Furthermore, this chapter discusses the fundamental concepts needed to understand the proposal.

Chapter 4 discusses the proposed solution for this project, and all the steps taken to reach that final solution are explained. While, chapter 5 will look at the results of the different approaches tried in this project.

Chapter 7 will look into the temporal cost of the project. Finally, chapter 8 will discuss the conclusions of the project and the future lines of work that will follow this project.

Chapter 2

LITERATURE REVIEW

Currently, most drug research tries to accelerate the drug discovery process by using classical computational methods. These methods usually include molecular dynamic simulation, applying the density functional theory, and artificial intelligence. However, with the current development of quantum computing technology the future of quantum chemistry looks very promising. Using quantum computers to solve quantum chemistry problems has an inherit advantage [21]. That is because quantum computers excel at simulating quantum systems due to the way they operate. Nevertheless, quantum computers are still in the NISQ (Noisy Intermediate-Scale Quantum) era [70], thus are not able to perform operations on the scale required by the pharmaceutical research yet. This section looks at some of the most common classical approaches used in drug discovery today, as well as work done with quantum computers that is similar to the approach proposed by this project.

2.1 Density Functional theory

Density functional theory (DFT) is a quantum-mechanical method used to calculate the electronic structure of molecules and be able to find the ground state energies of these molecules. This theory was first used in quantum chemistry after the 1990s. This is due to the emerge of new approximations that made DFT sufficiently accurate for the quantum chemistry applications [81].

One of the advantages of such system is its ability to simplify the computational complexity of the electron-electron interactions (known as the many-body interactions). Normally, such interactions grow in computational complexity exponentially as more electrons and orbitals are added into the observed system. However, DFT avoids that by calculating the electronic structure (as a density functional) and using it as the basis for this calculation instead of considering every single interaction between the electrons set.

Normally, the energy of a molecular system is given by solving the many-body Schrodinger equation. This equation is a 3N-spatial-coordinates problem, where N is the number of electrons in the system. This is an NP-complete problem as the complexity of this problem grows exponentially with the addition of more electrons [19]. However, DFT suggests that the energy of the system can be found in three spacial coordinates by focusing on the electronic density given the Born-Oppenheimer approximation. The Born-Oppenheimer approximation suggests that the electronic energy and the electronic properties of a system can be uniquely defined by the ground state electron density [95].

Moreover, as DFT can handle any element in the periodic table with any molecular arrangement without the need to modify any experimental input parameters. This provides a very strong predictive advantage even for newly discovered molecules.

There are many software tools that can simulate this calculation. For instance, The Vienna Ab initio Simulation Package (VASP) [89] which is a computer program that can calculate electronic structures and is able to compute the many-body Schrodinger equation using either the DFT or Hatree-Fork (HF) approximations [76, 30]. This program can also be used to model quantum-mechanical molecular dynamics.

DFT can be combined with Molecular dynamics simulations to provide a more versatile and accurate simulations, allowing the researchers to further understand the chemical structure of the molecules [64].

2.2 Molecular Dynamics Simulation

Molecular dynamics (MD) [61] is a method of analyzing the physical properties of atoms and molecules in a quantum system. This facilitates ways to understand the physical structure and how the system will react to changes in its environment under some specific conditions.

This simulation allows to study the way certain proteins or molecules move and behave in different given environments. For instance, when a small drug molecule approaches its target, the target is not a frozen molecule but rather in constant motion. This motion plays a critical role in the binding process of these drugs with the target molecules.

The main disadvantage of the MD simulations is that they are very computationally demanding considering the complexity of the systems that they simulate. Moreover, they are mostly focused on the effect of the forces on the different molecules but they don't focus a lot on the quantum mechanics effects on these molecules. This makes them less accurate when it comes to simulating quantum mechanical systems [47].

The way this simulation works is by starting with an initial state of the system. Then, calculating the molecular forces acting on the atoms in the system. After that, the atoms are moved according to the calculated forces acting on them. Finally, the simulation is advanced by a frame, and the current state of the system is considered to be the new initial state repeating the previous two steps.

The main limitation of MD is the high computational demands of most chemical systems. For example, a relatively small system of 25K atoms can take several months to be processed. Normally, MD simulations run simultaneously on different computer clusters or supercomputers to make the process faster. However, finishing one simulation can still take up to several months to complete.

It is possible to integrate some other methods into this simulation like DFT to represent the quantum mechanics effects. Nevertheless, this will increase the complexity of the simulation, thus increasing the time it takes to finish this simulation.

2.3 Artificial Intelligence

Artificial intelligence is used in various sectors of the drug discovery and development process. The areas that benefit the most from AI include drug design, chemical synthesis, drug re-purposing, and drug screening [26]. AI could also be used to predict the bio-activity and the toxicity of the drug. Generally AI might not provide the most accurate results, however, it provides a fast way to validate the drug targets and optimize the design of the drug.

AI has multiple stages to be able to predict the output to a given input. There needs to be a way to represent the knowledge in a way that makes it easier for the machine learning method to detect patterns, and be able to learn from the training input. Then, using this training data a machine learning model is built and optimized. Machine learning algorithms include a wide variation of methods and ways to classify the input data. Deep learning (DL) [57] is the most popular machine learning methods nowadays. DL uses artificial neural networks (ANNs), which is modeled after the nervous system neural networks. These networks has multiple input, intermediate, and output nodes that work together to calculate the output of a given input. These nodes are interconnected using different weights which the network modifies during the learning process to fit the patterns present in the input of the network.

Although Deep learning is the most common technique nowadays, other techniques like Decision Trees and Instance-based algorithms are still wildly used. These methods include algorithms like random forests [71], support vector machines (SVM) [65], and K-NN [39]. These methods generally return less accuracy than DL methods, but they offer more control over what happens within the model than most DL methods.

Current chemical researches mostly rely on Deep Learning. This is because it offers a very flexible framework to train a model, with a lot of pre-built Network options optimized for different purposes. Moreover, There are a lot of tools developed for AI that can be directly applied to chemical problems. For example, DeepChem [72] is a library that offers a Multilayer perceptron (MLP) [49] model developed to find a suitable candidates in the drug discovery process [72].

2.4 Quantum Computers

Quantum computers use particles known as quantum bits (qubits) as their base computational unit instead of standard bits. These particles use the properties of quantum mechanics to perform computations more efficiently. Quantum computing offers computational power that is superior to that of classical computers [5].

However, today's quantum computing systems are still noisy [75], making them prone to errors. Furthermore, these systems do not yet have enough qubits to perform operations on a large enough scale to be useful.

Theoretically, a quantum computer can possibly find the ground state energy faster than any classical computer, as they are very efficient in solving optimization problems [1, 10, 9, 8, 11, 67, 7, 4]. This is because quantum computers can explore multiple solutions simultaneously, making them more efficient at exploring the solution space.

Moreover, using the variational principle [36] finding the ground state energy can be mapped to an optimization problem. Then, this problem can be solved using different algorithms like the variational quantum eigensolver (VQE) [69] (see section 3.1), that uses the quantum properties to provide an exponential speedup in execution time.

Algorithms like these still use both classical and quantum computers to be able to solve complex problems on a scale that current quantum systems can compute. These algorithms are known as quantum machine learning (QML) [18]. QML allows current research to take advantage of the computing power current quantum systems can offer, by executing parts of the algorithm on a classical computer.

Nevertheless, The utility of this approach is still limited by the number of qubits a quantum processor has, making them unable to find the ground state energy for larger molecules even with the help of classical computers.

2.5 Related Work

Several research has been conducted in an attempt to estimate the Hamiltonian's ground state energy. Some of these studies concentrate on utilizing deep learning to try to predict this ground state energy. There are different approaches taken by different researchers to create a generalized model capable of predicting this ground state energy.

The paper "Deep learning and the Schrödinger equation" [63] looks into the possibility of using convolutional neural networks to bypass the need to numerically solve the Schrödinger equation. Their idea was to let the neural network learn the mapping between the potential and the ground state energy. However, the data they used was randomly generated and there was no analytic form for either the potentials or the energies the model was tried with.

Moreover, in similar paper titled "Numerical solution of the Schrödinger equation by neural network and genetic algorithm" [79], they described a way to solve the Schrödinger equation using genetic algorithms and artificial neural networks. In their work they used the generic algorithm to optimize the neural network parameters and improve the convergence of the network.

Another similar work was done in the paper "Tree based machine learning framework for predicting ground state energies of molecules" [43]. In this paper they discusses the benefits of the boosted regression tree algorithm when used to predict the ground state energy of molecules.

The paper "Predicting excited states from ground state wavefunction by supervised quantum machine learning" [50], takes a similar approach taken in this project. In this paper, they attempt to calculate the molecule's excited states using a supervised machine learning model. This models is trained using the wave function of the ground state energy. The general idea of this paper is not exactly the same as the idea proposed by this project. However, the approach taken in here resembles the one used by the project.

Unlike the previous approaches, this project studies the possibility of using the quantum circuit prepared to calculate the Hamiltonian of a system as the input of a sequential model. Then, this model tries to predict the output that would have been obtained by executing this circuit.

This approach uses a recurrent neural network to learn the input. Recurrent neural networks are widely used for natural language understand; and this project tries to treat the quantum circuit as a language model that could be learned by this network. This would allow machine learning to build quantum computing simulators that are less demanding computationally than current simulators.

Chapter 3

FUNDAMENTALS AND STATE OF THE ART

3.1 Quantum Computing

Classical computing uses the binary system as its operational basis, thus every piece of information can be broken into a sequence of bits. Each one of these bits can take on one of either two values, 1 or 0. Following the Boolean algebra a binary system can perform a set of simple operations that combined would allow the system to perform much more complex operations. To compute more efficiently in such systems, the application is divided into many threads that often execute on multiple cores, increasing the computation speed linearly.

However, finding the best solution to a problem on these systems still requires the hypothesis to be tested on every possible solution. This means that the more the solution space grows, the more complex finding a solution becomes.

A quantum computer, on the other hand, employs quantum mechanics techniques to tap into the computational power of particles. These particles can take on states that are equivalent to those provided by a classical system, as well as a combination of these states. This permits a single particle to hold several values at the same time, allowing the system to perform at an exponentially faster rate.

In order to understand the way a quantum computer operates it is essential to have a basic understanding of some quantum mechanics fundamental concepts. Most of the information in the following sections are primarily based on a book called "Quantum Computing for Computer Scientists" written by Noson S. Yanofsky and Micro A. Mannucci [94]. This books covers the basic information needed to understand the logic behind a quantum computer.

Introduction to Quantum Theory

Quantum mechanics defines the physical properties at the scale of atoms and particles. Quantum mechanics can predict how a system will behave in the future, but it cannot determine for certain the outcome of that system. This is due to the fact that particles exhibit wave-like behavior, which is mapped to a wave function that provides a probability distribution of how the particle might behave in the future. These quantum systems operate with probabilities, where the probability of an outcome is given by a complex number.

Normally in probability, for an experiment with N outcomes, each outcome is associated with a weight p, where p is a real number that satisfies $0 \leq p \leq 1$ and $\sum_{i=0}^{N} p_i = 1$. On the other hand, in quantum mechanics these outcomes are associated with a complex number c as their weight. This complex number needs to satisfy these two conditions: $0 \leq |c|^2 \leq 1$ and $\sum_{i=0}^{N} |c_i|^2 = 1$. However, due to the properties of complex numbers, the squared moduli of two complex numbers are not always less than the modulus squared of their sum. In other words, given two complex numbers c_1 and c_2, then $|c_1 + c_2|^2$ is not necessarily grater than $|c_1|^2$ nor $|c_2|^2$. This is because complex numbers can cancel each other when added.

This property in quantum mechanics is known as interference [32], and it plays a crucial role in the formulation of the quantum theory. This phenomenon is extremely important in quantum computing, as it helps to understand how to control the probability of a certain system and add a specific bias to some of the outcomes. Furthermore, it is one of the causes of quantum decoherence. As some incidental measurements may cause the system's state to be disrupted as a consequence of the interference.

The possible outcomes of the probability distribution defined by the wave function form the state basis that particle can take. Each these states is represented by a complex column vector of N dimensions $[c_0, c_0, \cdots, c_{N-1}]^T$, where N is the number of dimensions in the state space. Each of the positions in that column vector represents the probability of the measured particle being in corresponding state. Moreover, the values in this column vector are known as the complex amplitudes of the state. Each state from the set $[x_0, x_1, \cdots, x_N]$ is denoted by the direct Ket notation as $|x_i\rangle$.

Therefore a state $|\psi\rangle$ can be written in terms of the state space basis associated with their complex amplitudes as follows:

$$|\psi\rangle = c_0 |x_0\rangle + c_1 |x_1\rangle + \cdots + c_{N-1} |x_{N-1}\rangle \tag{3.1}$$

In other words, the state in a quantum system can be defined as a linear combination of basis states of the state space. This phenomena is known as **superposition**. which means that the particle is simultaneously in a combination of all the possible states it can take. Nevertheless, these states are not equally present in the superposition state of the particle, but they are rather proportional to the complex amplitude corresponding to that state in the particle's wave function. Superposition is one of the properties that make quantum computers superior to classical computers. This because superposition (combined with Entanglement, which will be discussed later in this section) can allow a system to simultaneously exist in multiple states, meaning that the system will be computing the output with multiple input values at the same time. This provides an exponential speed up in execution time and an exponential increase in memory storage as it allows the hypothetical quantum system to store 2^N values using only N particles [48].

Observables

Observables in classical physics are things that can be measured, such as momentum or position. In quantum mechanics, observables are operators that, when applied to a particle, reveal some physical properties about the quantum state of that particle.

When an observable Ω is applied to a state vector $|\psi\rangle$, it results in mapping this state vector to a new state $\Omega |\psi\rangle$. Moreover, This observable Ω is a linear operator belonging to the Hilbert space representing the state vector $|\psi\rangle$. The Hilbert space can be defined as an infinite-dimensional inner product vector space [37]. This operator is represented in the form of a Hermitian matrix. A hermitian matrix is a spacial matrix in the complex Hilbert space where the matrix is equal to its conjugate transposed, as shown in equation 3.2.

$$M \text{ is Hermitian} \Longleftrightarrow M = M^\dagger \tag{3.2}$$

The eigenvalues of a hermitian matrix are real, and the eigenvectors are orthogonal. This means that the observable has a set of real eigenvalues that represent the possible outcomes of the measurement the observable is conducting. Moreover, the eigenvectors of the observable form a basis for the new state vector of the particle after the observable ineracted with it. Each of these eigenvectors also represents the resulting state of the particle after the measurement. For example, let Ω be an observable with $[\lambda_1, \lambda_1]$ as its eigenvalues, and $[|e_1\rangle, |e_2\rangle]$ as their corresponding eigenvectors. The result of measuring the observable Ω on the state $|\psi\rangle$ is either λ_1 or λ_2. Depending on the result of the measurement, the state $|\psi\rangle$ is the eigenvector corresponding to the measured eigenvalue. This means that if the output of the measurement was λ_1 then the state $|\psi\rangle$ becomes $|e_1\rangle$.

Particle Spin

An experiment known as the Stern–Gerlach experiment [33] was conducted in 1922 to study the behaviour of particles traveling through an inhomogeneous magnetic field. The experiment was done using a device that generated a inhomogeneous magnetic field between its pols shown in figure 3.1. Then, a beam of particles were passed through the device. These particles were not previously prepared into any specific states. After that, The particles deflected under the magnetic field before hitting a screening device. Originally, it was thought that the deflection of the particles would be proportional to the original state the particle had before entering the magnetic field. Meaning that the different particles will be effected differently by the magnetic field depending on their original state. However, the experiment showed that the beam was split into two possible states with a probability of 50% for a particle to be found in either state.

This phenomena led the researchers to believe that the particles have a spin of their own. Therefore, the particle generates its own magnetic field that then was aligned with the external magnetic field generated by the device. This experiment was repeated multiple times after rotating the pols of the machine generating the external magnetic field. The beam of particles consistently split into the two states with respect to the new magnetic field.

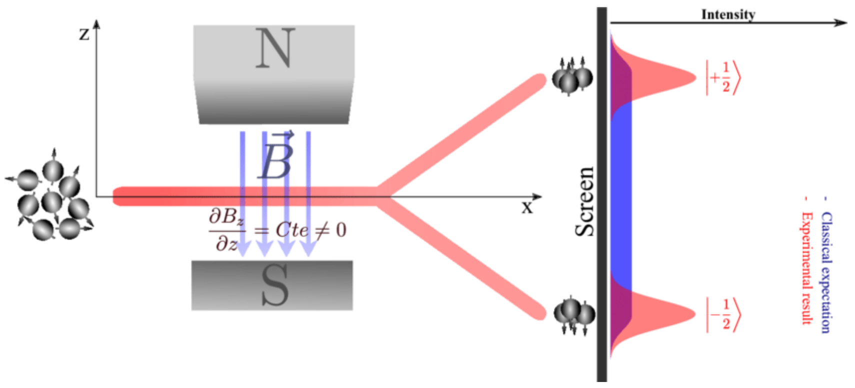

z
N
$\vec{B}$
$\frac{\partial B_z}{\partial z} = Cte \neq 0$
S
x
Screen
Intensity
$\left|+\frac{1}{2}\right\rangle$
$\left|-\frac{1}{2}\right\rangle$
- Classical expectation
- Experimental result

Figure 3.1: A figure showing the Stern-Gerlach experiment where a beam of atoms is sent through a inhomogeneous magnetic field. The atoms came out of the field and collided with a screen perpendicular to their initial trajectory. The distribution of the hits observed on the screen shows that the atoms were only seen in two states, corresponding to atoms with spin up and atoms with spin down with a probablity of 50% for an atom to hit in either state. This figure is taken from the paper "Entanglement in high dimensional quantum systems" by Saideh, Ibrahim [73]

According to the results of this experiment the particle can only exist in two states with respect to the direction the particle was measured in. The particle is either spinning clockwise, or anti-clockwise. For instance, if the measuring device was oriented vertically (along the z axis), then the two states for a given particle are represented as spin up $|\uparrow\rangle$ or spin down $|\downarrow\rangle$. As a result, the state vector of these particles can be represented as follows:

$$|\psi\rangle = c_0 |\uparrow\rangle + c_1 |\downarrow\rangle , \tag{3.3}$$

where c_0 and c_1 are the complex amplitudes of this state vector.

This measuring device can be thought of as an observable. When this observable acts on a particle, it changes the particle's state. The resulting particle state, as mentioned earlier, is a linear combination of the observable's eigenvector. However, whenever a particle is measured, it is always found to be in only one of its possible states, and not in a linear combination of them. In other words, the measurement process disturbs the particle's state. Not only does it transfer the particle's state to a state space defined by the measuring device's eigenvectors, but it also forces the particle to collapse onto one of its states.

This explains why, regardless of the machine's orientation, the particles are always collapsed into a state that is aligned with the magnetic field acting on that particle.

Qubits

In classical computation the principle information unit is known as a bit. A classical bit can be in one of either two states, 1 or 0. However, knowing that particles in the quantum realm can exist in multiple states simultaneously, This information unit can then be represented in quantum mechanics with a quantum bit (qubit). Qubits are particles that have two states and define a 2-dimensional quantum system. These particles can be in states 0 or 1, but also they can exist in a linear combination of these states. Therefore, the state of a qubit can be defined as follows,

$$c_0 |0\rangle + c_1 |1\rangle , \tag{3.4}$$

where the probability of the qubit being in the 0 state is given by $|c_0|^2$, and the probability of it being in the 1 state is given by $|c_1|^2$.

When measuring qubits, they collapse to one of two states, $|1\rangle$ or $|0\rangle$. As a result, a collapsed qubit has no advantage over a classical bit because they both can only be in one of two states. This isn't to say that qubits don't provide any advantages over traditional bits. Qubits only collapse when they are measured; however, qubits are only measured when the result has to be obtained at the end of the execution. While during execution, the qubits make use of quantum mechanics properties to provide all of the benefits that a quantum system has over its classical equivalent.

Moreover, in classical computation, a single bit is not sufficient enough to perform most operations, and usually many bits are combined to store and process more information. Similarly in quantum computation, a quantum system composed of a single qubit is not very useful. This is why multiple quantum systems need to be combined to form a bigger system with enough processing power to solve large problems.

Quantum systems can be combined by finding the tensor product of their state spaces, and the states forming the new system will be constructed by the tensor product of the states of the old systems. This phenomena is known as **Entanglement** , which allow multiple qubits to operate as a unified, inseparable quantum system. For instance, consider two qubits q_0 and q_1 represented by the basic states $[|0\rangle_0, |1\rangle_0]$ for the first qubit and $[|0\rangle_1, |1\rangle_1]$ for the second qubit. After entangling these two qubits, the resulting system can be represented as,

$$\begin{aligned} \left|\psi_{1,2}\right\rangle &= c_0(|0\rangle_0 \otimes |0\rangle_1) + c_1(|0\rangle_0 \otimes |1\rangle_1) \\ &\quad + c_2(|1\rangle_0 \otimes |0\rangle_1) + c_3(|1\rangle_0 \otimes |1\rangle_1) \end{aligned} \tag{3.5}$$

The state $|0\rangle_0 \otimes |0\rangle_1$, represents that both qubits q_0 and q_1 are in the state $|0\rangle$. While the state $|1\rangle_0 \otimes |0\rangle_1$ represent that qubit q_0 is in state $|1\rangle$ and q_1 is in state $|0\rangle$. The quantum system's new state cannot be described in terms of only one of the qubits, but both qubit states need to be measured to know the state of the system.

This phenomena becomes particularly interesting when the complex amplitude of the states are not equal. For example, consider a quantum system with two entangled qubits, and the following state,

$$\begin{aligned} \left|\psi_{1,2}\right\rangle &= (4+i)(|0\rangle_0 \otimes |0\rangle_1) + (0)(|0\rangle_0 \otimes |1\rangle_1) \\ &\quad + (0)(|1\rangle_0 \otimes |0\rangle_1) + (i)(|1\rangle_0 \otimes |1\rangle_1) \\ \left|\psi_{1,2}\right\rangle &= (4+i)(|0\rangle_0 \otimes |0\rangle_1) + (i)(|1\rangle_0 \otimes |1\rangle_1) \end{aligned} \tag{3.6}$$

In this case, knowing the state of the first qubit can tell us a lot of information about the state of the other. For instance, if the measurement of the first qubit showed that it is in state $|0\rangle$. Then, it is possible to conclude that the state of the second qubit is $|0\rangle$ as well given that the state $|0\rangle_0 \otimes |1\rangle_1$ has a coefficient of 0.

Unitary Operators and Time-dependant Systems

An observable acting on a quantum system is a time-independent operation, that is not necessarily reversible. Nevertheless, quantum systems do evolve over time, and this evolution is produced by unitary operation. Unitary operations are operators that satisfy

$$UU^* = U^*U = I, \tag{3.7}$$

where U is the unitary operator and U^* is the adjoint of that operator. On the other side of the equation, I denotes the identity operator. Moreover, these operators closed under composition and inverse. Meaning that the product of two unitary operators is also unitary, and the inverse of a unitary operator is unitary.

Unitary operators present the quantum system with a very important property, which is reversibility. This is because the effects of a unitary operator can be reversed by applying the inverse of that operator onto the quantum system. For instance, let U be a unitary operator, then

$$U^{-1}(U|\psi\rangle) = |\psi\rangle . \tag{3.8}$$

Therefore, a quantum system evolves over time by applying a sequence of unitary operators on it. As a matter of fact, let U be a rule that associates with every instance of time $t_0, t_1, \cdots, t_n$ a unitary operator $U[t_0], U[t_1], \cdots, U[t_n]$, then the system's state $|\psi\rangle$ at time t+1 can be given by:

$$|\psi(t+1)\rangle = U[t]\,|\psi(t)\rangle . \tag{3.9}$$

This time evolution in a quantum system is governed by the Schrodinger equation shown in equation 3.10. This equation built on the fact that the total energy of an isolated system is preserved with the evolution of time. The total energy of an isolated system is an observable known as the Hamiltonian of the system ($\hat{\mathcal{H}}$). According to the Schrodinger equation the quantum state $|\psi(t)\rangle$ changes with respect to time t, is proportional to $|\psi(t)\rangle$ multiplied by the Hamiltonian of that system. If the Hamiltonian of the system is known and this equation is solved, then it would be possible to calculate the evolution of the quantum system over time.

$$\frac{|\psi(t+\delta t)\rangle - |\psi(t)\rangle}{\delta t} = -i\frac{2\pi}{\hbar}\hat{\mathcal{H}}\,|\psi(t)\rangle \tag{3.10}$$

Quantum gates

Quantum gates are unitary operators that act on a qubit. These operators must be reversible, allowing the system to be restored to its original state even after a series of operations have been performed on it. Since not all classical gates are reversible, they cannot all be mapped to quantum gates. An AND gate, for example, cannot be transferred to quantum computing since it is impossible to reconstitute the system's initial state after applying this gate to it. In reality, if an AND gate's output is $|0\rangle$, the input may be $|00\rangle$, $|01\rangle$, or $|10\rangle$.

One of the most common quantum operators that act on a single qubit are the Pauli operators. There are three Pauli operators that measure the spin of the particle across the 3 dimensional Euclidean space. these operators are represented by a 2x2 unitary matrices:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3.11}$$

$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{3.12}$$

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{3.13}$$

Each one of these matrices correspond to the spin of the particle in one of the three Euclidean space's dimensions. For instance, σ_x corresponds to the spin of the particle along the x axis. Furthermore, these matrices serve as bases for the real vector space, therefore any 2x2 matrix in this space may be expressed as a linear combination of these matrices.

The Hadamard gate is another single-qubit gate that is very important in quantum computing. The Hadamard gate places a qubit with initial states $|0\rangle$ or $|1\rangle$ in a superposition of both states. This transformation is represented by the following matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.14}$$

For example, if this gate act on a qubit in the state $|0\rangle$, then

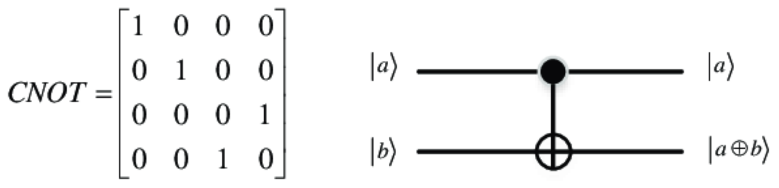
$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{3.15}$$

Following the calculations in equation 3.15, the effect of this gate on the states $|0\rangle$ and $|1\rangle$ can be represented as follows:

$$|0\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{3.16}$$

$$|1\rangle \rightarrow \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{3.17}$$

Nevertheless, there are other gates that act on multiple qubits simultaneously. One of the most popular gates that act on two qubits is the Controlled-Not gate (CNOT) gate. This gate performs the Not operation on the second qubit, only if the the first qubit is in the state $|1\rangle$. The unitary matrix representing this gate along side the circuit representation are shown in figure 3.2



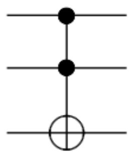
$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 3.2: This figure shows the CNOT quantum gate symbol and the associated matrix. This image is taken from the article "A Novel Quantum Visual Secret Sharing Scheme" published in the IEEE Access journal[59]

a	b	a’	b’
$\lvert 0\rangle$	$\lvert 0\rangle$	$\lvert 0\rangle$	$\lvert 0\rangle$
$\lvert 0\rangle$	$\lvert 1\rangle$	$\lvert 0\rangle$	$\lvert 1\rangle$
$\lvert 1\rangle$	$\lvert 0\rangle$	$\lvert 1\rangle$	$\lvert 1\rangle$
$\lvert 1\rangle$	$\lvert 1\rangle$	$\lvert 1\rangle$	$\lvert 0\rangle$

Table 3.1: This table describes the output of a CNOT gate. On the right side of the table the input states are represented. Whereas, the left side contains their corresponding outputs

The inputs and outputs of this gate are represented in the table 3.1. This gate can be extended to a gate that acts on three qubits known as the Toffoli gate (CCNOT). This gate Negates the value of the third qubit if and only if the first two qubits are in state $|1\rangle$. The gate symbol and the matrix representing this gate are shown in figure 3.3.



$$\text{CCNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 3.3: This figure shows the Toffoli quantum gate symbol and the associated matrix. This image is taken from the article "Developing A Quantum Circuit Simulator API" published by the ACTA Universitatis Cibiniensis's journal[28]

This gate is known for being a universal gate, thus using this gate it is possible to create any classical gate in a reversible manner. For instance, creating an AND gate (which is normally not reversible) using the Toffoli gate is possible. This is done by setting the third bit to zero as shown in figure 3.4.

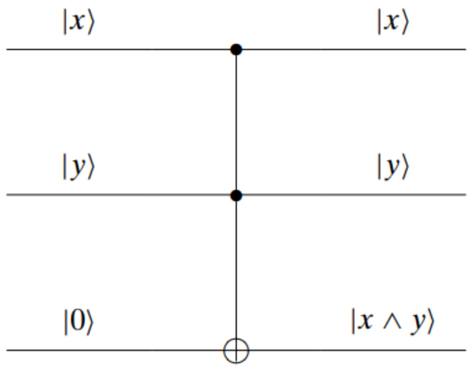


Figure 3.4: This figure shows how to create a classical AND gate using the Toffoli gate. This image is taken from the book "Quantum Computing for Computer Scientists" written by Noson S. Yanofsky and Micro A. Mannucci [94]

Nevertheless, the Toffoli gate is not universal when it comes to quantum computer. In order to make a universal set of gates for a quantum computer, the Hadamard gate needs to also be included in this set [3]. However, this set (Hadamard, Toffoli) is not the only universal quantum computing set. There are many other set of gates that can produce the same universality. For example, combining the rotation operators (R_x, R_y, and R_z) with the phase shift gate (P) and the CNOT gate also produces the same universality.

The rotation and the phase shift operators are special operators the manipulate the phase and the rotation of the qubit when represented by the Bloch sphere representation. The Block sphere is a way of representing qubits by two angles that describe an arrow from the origin of a three dimensional sphere as shown in figure 3.5. These angles are the longitude (ϕ) and the latitude (θ) where $0 \leq \phi \leq 2\pi$ and $0 \leq \theta \leq \pi$. Each point on one of the hemispheres represents a different qubit. The same qubit is mapped twice on both the upper and the lower hemisphere with a phase shift. These qubits can be parameterized using the following mapping:

$$\begin{aligned} x &= \cos\phi \sin 2\theta \\ y &= \sin 2\theta \sin\phi \\ z &= \cos 2\theta \end{aligned} \tag{3.18}$$

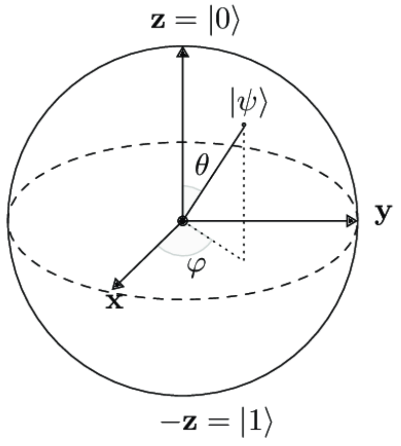


Figure 3.5: This figure shows the Bloch sphere representation of a qubit. This figure is taken from the reference [93]

Therefore, the rotation operators can be defined as operation that would rotate the qubit along one of the dimensions of the Bloch sphere. This rotation can be represented with the following matrices,

$$
\begin{aligned}
R_x(\theta) &= \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \\
R_y(\theta) &= \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \\
R_z(\theta) &= \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}.
\end{aligned}
\tag{3.19}
$$

Whereas, the phase shift operator doesn't modify the state of the qubit, but just shifts it to the other hemisphere of the Bloch sphere. This operator is given by

$$
P(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{\theta} \end{pmatrix}. \tag{3.20}
$$

Circuit Ansatz

The ansatz in the context of variational quantum circuit, is the sequence of gates applied to the different qubits. The ansatz usually refer only to the structure of the circuit without specifying the gate types or their parameters. These parameters are optimized using a variational procedure that will be discussed in the next section.

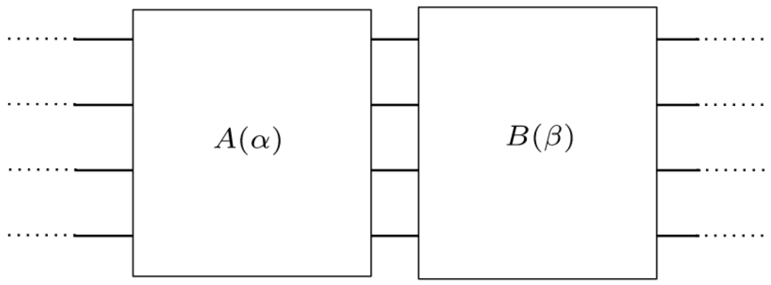


Figure 3.6: This figure shows an example of a layered gate ansatz. The figure is taken from a tutorial by Pennylane [68]

Variational Quantum Eigensolver (VQE)

This algorithm utilizes the quantum variational principle which states that: for a given hermitian matrix H, its ground state energy E_0 is always less than or equal to the expectation value of that matrix calculated with respect to the trail wave function Ψ [84]:

$$
E_0 \leq \langle\Psi|H|\Psi\rangle \tag{3.21}
$$

The offers an upper bound to the ground state energy of a system, and VQE tries to find a parameterization of $|\Psi\rangle$ that would minimize the expectation value. As mentioned before this is a hybrid algorithm, in which the iterator is implemented on a classical computer, while the quantum computer is used to calculate the expectation value of the expression using the current parameters.

This method could also be used to find the structure of the ansatz in an adaptive manner [38].

Limitations of Quantum Computers

Currently, quantum systems suffer from a lot of issues in the form of noise and lack of quantum coherence. This combined with the fact that current systems offer very limited number of qubits, is why the current era of quantum computing is known as the Noisy Intermediate-scale quantum (NISQ) [70] era.

Quantum computers in the NISQ era are struggling to maintain the quantum coherency during the execution of a quantum algorithm. This loss of coherency is known as decoherence, and is usually caused by external factors to the system. Examples of these factors my include vibrations or temperature fluctuation. Solving the problem of coherency is very complicated, this is because quantum computers operate on such a small scale that the slightest change in the outside world can effect these systems.

Fault tolerant systems are still not available and it seems that they will not be available in the foreseeable future. This is why current systems in the NISQ era utilize a hybrid approach to quantum computing. This approach tries to use both classical and quantum computers together to exploit the best advantages of both. Algorithms like the variational quantum eigensolver (VQE) [69] Or the quantum approximate optimization algorithm (QAOA) use this hybrid approach to reduce the amount of calculations done on the quantum computer by implementing some of the more trivial parts on a classical machine. This branch of computation is known as quantum machine learning (QML) and it this allows current quantum systems to be used despite their low fault tolerance. QML has been applied into many scientific areas, most notably the applications it has on chemistry and finance.

3.2 Quantum Chemistry

Quantum chemistry is a branch of chemistry that focuses on applying quantum mechanics to chemical systems. Furthermore, quantum chemistry is concerned with the study of the ground states of atoms and molecules, since these quantum effects on the molecule could determine the molecule's structure and behavior.

Quantum chemistry is an excellent application for quantum computation [22]. This is because quantum chemistry does not require a large number of qubits, and the quantum speedup provided by quantum computers can be extremely beneficial to solve the quantum chemistry problems.

Each molecule is made up of many atoms. The atoms make a specific bond with one another, sharing electrons, resulting in all of the atoms in this molecule reaching a lower energy state than when they started, making them more stable.

Knowing the value of this ground state energy for a given molecule explains the way this molecule react to other molecules, temperature, ...etc. This reaction rate allows researchers and drug developers to know which molecules to use in order to attack specific proteins.

This energy can be calculated using a special observable called the Hamiltonian, which measures the total energy of a given system.

Molecular Hamiltonian

In quantum chemistry, the Molecular Hamiltonian represents the total energy of the electrons and nuclei in the molecule. The general formulation of the Hamiltonian can be given by:

$$\hat{\mathcal{H}} = T + V \tag{3.22}$$

where T is the kinetic energy of the system and V is the potential energy. When this is mapped onto a molecule, this Hamiltonian can be generically expressed as:

$$\hat{\mathcal{H}} = \hat{\mathcal{H}}_e + \hat{\mathcal{H}}_n + \hat{\mathcal{H}}_{en} \tag{3.23}$$

where $\hat{\mathcal{H}}_e$ represents the Hamiltonian of the electrons in the molecule, and $\hat{\mathcal{H}}_n$ represents that of the nuclei. $\hat{\mathcal{H}}_{en}$ represents the Hamiltonian of the interactions between the electrons and nuclei in the system [92].

Nevertheless, the mass of the nuclei is much larger than that of the electrons. This means that the motion of both electrons and nuclei can be separated as the electrons move at a much larger speed making the nuclei seem fixed in space. Therefore, the nuclear motion can be ignored and focus more on the electron motion for a given set of nuclei.

As a result of the previous assumption the wave function of the system can be separated to two. One that governs the movement of the electrons and one that governs the movement of the nuclei. The movement of the nuclei is ignored, provid a mathematical approximation of the molecular dynamics known as the Born-Oppenheimer approximation [91].

Using this approximation the Hamiltonian can be simplified by ignoring the nuclear kinetic energy as the nucleus is considered fixed in space. Moreover, the internal nuclear interactions can also be ignored as it becomes just energy shift between the components of the nuclei without effecting the overall energy of the system. Thus, the Hamiltonian equation becomes:

$$\hat{\mathcal{H}} = -\sum_{i=1}^{N} \frac{1}{2}\nabla_i^2 - \sum_{i=1}^{N}\sum_{A=1}^{M} \frac{Z_A}{r_{iA}} + \sum_{j>i} \frac{1}{r_{ij}} \tag{3.24}$$

where N is the number of electrons in the system, while M is the number of nuclei. The positions of a particle in the three-dimensional Cartesian space is given by r_i, and the distance between two particles is represented as $r_{ij} = |r_i - rj|$. ∇_i^2 is the Laplacian operator, which represents the sum of the second derivatives of the particle with respect to all of its Cartesian components. Finally, Z represents the number of protons in the given nucleus.

The kinetic energy of the electrons in the system is represented by the term:

$$\sum_{i=1}^{N} \frac{1}{2}\nabla_i^2,$$

while the interactions between the nuclei and the electrons (attractive interaction) is calculated in the term:

$$\sum_{i=1}^{N}\sum_{A=1}^{M} \frac{Z_A}{r_{iA}}.$$

Lastly, the interactions of the electrons between each other (repulsive interaction) is calculated by:

$$\sum_{j>i} \frac{1}{r_{ij}}.$$

The Hamiltonian of a system is present in the Schrödinger's equation [17], given by the following equation:

$$\hat{\mathcal{H}} |\Psi\rangle = E |\Psi\rangle \tag{3.25}$$

where ψ represents the wave function of the quantum system and E is the energy value of the system. Therefore, the minimum eigenvalue of the Hamiltonian represents the minimum value that E can take, which represents the ground state energy of that system.

Wave Function

The wave function of the system is one of the key components in solving the Schrödinger's equation. According to the assumptions made by the Born-Oppenheimer approximation, the wave function in the studied molecular systems represents only the electrons in the molecule. This wave function can be viewed as a probability amplitude, which represents the probability density of finding the electron in a certain position in space.

Writing and solving a wave function for a many-electron system is still an extremely complex procedure, even with the Born-Oppenheimer approximation. As a result, to simplify the many-body Schrödinger's equation, the Hatree-Fock approximation [30] was used. This approximation is based on the concept that electrons (or fermions) are indistinguishable particles with identical wave functions. Furthermore, because the particles involved in this wave function are fermions, the Pauli exclusion principle [54] states that two identical particles with an integer half-spin (like fermions) cannot share the same quantum state within the same quantum system. As a result, we have an antisymmetric wave function. This means that the antisymmetrized product of the single-electron wave functions of the electrons in the system can be used to determine the total wave function of the many-electron system. This single-electron wave function is referred to as a "spin orbital." The wave function of two electrons in a system, for example, can be determined using the formula:

$$|\psi_{total}\rangle = \sum_{1 \leq p < q \leq N} \alpha_{pq}(|p\rangle\,|q\rangle - |q\rangle\,|p\rangle), \tag{3.26}$$

and by adding up all the possible permutation of these two particles, the wave function becomes:

$$|\psi_{total}\rangle = \frac{1}{\sqrt{2}}(|p\rangle\,|q\rangle - |q\rangle\,|p\rangle). \tag{3.27}$$

This state represents an anti-symmetric superposition of the two electrons. This anti-symmetry manifests mathematically as a change in the wave function's sign upon swapping the particles in the system.

$$P_{1\to 2} : |\psi(r1, r2)_{total}\rangle = -|\psi(r2, r1)_{total}\rangle$$

Given this anti-symmetrisation the wave function can be written as:

$$|\psi_{total}\rangle = \sum_{\substack{n_1 \cdots n_N \in \{0,1\} \\ \sum_p n_p = M}} \alpha_n |n_1 \cdots n_N\rangle \tag{3.28}$$

where N is the total number of orbitals and M is the total number of electrons in the molecule.

This function then can be defined using the Slater determinant [78], which represents the general wave function of N electrons by this formula:

$$|\psi_{total}\rangle = \frac{1}{\sqrt{N!}} \begin{vmatrix} |X_1\rangle_1 & \cdots & |X_1\rangle_N \\ \vdots & \ddots & \vdots \\ |X_N\rangle_1 & \cdots & |X_N\rangle_N \end{vmatrix} \tag{3.29}$$

each of the N electron's states represent a fermionic orbital.

Quantization

Quantization [15] is the process of translating the current understanding of a phenomena in the classical world, to a new understanding using quantum mechanics. The quantization this project is concerned with is the second quantization of the Canonical quantizations.

The Second Quantization

This quantization [56] uses the creation and annihilation operators to translate the wave functions.The fermion Creation/annihilation operators are used to add/remove a fermion to the single particle state respectively.

The creation operator is denoted by $\hat{\mathbf{a}}^\dagger$, and when applied to a fermionic state it has the following effect on it:

$$\hat{\mathbf{a}}^\dagger |1\rangle = 0$$

$$\hat{\mathbf{a}}^\dagger |0\rangle = |1\rangle$$

On the other hand, the annihilation operator is denoted by $\hat{\mathbf{a}}$, and it has the opposite effects on the fermionic state:

$$\hat{\mathbf{a}}\,|1\rangle = |0\rangle$$

$$\hat{\mathbf{a}}\,|0\rangle = 0$$

These states can also acting on specific orbital in the fermionic state vector. For example, here is the effects of both operators acting on the i^{th} orbital of a fermionic state with N orbitals:

$$\hat{\mathbf{a}}^{\dagger}\,|\cdots n_i \cdots\rangle = (1 - n_i)(-1)^{\sum_{i<j} n_j}\,|\cdots (n_i + 1)\cdots\rangle$$

$$\hat{\mathbf{a}}\,|\cdots n_i \cdots\rangle = (n_i)(-1)^{\sum_{i<j} n_j}\,|\cdots (n_i - 1)\cdots\rangle$$

the term $(-1)e^{\sum_{i<j} n_j}$ in both represents the phase factor. This is added to satisfy the Pauli exclusion principle and maintain the anti-symmetrization of the wave function. For instance, if the order in which these operators act on the fermionic state switches then the this change should be compensated by a switch in the sign of the wave function:

$$\hat{\mathbf{a}}_i\,\hat{\mathbf{a}}_j^{\dagger}\,|\psi\rangle = -\,\hat{\mathbf{a}}_j^{\dagger}\,\hat{\mathbf{a}}_i\,|\psi\rangle$$

Moreover, as the fermionic system satisfies the Pauli exclusion principle, a creation operator can act only once on an empty orbitals. This insures that no two fermions exist within the same orbital simultaneously.

The fermionic system being studied is represented in the Fock space [23]. Which is a sub-space of the Hilbert space that is defined for a variable number of identical particles (like fermion or bosons). Moreover, all operators in the Fock space could be written as a combination of creation/annihilation operators. Furthermore, the molecular Hamiltonian's wave function became solely concerned with fermionic interactions, after using the Born-Oppenheimer approximation. This wave function can therefore be represented in Fock space, and its Hamiltonian is a Fock space operator. As a result, the Hamiltonian can be rewritten as a set of creation and annihilation operators.

Re-Writing the Hamiltonian

The Hamiltonian consists of two one-body operators that operate on a single electrons, and one two-body operator that acts on two electrons. after re-writing the operators as creation and annihilation operators:

$$\hat{\mathcal{H}} = -\sum_{i,j} \frac{1}{2} \langle i| \nabla_i^2 |j\rangle \, \hat{\mathbf{a}}_i^\dagger \, \hat{\mathbf{a}}_j + \sum_{i,j} \langle i| \frac{Z_A}{r_{iA}} |j\rangle \, \hat{\mathbf{a}}_i^\dagger \, \hat{\mathbf{a}}_j + \sum_{i,j,k,m} \langle i,j| \frac{1}{r_{ij}} |k,m\rangle \, \hat{\mathbf{a}}_i^\dagger \, \hat{\mathbf{a}}_j^\dagger \, \hat{\mathbf{a}}_k \, \hat{\mathbf{a}}_m \tag{3.30}$$

after simplifying the previous expression, the Hamiltonian could be re-written as:

$$\hat{\mathcal{H}} = \sum_{pq} h_{pq} \, \hat{\mathbf{a}}_p^\dagger \, \hat{\mathbf{a}}_q + \sum_{pqrs} h_{pqrs} \, \hat{\mathbf{a}}_p^\dagger \, \hat{\mathbf{a}}_q^\dagger \, \hat{\mathbf{a}}_r \, \hat{\mathbf{a}}_s \tag{3.31}$$

where h_{pq} represents the one-body integral, and h_{pqrs} represents the two-body integral. Nevertheless, This Hamiltonian can be further simplified if only the electrons belonging to the active space of the molecule were considered.

Orbital Active Space

The active space is the space residing between the high and low level orbitals. This space is especially compelling because it accounts for the majority of molecular energy. This is because the high level orbitals are typically empty, whereas the electrons in the low energy orbitals have low levels of energy and do not move around much. As a result, the electrons in the active zone are those that are constantly moving and hence represent the molecule's overall energy.

Qubit Mapping

in order to map the Hamiltonian into a Hilbert space using qubits two steps need to be carried out. First, the mapping used to translate the quantum state from the Slater determinant representation to the qubit state must be defined. Then, The fermionic algebra defined on the Fock space needs to be translated to fit the qubit representation.

Jordan–Wigner transformation

Jordan–Wigner transformation [58] defines a direct (1:1) mapping into the qubit representation, where each orbital can be mapped into a qubit. This implies that N qubits will be required to map a Fock space with N fermionic orbitals.

The Jordan–Wigner also defines the following transformation to map the fermionic algebra into the qubits representation:

$$\hat{\mathbf{a}}_i^\dagger \rightarrow \prod_{i<j} \sigma_j^z \sigma_i^+$$
$$\hat{\mathbf{a}}_i \rightarrow \prod_{i<j} \sigma_j^z \sigma_i^-$$

where:

$$\sigma_j^+ = \frac{\sigma_j^x + i\sigma_j^y}{2}$$
$$\sigma_j^- = \frac{\sigma_j^x - i\sigma_j^y}{2}$$

and $\sigma^x, \sigma^y, \sigma^z$ are the Pauli operators.

However, there is a significant flaw in this mapping where the number of Pauli operators grows linearly for each creation/annihilation operator. Given the current state of the quantum technology this mapping will be very demanding, and very hard to implement.

Bravyi-Kitaev transform

The Bravyi-Kitaev mapping [20] defines a mapping that only requires $O(\log N)$ qubits to map N fermions. This mapping defines three sets that it acts on [74]:

The first set is the Parity set $P(j)$. For an creation or annihilation operator acting on orbital j; this set calculates which qubits tell us if the state of the system needs to acquire a phase change of -1.

The second set is the update set $U(j)$. This set defines the set of qubits (different than the qubit representing orbital j) that need to change when the orbital j is updated.

The final set this mapping defines is the flip set $F(j)$. This set determines the set of qubits that will determine if the qubit in position j has the same or the inverted parity with respect to the orbital j.

Given these three sets, the the mapping of the creation and annihilation operators to Pauli operators can be represented as follow:

$$\begin{aligned} \hat{\mathbf{a}}_j^\dagger &= \frac{1}{2}(X_{U(j)}X_j Z_{P(j)} - iX_{u(i)}Y_j Z_{\rho(j)}) \\ \hat{\mathbf{a}}_j &= \frac{1}{2}(X_{U(j)}X_j Z_{P(j)} + iX_{u(i)}Y_j Z_{\rho(j)}) \end{aligned} \tag{3.32}$$

where given a set of qubits S and a Pauli operator σ^x, then the notation X_S is a shorthand for the σ^x operator acting on all the qubits in the set S. This same notation applies for all the Pauli operators, where Z represent the Pauli operator σ^x and Y represents σ^y.

This mapping is the one used throughout the project as it is the one that saves the largest amount of qubits. This is very important as the current quantum systems suffer from a qubit shortage.

3.3 Artificial Neural Networks

Artificial neural networks (ANN) [12], also known as neural networks (NN), are information processing systems modeled after the nervous system. This system is made up of numerous linked neurons that send information to one another. Similarly, ANNs simulate these networks so that a machine may learn and make decisions in a human-like manner. These networks form the basis for machine deep learning (DL) as they allow the machine to be able to extract the patterns from a training data set without human help.

ANNs are composed of a large number of interconnected nodes (representing the human neurons) working together to solve a specific problem. ANNs can be trained to solve a specific problem like classifying, or pattern recognition. This training process is example based, and the network is automatically modified by the machine to be able to solve this problem.

The explanation provided in this section is based on a book called "Neural Networks and Deep Learning: A Textbook" by Charu C.Aggarwal [2].

Artificial Neurons

The basic building block of an ANN is known as a node or neuron. These nodes receive a input signals then, after processing them, it produces an output signal as shown in figure 3.7. The inputs of a neuron are weighted, thus not all the input signals have the same influence. These input signals are combined together taking their weights into consideration,

$$\sum_{i=0}^{N} X_i w_i + b, \tag{3.33}$$

where N is the number of input signals, X_i is the input signal in position i, and w_i is the signal's corresponding weight. Finally, b is the learning bias of the node. The output of this summation is normally fed into an activation function that determines the neuron's output.

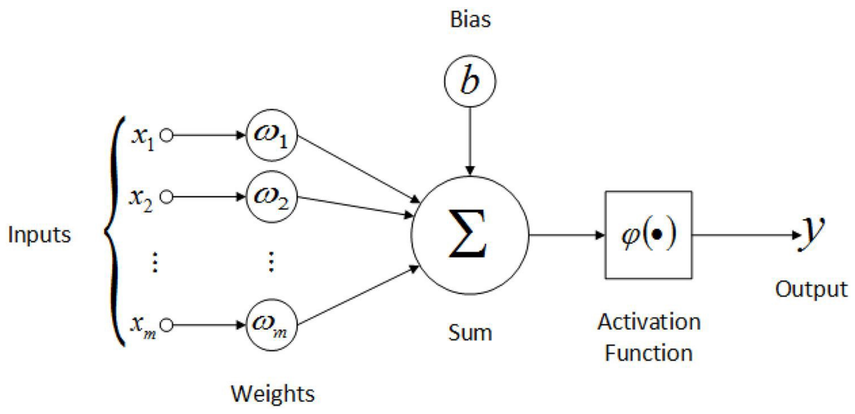


Figure 3.7: This figure shows the structure of an artificial neuron used in ANNs. This image is taken from a Medium post by Ricardo Mendoza [62]

Activation Functions

Activation functions take in the weighted sum of the input signal and determine the output of the neuron. There are many activation functions and each serve a specific purpose, and choosing one depends on the problem that the network is trying to solve [2]. The most basic activation used in neural networks is the linear activation,

$$\Phi(v) = v, \tag{3.34}$$

where Φ denotes the activation function, and v is the weighted sum of the neuron's inputs. This activation is mostly used when the output is expected to be a real number.

On the other hand, if the expected output is expected to have a value between $[0, 1]$, then a *Sigmoid* activation can be used. *Sigmoid* (shown in figure 3.8 image (c)) outputs values between 0 and 1, which is very helpful when the expected output is a probability. This activation is defined by the following equation:

$$\Phi(v) = \frac{1}{1 + e^{-v}}. \tag{3.35}$$

If the expected output is a binary value (0 or 1) then a *sign* activation can be used . This activation (shown in figure 3.8 image (b)) defines a cut off at 0, where all the values below 0 are mapped to -1 and all values above 0 are mapped to one as defined in equation 3.36.

$$\Phi(v) = Sign(v) = \begin{cases} 0, & \text{if } x \leq 0 \\ 1, & \text{if } x > 0 \end{cases} \tag{3.36}$$

Moreover, if the expected output is between $(-1, 1)$ then a tanh activation would be helpful. This activation (shown in figure 3.8 image (d)) is very useful when there is a desired distinguishability between positive and negative values in the output. This activation is given by the following equation:

$$\Phi(v) = \frac{e^{2v} - 1}{e^{2v} + 1}. \tag{3.37}$$

A modified version of this activation function known as Hard tanh (shown in figure 3.8 image (f)) have mostly replaced the soft tanh activation due to the ease it provides in training the network. Moreover, the Rectified Linear Unit (ReLU) (shown in figure 3.8 image (e)) has proven to be one of the most powerful activations in modern neural networks. This activation is known for its simplicity, speed, and faster convergence rates. The ReLU activation discards all the negative values by replacing them with a 0, while not changing the positive values.

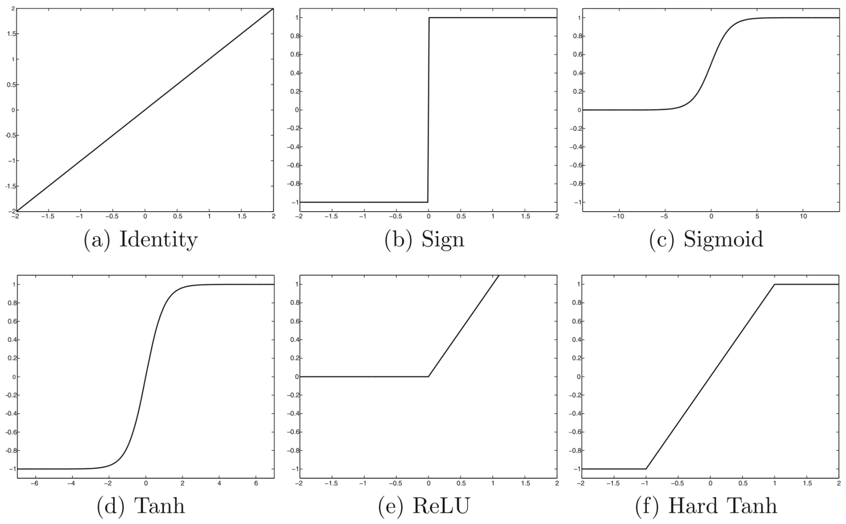


Figure 3.8: This figure shows the graphs of various activation functions. This image is take from the book "Neural Networks and Deep Learning: A Textbook" [2]

Loss Functions

The loss function measures the error in the output, which indicates the amount of change that needs to happen to the network's weights and biases. Similar to activation functions, the loss function is also problem dependent [2]. For instance, for numeric outputs a simple squared error function can be used that looks like $(y - \hat{y})^2$ for every training instance, where y is the expected output and $\hat{y}$ is the predicted output. Moreover, if the numerical output is between $[-1, 1]$ then a hinge loss can be used. This loss is defined by

$$Loss = max\{0, 1 - y\hat{y}\} \tag{3.38}$$

On the other hand, for probabilistic predictions it depends on if the target is a binary prediction or a categorical prediction. If it is a logistic regression problem, then it would have a binary prediction and the following activation can be used:

$$Loss = \log(1 + \exp(-y\hat{y})) \tag{3.39}$$

If the output is a categorical probability, thus there are multiple output classes and each contain the probability of the output being that class. The loss function in this case is cross-entropy loss and it is given by

$$Loss = -\log(\hat{y}_r) \tag{3.40}$$

where r is the index of the correct class. There are many different types of activation and loss functions, however, choosing one depends on the other and on the nature of the output. Moreover, such decision will change depending on the problem at hand as this will determine the nature of the output, the values this output can take, and the type of network to be used.

Multilayer Neural Networks

A single-layer neural network has an input layer where the data from the input is collected, and an output layer where the input information is processed and a decision is made. Note that the input layer doesn't count as a layer in neural networks as no computation happens in this layer. However, this network is fairly simple as it only performs a weighted some to the input data and applies an activation function to that. More complexity can be added to this network by adding more intermediate layers between the input and the output layer. The multi-layer neural network is known as a feed-forward network, as the output of every layer is used as the input to the consecutive layer. The user of such network can only see the results of the output layer, all the intermediate calculations and results are hidden from the end user. This is why these layers are known as the hidden layers.

The architecture of a multi-layer network is defined by the number of hidden layers it has, and the number of neurons each layer has. Another important factor to consider in the network design is the loss function to be used, and the activation functions used in each layer.

Training a Multilayer Neural Network

Training a neural network works by providing the network with a set of examples and then letting the network learn from the errors it makes. This process consists of two main steps. First, giving the network the examples and letting the network guess the answer. Then, the error is calculated and the weights and biases across the network are modified accordingly. This process is repeated enough times until the network converges and the patterns in the input data are learned.

The first step in the learning process is known as the forward propagation step. In this step the input is passed down the layers of the network. Each layer calculates the weighted sum of the input it was provided and then applies the activation function to that sum. The output of this operation is then passed on to the consecutive layer as the new input. Finally when this reaches the output layer, the values produced by that layer represent the output of that network. This prediction produced by the output layer, is then compared to the expected result in the case of supervised learning. The error in this prediction is calculated according to the loss function defined in the network.

This error is then pushed backwards through the layers in a step known as back-propagation. One of the most popular tools used to calculate this back-propagation is the gradient decent. In each layer, the gradient of the loss function with respect to the weights is calculated using the chain rule of derivatives. then this gradient is used to updates the weights and biases in that layer. The gradient decent will help minimize the loss function, thus get the network one step closer to convergence.

Practical Issues in Neural Network Training

The previously outlined training process does not always go smoothly. This could be due to a variety of factors, particularly two are discussed in this section. The most common of which is overfitting. This problem occurs when a model produces good results when tested with the training dataset, but does not guarantee high performance when provided with new data. In other words, the model's training and testing data performance are vastly different. The most common cause of this issue is when the model is highly complex and there is very little data available to train it. As a result, instead of learning the patterns in the training data, the model memorizes it. There are several approaches to this problem, such as early stopping or adding noise to the training data, but the solution will ultimately rely on the network and the data available.

The other recurrent problem with networks with a very large depth is the vanishing and exploding gradient problem. This problem happens due to the chain-rule applied when calculating the gradient decent across the network in the back-propagation stage. In some cases, the gradient of the earlier layers becomes negligibly small by the time the updates reach the later layers. In other situations, however, these updates can become increasingly large, causing the learning process to be disrupted. In order to solve this issue, a change of the activation functions used in the hidden layers may help. Moreover, other approaches like adaptive learning or batch normalization can also be useful in this case.

Recurrent Neural Networks

Recurrent neural network (RNN) is a neural architecture that is designed to deal with sequential data. This is particularly useful when the input size is not known, and can not be truncated or padded. One of the most common uses of RNNs is when dealing with textual data. This due to the nature of text, as the size of every input is very different, and sometimes it is not possible to truncate as valuable information can be lost. Moreover, the sequential data structure provides RNNs a very important advantage, as it allows for a positional encoding of data. In other words, the position (aka. timestamp) at which the data point is provided, plays a crucial role in its influence on the output in this type of networks.

The basic architecture of such network consist of a single node with a feedback connection as shown in figure 3.9. The weights of the node are shared between all timestamps, but the network passes on a state vector between the timestamps allowing the network to form a general understanding of the input data. This network can be unfolded to show the evolution of the network over time (as done in figure 3.9). The output of this network depends on the problem at hand. Generally, every step produces an output vector. These vectors can accumulated to form a sequence as the output, or simply the last value returned by the network could be used.

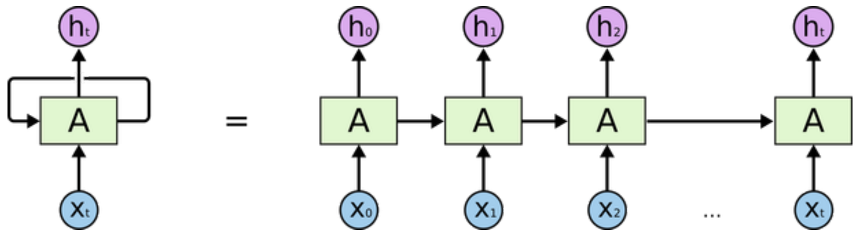


Figure 3.9: This figure shows the architecture of a simple RNN. Moreover, it shows how this single node unfolds over time, showing the same unit but over every time step. This figure is taken from a post by Vivek Singh Bawa [14].

Long Short Term Memory

Long Short Term Memory (LSTM) [44] is a form of RNN that seeks to keep extra long term context in addition to the short term context that RNNs generally retain. This method attempts to address the problem of vanishing signals for extended sequences that is common in RNN networks.

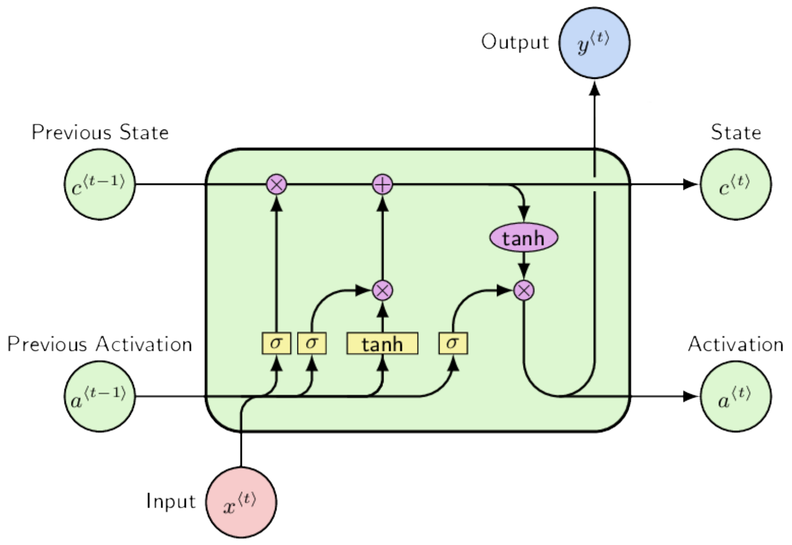


Figure 3.10: This is a figure illustrating an LSTM cell and the flow of the input and state vectors through the cell and is taken from reference [34]. The figure shows the different gates that the input passes through and the different changes that happen to the state vector, showing how the output is formed.

LSTM defines a set of gate units and memory cells (shown in figure 3.10) in order to allow a consistent error flow through the units. There are three main gates in an LSTM cell, the forget gate, the update gate, and the output gate.

The forget Gate is responsible of deciding which information should pass through the network and which information should be dropped. The forget gate at a given timestamp is denoted by f_t and calculated using the following equation:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{3.41}$$

where t is the current timestamp, x is the input at the current timestamp, and h is the output from the previous timestamp. Furthermore, W_f denotes the weight of the current timestamp's input, while U_f is the recurrent weight of the previous timestamp's output, and b_f is a bias weight.

Similarly, the update gate is used to determine which information presented in the current input should be used to update the information in the current cell. The update gate has a similar formulation to the forget gate, except that it uses different weights, as shown in the following equation:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{3.42}$$

where the update gate has three main weights: the input's weight at the current timestamp W_i, the recurrent output weight at the previous timestamp U_i, and finally the bias b_i.

Nevertheless, LSTM doesn't only rely on the update gates to decide what information from the current input will be kept, but it also adds another layer to decide which candidate values from the input could be added to the state vector. this layer is given by :

$$\tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C) \tag{3.43}$$

the most important difference between this layer and the other two gates discussed earlier is that this layer uses a tanh as its activation preserving the sign of the input. This step uses different weights to calculate the output. However, the structure of the weights is similar to that of the previous two gates.

These three components are used to update the state cell C that stores the long term memory of the network. The state cell at the current timestamp t is given by:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.44}$$

Finally, in order to compute the output of the LSTM node, the output gate is used. This gate has a similar structure to the forget and update gates, and it's given by the following formulation:

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{3.45}$$

this gates is responsible of deciding which parts of the input are going to be passed throw to the output in the current timestamp. Furthermore, the output is calculated by the following formula:

$$h_t = o_t * \tanh C_t \tag{3.46}$$

this output is constructed using both the output gate and the current state vector.

In the back-propagation step of this neural network, the gradient of every gate needs to be calculated in order to be able to know how to update the weights and biases of the network. First, the derivatives of all the activation functions used by the gates need to be calculated. LSTM uses two activation functions, $sigmoid$ and tanh. The $sigmoid$'s derivative is given by the following equation:

$$\frac{d}{dx}\sigma(x) = \frac{d}{dx}\frac{1}{1+e^{-x}} = \frac{1}{1+e^{-x}}(1 - \frac{1}{1+e^{-x}}) = \sigma(x)(1-\sigma(x)), \tag{3.47}$$

where x denotes the input provided to the $sigmoid$ function. Similarly, the derivative of the tanh can be calculated, and it is given by,

$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x). \tag{3.48}$$

Knowing the derivative of the activation functions, then the derivatives of the gates can be calculated using the chain rule represented in equation 3.49.

$$\frac{d}{dx}[f(g(x))] = f'(g(x)).g'(x) \tag{3.49}$$

Finding the gradient of each timestamp requires calculating the gradient of the gates in the reversed order in which the they were calculated in the feed-forward step. Furthermore, this method must begin with the result and work backwards until it reaches the earliest timestamp. To compute the gradient of the LSTM node at a random timestamp t, first the gradient of the node's output at that timestamp needs to be calculated.

The gradient of the node's output at a given timestamp t is calculated as such,

$$\delta h_t = \Delta_t + \Delta h_t, \tag{3.50}$$

where Δ_t is the accumulated output difference as calculated by the subsequent layers, and Δh_t the output difference of the current node. After that the gradient of the LSTM gates can be calculated:

$$\begin{aligned}
\delta o_t &= \delta h_t \odot \tanh(C_t) \odot o_t \odot (1 - o_t) \\
\delta C_t &= \delta h_t \odot o_t \odot (1 - \tanh^2(C_t)) + \delta C_{t+1} \odot f_{t+1} \\
\delta f_t &= \delta C_t \odot C_{t-1} \odot f_t \odot (1 - f_t) \\
\delta \tilde{C}_t &= \delta C_t \odot i_t \odot (1 - \tilde{C}_t^2) \\
\delta i_t &= \delta C_t \odot \tilde{C}_{t-1} \odot i_t \odot (1 - i_t)
\end{aligned} \tag{3.51}$$

Once the gate's gradient is calculated, then they can be used to update the weights. The gradient of the current timestamp's weight W is calculated using the gradient of the gates and the input of the node at the timestamp as shown in equation 3.52.

$$\delta W = \delta f_t \otimes x_t + \delta i_t \otimes x_t + \delta o_t \otimes x_t + \delta \tilde{C}_t \otimes x_t \tag{3.52}$$

The recurrent weight is calculated using the gradient of the gates in timestamp $t + 1$ and the output of the current timestamp.

$$\delta U = \delta f_{t+1} \otimes h_t + \delta i_{t+1} \otimes h_t + \delta o_{t+1} \otimes h_t + \delta \tilde{C}_{t+1} \otimes h_t \tag{3.53}$$

The bias of the current timestamp is updated using the gradient of the timestamp $t + 1$.

$$\delta b = \delta f_{t+1} + \delta i_{t+1} + \delta o_{t+1} + \delta \tilde{C}_{t+1} \tag{3.54}$$

The input's gradient is calculated using the current timestamp's weights and the gradient of the current timestamp's gates. This to propagate the gradient to other nodes preceding the LSTM node.

$$\delta x = \delta f_t W^T + \delta i_t W^T + \delta o_t W^T + \delta \tilde{C}_t W^T \tag{3.55}$$

Finally, the gradient of the previous timestamp's output is calculated using the gradient of the current timestamp's gates alongside the current timestamp's recurrent weights.

$$\Delta h_{t-1} = \delta f_t U^T + \delta i_t U^T + \delta o_t U^T + \delta \tilde{C}_t U^T \tag{3.56}$$

Adam Optimizer

Gradient descent usually has a single constant learning rate that is used to update all the different weights. This is not very efficient as the learning rate plays a crucial role in the convergence of the network. If the learning rate is big, then the network most likely will never converge as the gradient descent will keep on fluctuating. On the other hand, if the learning rate is very small, then the network convergence process can be very slow. This can possibly cause some models not to converge because the gradient descent was not able to reach a minimum during the execution time.

In order to avoid this issue, adaptive learning rate methods are used. One of the most well known method is the adaptive moment estimation (Adam [51]) method is used. Adam overcomes this problem by maintaining a per-parameter learning rate that change based on the changes in the gradients.

The algorithm calculates the exponential averages of the recent gradients of a specific weight, along side the squared gradient. Moreover, the two parameters β_1 and β_2 are used to control the decay rates of these averages. These averages are estimates of the mean value, and the variance of the gradient. The parameters β_1 and β_2 are usually values between 0 and 1, where 1 is excluded. Normally, it is recommended that these values are initialized to 0.9.

The pseudo code for the algorithm as stated in the original paper can be seen in figure 3.11. As seen in this code, Adam calculates the first moment estimate m_t (representing the mean) and the second raw moment estimate v_t (representing the uncentered variance). These terms are computed as follows:

$$\begin{aligned} m_t &= \beta_1 m_{t_1} + (1 - \beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \end{aligned} \tag{3.57}$$

where t is the current timestamp, and g_t is the gradient at the current timestamp. Moreover, the g_t^2 indicates an element wise square ($g_t \odot g_t$).

Nevertheless, in case the parameters β_1 and β_2 are initialized as zero vectors, then the moment estimates are going to be bias towards zero in the first steps. This is why a bias correction step is done to ensure these effects are not present. This is done as follows:

$$\hat{m}_t = \frac{m_t}{(1-\beta_1^t)}$$
$$\hat{v}_t = \frac{v_t}{(1-\beta_2^t)} \tag{3.58}$$

Finally the corrected moment estimates are used to calculate the changes to the weight (θ) as follows:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t}}, \tag{3.59}$$

where α is the learning rate.

```
**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0,1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t/(1-\beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t/(1-\beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure 3.11: This figure shows the pseudo code for the Adam algorithm. This code is taken from the paper "ADAM: A Method For Stochastic Optimization" [51]

3.4 Software Tools

There are many software tools used to complete this project. Mainly, these tools can be split into three categories: The general environment tools, the tools used for the quantum simulations, and the tools used to build the machine learning models.

General Environment Tools

The programming language chosen for this project was Python [87]. This is because of the variety of packages and resources available for this language for both quantum computation and machine learning.

This project also relied on using Jupyter notebooks [53] as the way to analyze and test the results. Jupyter notebook is an open-source software that offers a web-based computing platform.

Quantum Computing Tools

In order to perform all the quantum computing simulations required by the project, the python library Pennylane [16], which is a library used to program quantum computers. This library provides different modules to utilize quantum computing in different areas. For instance, Pennylane can be used to build quantum machine learning (QML) models or perform quantum chemistry simulations.

This project used the quantum chemistry module (qchem) of Pennylane. This module is the one used to simulate chemical systems on quantum computers. This model contains a Hatree-Fock solver, that can be combined with VQE to find the ground state energy of a molecule. Building the molecular Hamiltonian formulation of a molecule in Pennylane, uses the openfermion-pyscf plug in.

OpenFermion [60] is an open source library to compile and optimize quantum algorithms that simulate fermionic systems. the openfermion-pyscf interfaces the PySCF [80] package through the OpenFermion library. PySCF is an electronic structure package that provides a lot of useful information about the electronic structure of different molecules.

Machine learning Tools

The machine learning model in this project was built from scratch, this is why libraries like Tensorflow or PyTourch were not used.

Originally, the Keras library[25] from Tensorflow was used to build the machine learning model. Nevertheless, the dependency on this library was later dropped.

SciPy [88] is another library that was used to perform some prepossessing on the data in the early stages of the project. The main use of this library was to split the data into training and testing datasets.

A lot of the mathematical calculations relied on the operations offered by Spacy [45] and NumPy [40].

Spacy is a machine learning library that offers a module for using sparse matrices. These matrices were the used by this project to store the weights and the biases of the network.

NumPy is a library for multi-dimensional arrays and matrices. Moreover, this library provides a wide collection of optimized mathematical functions to operate on these arrays.

These two libraries were used to build the LSTM neural network. The storage was done using the Spacy sparse matrices, while the computations were done using the NumPy module.

In order to plot the results obtained by this model, the library Matplotlib [46] was used. This library offers powerful tools for plotting graphs.

Chapter 4

PROPOSAL

The primary goal of this project is to develop a method for predicting the ground state energy of the molecular Hamiltonian without having to solve it. This project tried to solve the problem by training an LSTM neural network on a set of molecular Hamiltonians produced by a quantum computer.

Current quantum computers are not capable of solving large Hamiltonian equations due to the number of qubits limitation current systems suffer from. On the other hand, classical computers lack the computational power to efficiently solve these full Hamiltonian equations. This is why this project tries to use hybrid method to try and find a solution to the problem.

The main idea this project proposes is to predict the output of the Hamiltonian's quantum circuit instead of finding the exact value. This idea was inspired by a blog post by Pavan Vadapalli [85] that uses RNNs to predict the output of simple math equations instead of solving them. Similar to the human intuition to approximate the answer of a simple math equation without solving it, computers can learn to visualize the math equation and approximate its answer.

This concept was also further explored by some researchers from Meta (formally known as Facebook) [24]. In their research they explored the possibility of solving first and second level differential equations using machine learning. The results they obtained were remarkable and proved that machine learning can be utilized not only to make predictions but in some cases it can also be trained to provide very accurate results. Based on this hypothesis, the project tries to teach a machine learning model to predict the outcome of the quantum circuit without having to execute it.

4.1 Molecular Hamiltonian Circuit

The first step in this project was to be able to obtain the ansatz for a given molecular Hamiltonian. Moreover, This ansatz has to be solved to obtain the training and testing data needed to train the model.

There are various libraries that give tools to create such an ansatz if the molecular structure is provided. Google, for example, created the OpenFermion Library [60], an open source library used to model fermionic systems. Furthermore, Pennylane constructed their own quantum chemistry module called QChem using some of the features in the OpenFermion package. Because it is a module developed on top of OpenFermion, QChem offers more simple and intuitive functions than the ones offered by OpenFermion. Additionally, Pennylane supplied more simple examples with highly extensive step-by-step tutorials to assist new users in becoming acquainted with the platform. As a result, the project ended up employing Pennylane's Qchem module to generate and find the ground state of the Hamiltonian.

Defining a molecular structure is required to create an ansatz. This means that all of the atoms in that molecule, as well as their positions in three-dimensional Cartesian space, must be defined. Moreover, The active zone of that molecule can be defined providing the number of active electrons and active orbitals. This will decrease the number of electrons considered for the calculation of Hamiltonian, thus making the calculation faster and the ansatz smaller.

It's also necessary to define the spin multiplicity, charge, basis, and mapping transformation. The charge is the molecule's net charge. The spin multiplicity is given by the equation $N_{unpaired}+1$, where $N_{unpaired}$ is the number of unpaired electrons in the HF space. The basis is the atomic basis that is used to depict molecular orbitals; for this project, the basis is mainly 'STO-3g.' This is one of the most popular basis, and it constructs 3 Gaussian function to approximate the atomic orbitals representation [77].

Finally, the mapping is the procedure for converting a fermionic Hamiltonian to a qubit Hamiltonian. Throughout the project, the Bravyi Kitaev mapping was chosen because it is the mapping that uses the fewest qubits. Once the molecule's structure has been determined, the Hamiltonian ansatz is utilized to compute it.

In order to build a quantum circuit to calculate the ground state energy of a molecule a tutorial provided by Pennylane [82] was followed. In this tutorial, they demonstrated how to design an adaptive quantum circuit to calculate the molecule's ground state energy and then solve it using the variational quantum eigensolver (VQE) [69].

Typically, a pre-selected wave function ansatz is used to construct the circuit, which includes all potential single and double excitations from the occupied spin-orbitals. This approach delivers great generalization, but it suffers from a performance disadvantage as incorporating all conceivable excitations often increases simulation costs without enhancing the accuracy of the results. That's why, in the tutorial, they create the circuit for each molecule using an adaptive variational algorithm [38].

The gates are chosen by computing the gradients for all possible excitation gates for each molecule. Then, selecting the gates depending on the magnitude of the gradients. Moreover, usually a cut off is defined to decide if the gates needs to be included or not.

The first step needed to calculate the gradients is to construct the Hamiltonian and calculate all its possible excitations. This can be achieved by using some functions from Pennylane's Qchem library. In order to construct the Hamiltonian of a molecule, The function $molecular\_hamiltonian()$ can be used. This function requires the molecular structure in order to generate the Hamiltonian ansatz. After computing the molecular Hamiltonian it would be possible to obtain the single and double excitations. Each one of these excitations is linked to a gate that excites electrons from the occupied orbitals into the unoccupied ones.

Then, it's necessary to calculate their gradient in order to select which gates stay in the final circuit. The first step is to calculate the gradients for the double excitations, and select the excitations that are grater than the pre-determined threshold. Once the excitations have been selected it's possible to perform VQE on the selected gates to obtain the optimized parameters for these set of excitations. After that, the same process is repeated for single excitations. Finally, perform a final VQE optimization on the selected gates added to the final circuit which will provide the ground state energy after executing it.

Nevertheless, having only one Hamiltonian circuit per molecule will not provide sufficient data to train a model. This is a consequence of the limited number of molecules that can run on current simulators with limited number of qubits. To overcome this limitation, for each molecule the atoms of that molecule were moved around the Cartesian space providing different Hamiltonian expressions for the same molecule.

There will always be one and only one Hamiltonian representing the ground state energy of the molecule. This Hamiltonian usually requires the atoms of the molecule to exist within a specific distant and form a specific bound. Nevertheless, for the sake of training the model it is not necessary to use the Hamiltonian representing the ground state of the molecule as the only thing that is needed is a dataset of Hamiltonian circuits with their corresponding energy. This is because the model is trained to predict the output of the Hamiltonian circuit, not to predict the ground state energy of the molecule.

4.2 Implementation

This project was re-implemented three different times. Each time the results of the previous implementation were used to construct the new approach.

Initial Approaches

The initial approach was set to use the Hamiltonian circuit as a string. This string was then encoded character by character using a predefined dictionary. This dictionary mapped each character that can appear in the string representing the Hamiltonian circuit into a specific vector. The encoding is represented in table 4.1. This encoded circuit was then used to train an LSTM neural network. An LSTM neural network was chosen because of its capacity to preserve both short and long term memory in the input sequences it is given. This would provide a significant benefit in terms of being able to identify patterns that connect the Hamiltonian to its corresponding energy.

Character	Encoding
’0’	0
’1’	1
’2’	2
’3’	3
’4’	4
’5’	5
’6’	6
’7’	7
’8’	8
’9’	9
’+’	10
’-’	11
’*’	12
’/’	13
’I’	14
’X’	15
’Y’	16
’Z’	17
’(’	18
’)’	19
’[’	20
’]’	21
’e’	22
’.’	23

Table 4.1: This table shows the encoding used in the char-by-char encoding proposed by the first approach of the project

This encoding was not only applied to the inputs of the network, but it was also applied to encode the outputs. The outputs of the network were encoded into vector with 22 positions. This meant that some of the output’s decimals had to be cropped so that the length of the output string would be 22 characters. The number 22 was chosen as it provided enough precision for the various energy values in the training dataset.

The LSTM network in this approach was built using the Keras library provided by TensorFlow [25]. The model consisted of 4 layers (as shown in figure 4.1); the first layer is an LSTM layer. This layer takes in an input that has the shape of *(batch size, timestamps, 24)* where the 24 is the number of features in the encoding vector for every character. The batch size is the number of input vectors used to train the network in each training batch.

The LSTM network then returns an output of shape *(batch size, 128)* where 128 is the number of intermediate units. This vector is then passed through a Repeat Vector layer, which duplicates the vector 22 times providing an output of shape *(batch size, 22, 128)*. The output is then feed again into another LSTM layer. This layer is instructed to return the full sequence which is why the output's shape persists.

Finally, the output is passed through a Time Distributed layer, which is built with a softmax activation. The Time Distributed layer applies a two-dimensional convolution to each of the timestamps in the input. This convolution will transform the 128 units into 24 which is the number of features in the encoded vector. The output of this layer had the shape *(batch size, 22, 24)*. The 22 timestamps represented the characters of the output string. Each of these position is a 24 vector storing the encoding of that character.

The fundamental issue with this approach is that the model learned no patterns from the input sequences. This is because the input was very random once it was converted by the char-by-char encoding. This encoding did not present any simple patterns that could be learned with the model's limited dataset. Moreover, the output of this model was not always a valid float. This is mainly because the output is generated as sequence of encoded characters instead of it being just a float.

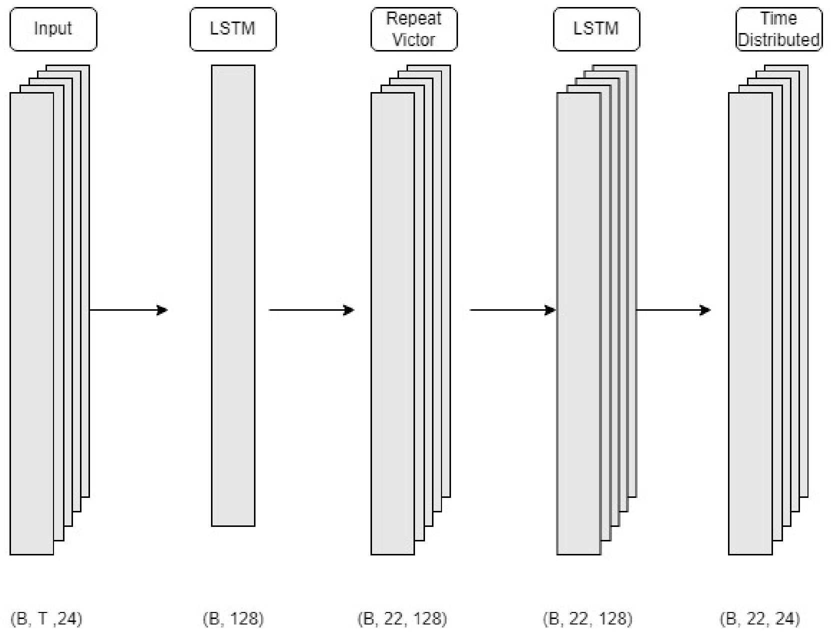


Figure 4.1: this figure shows the structure of the model used in the first approach of this project. In the figure, B represents the batch size, and T represents the timestamps. The first shown layer is the input of the model which had the shape of (batch size, timestamps, 24). The last layer is the output layer which had a shape of (batch size, 22 ,24) this output represents the predicted string which has a length of 22 characters. Each one of these characters is represented as an encoded vector with 24 features.

Second Approach

The second approach tried to avoid the mistakes of the first approach and use numbers for both inputs and outputs of the model instead of encoding them as strings. In order to find a numerical input to the model a lot of options were examined. As the input needed to represent the Hamiltonian as much as possible while maintaining general patterns allowing the model to learn from it.

Using only the coefficients of the gates in the Hamiltonian circuit was considered. But this idea was quickly discarded as these coefficients are not enough to represent the Hamiltonian. Because by stripping the circuit from the Pauli operators, then the representation of the Hamiltonian using the second quantization is lost.

This investigation led to the use of the double-body excitation matrix. This matrix represents the most complex part of the Hamiltonian, and it provides a lot of patterns due to its structure.

The double-body matrix is a 4-dimensional matrix where the number of positions in each dimension depends on the molecule and the size of its active zone. The shape of this matrix is (N, N, N, N) where N is the number of qubits needed to represent the Hamiltonian of that molecule. This matrix can then flattened to a two dimensional matrix with the shape (N^4, 1). This two dimentional matrix can then be fed to the LSTM network. For example, the Oxygen molecule O_2, which has 4 active electrons, has a matrix with the shape of (20, 20, 20, 20). Once this matrix is flattened the shape becomes (160000, 1).

To construct this matrix, the molecule's molecular data must first be extracted. The OpenFermion package from Google is used for this. The two body tensor from the molecular Hamiltonian can be obtained using the information supplied by OpenFermion. Nonetheless, in order to obtain molecular data, the OpenFermion library requires an HF (Hartree-Fock) file, and the Pennylane library's $meanfield()$ method can be used to generate this file.

Once the file has been produced, it is supplied to OpenFermion's $MolecularData()$ function. This function returns an object from which the molecular Hamiltonian of that molecule can be obtained by calling the $get\_molecular\_hamiltonian()$ function. Finally, the Hamiltonian returned has a field called two_body_tensor that contains the two-body excitation matrix information.

After attempting to train a model using this matrix as the input of the LSTM model depicted in figure 4.1, it was discovered that this matrix cannot be used to train a Keras or PyTourch model.

The main reason for this was driven by the matrix's massive size. This is a problem since there was no sufficient and accessible hardware available to train the model without running out of memory. Moreover, these libraries do not fully support sparse matrices (especially for LSTM), which would have helped in overcoming the memory issues.

However, examining the data in the two body matrices revealed that the matrices contained a large number of 0s. This led to the notion of representing these matrices using a sparse matrix, which will save a lot of resources. Therefore, if an LSTM model can use sparse matrices, then the training would be feasible on the hardware available. To test this theory, a custom sparse LSTM network had to be built from scratch.

SciPy's sparse matrices were chosen to create this network. This is due to SciPy's numerous functions and applications that make working with these matrices easier.

Furthermore, they were far more user-friendly and adaptable than other options.

This network was first implemented as a custom Keras layer, which would allow it to be integrated with the rest of the Keras layers in order to generate the model. However, Keras requires the use of Tensorflow tensors instead of SciPy's sparse matrices. This implied that if SciPy's sparse matrices were employed, Keras will not be able to calculate the gradient descent of the network automatically. Moreover, the sparse TensorFlow tensors did not offer all functionalities offered by the SciPy sparse matrices. Additionally, these sparse tensors were very hard to configure and work with. Due to these limitations, there were no significant advantages to adopting this Keras costume layer, as it just increased restrictions and provided no significant gains.

The second implementation of the LSTM network did not rely on TensorFlow or PyTourch; instead, it was built totally from scratch.

The first step took to build the LSTM network from scratch was to understand the math that is used to calculate the forward and the backward propagation in each LSTM node. The information for the forward propagation was obtained by reading Christopher Olah's blog [66] on understanding LSTM Networks. However, understanding the back-propagation was a harder challenge as it required finding the derivatives of all the equations needed to calculate the LSTM output and then using them to find the gradient decent that will later be used to adjust the weights. there was a lot of helpful information regarding the derivatives of the equations used in the LSTM nodes on both Augstinus Kristiadi [55] and Adian Gomez [35] web blogs.

Nevertheless, after implementing the first draft of the code using the information on the web blogs mentioned before, the output of the LSTM network did not match the shape of the training output. This is because the LSTM network has an output with the shape (batch size, number of intermediate units), whereas the training output was just a float. This resulted in the need to add a dense layer after each node to compress the output of that node.

The initial implementation of the Dense layer involved two steps. The first is to apply a weight and a bias to the input of the Dense layer. Then, the output of the layer consists of the average of all the elements of the weighted input vector. The Input vector was a 2 dimensional matrix. This is why the average needed to be calculated in order to reduce the dimensionality of this vector.

In order to avoid using a loop to calculate the average, the dot product of the weighted input vector with an all ones vector of an appropriate shape was calculated. The ones vector had the transposed shape of the input vector, which meant that the output of the dot product was the sum of all the elements in the weighted input vector. This number was then divided by the length of the input vector to find the average. The equations bellow show the calculations done in the Dense layer:

$$Dense = x_{dense}W_{dense} + b_{dense} \tag{4.1}$$

$$output = \frac{Dense_{ij}[1]_{ji}}{i} \tag{4.2}$$

where i and j are the number of rows and columns in the Dense matrix respectively.

Nonetheless, this approach proved to be incorrect. The main problem was with the input structure and how this LSTM handled it. Normally, the training input of the LSTM has the shape *(size of the batch, number of timestamps, number of features)*, and then for every input in the batch the timestamps are processed sequentially by passing them one after the other through the LSTM node.

However, with the first implementations of the LSTM network there was a misunderstanding regarding how the input is processed. As a consequence, instead of passing the input in the manner described before, each input in the patch was passed as a vector to the LSTM node, including all timestamps (see figure 4.2). Thus, the input was a matrix instead of being a vector. Furthermore, due to the way the weights are multiplied this added dimensionality was propagated down to the output making it very difficult to densify the output matrix into a single float.

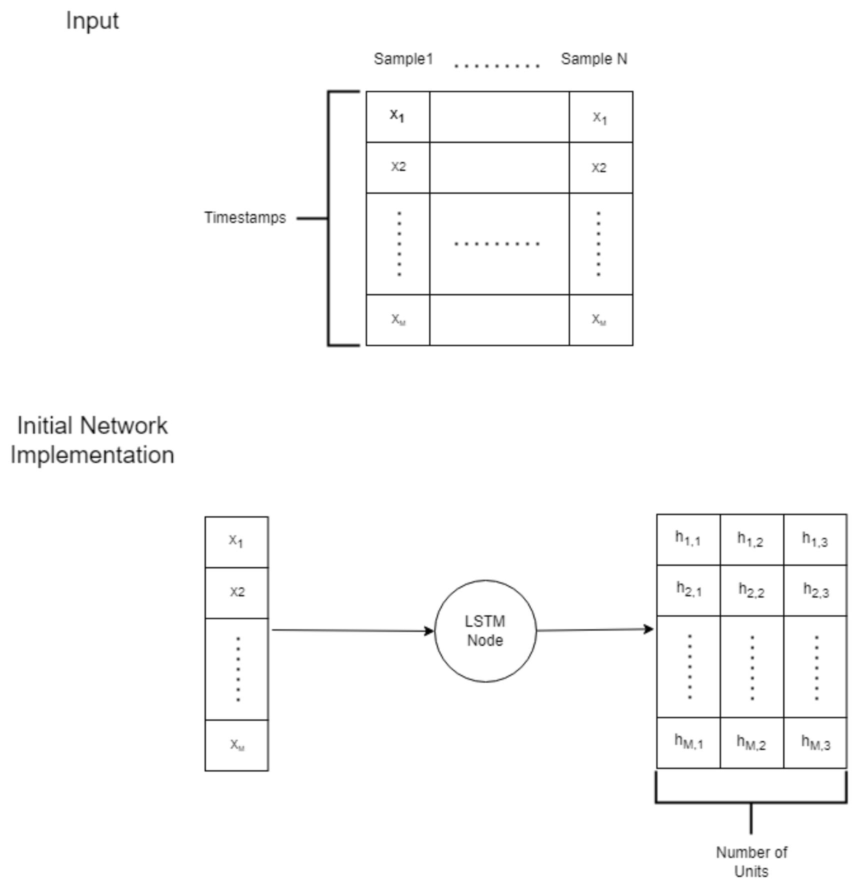

Input
Sample1 .... Sample N
Timestamps
x_1
x2
x_M
Initial Network Implementation
LSTM Node
$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{M,1}$ $h_{M,2}$ $h_{M,3}$
Number of Units

Figure 4.2: this figure shows the initial way the input was entered to the LSTM node, where the timestamps where all entered at once instead of one at a time resulting in the output being too large to compress.

This increase in the dimensionality implied an increase on the complexity of the model. Which made the model very complex to be trained with the available dataset. The output of the LSTM node was almost constant for all the inputs.

After fixing the input of the network, the dense layer was modified as there was no longer need to calculate the average of the values as the network returned a single vector not a matrix. The new dense layer is given by the equation below:

$$output = x_{dense}W_{dense} + b_{dense} \tag{4.3}$$

However, after modifying the input and the dense layer, the network still didn't work. This was because the output vector h_t was turning into all ones very quickly, thus, making the output of the network be a slight modification of the dense layer's weight.

The main reason for this was that the weights of the LSTM node were initialized only with positive values between [0,1).

After multiplying these weights with the input and the output of the previous nodes; the output was almost always positive, thus, when applied to a $Sigmoid$ activation function the output was always positive. This effected the forget gate, as it never forced the network to forget any of the values stored in the state vectors, leading the vectors to overflow with information. To fix this issue a normal distribution centered around 0 and with a standard deviation of 1 was used to initialize all the weights of the LSTM network.

After modifying all the weights of the network and running the first tests, the time to pass each input through the network was very large. For instance, the input of the O_2 molecule contained 160000 values that needed to be processed, and it took over 12 hours to train a model with 100 O_2 molecules. In order to optimise this temporal cost, the zero values in the input sequence were ignored, and only the non zero values were passed through the network. This presented a noticeable improvement on the time it takes to pass one molecule through the network. Moreover, this improvement was propagated to the backward step as well making the training process much faster.

Nevertheless, after training a simple model with a collection of molecules, the results of this approach were not very good. The model consisted of an LSTM layer followed by a dense layer, and it continuously got over-fitted. Moreover, the time it took to test and train the model was still very large even after reducing the size of the input. This temporal limitation rendered the model not scalable. As a result, the way the input was represented needed to change again to be able to create a model that is able to scale for larger molecules.

Final Approach

The final approach shifted the focus back to the Hamiltonian string instead of the two body excitation matrix. The core idea was to find a new way to represent the values of the coefficients and which qubits and gates they affect.

The way this is achieved is by encoding every term, which is a combination of a coefficient and a Pauli operator, into a vector with the shape (3, 1). Each position in this vector represents a Pauli operator. The encoding of the Pauli operators in the vector is $[X, Y, Z]$, where the X gate is encoded in the first position of the vector while the Z gate is the last.

Then for each qubit in the Hamiltonian string all the gates acting on that qubit were grouped together in the order they appear in the Hamiltonian string. Moreover, most molecular Hamiltonian start with an offset term that is multiplied with I gate acting on qubit 0, this offset is not considered in this encoding. However the value of the coefficient was also subtracted from the final energy to compensate for the removal of the operator.

For example, consider the following Hamiltonian,

$$\hat{\mathcal{H}} = -12.32[I_0] + (-0.23)[Z_1X_2Y_0] + (-0.45)[X_0Z_1Y_2]. \tag{4.4}$$

The process of encoding this Hamiltonian will start by subtracting the I Pauli operator's coefficient from the Hamiltonian $\hat{\mathcal{H}}$ final energy value. Then every term will be translated into a vector and ordered according the qubit it acts on. For instance, qubit 0 has two terms acting on it in this Hamiltonian circuit ($(-0.23)[Y_0]$ and $(-0.45)[X_0]$). This means that these terms are translated and placed at the begging of the sequence as shown below:

$$(-0.23)[Y_0] + (-0.45)[X_0] \rightarrow \begin{bmatrix} 0.0 \\ -0.23 \\ 0.0 \end{bmatrix} \begin{bmatrix} -0.45 \\ 0.0 \\ 0.0 \end{bmatrix}$$

The the encoding for the second qubits is calculated. This qubit also has two Pauli operators acting on it in this circuit ($(-0.23)[Z_1]$ and $(-0.45)[Z_1]$). Resulting in the following mapping:

$$(-0.23)[Z_1] + (-0.45)[Z_1] \rightarrow \begin{bmatrix} 0.0 \\ 0.0 \\ -0.23 \end{bmatrix} \begin{bmatrix} 0.0 \\ 0.0 \\ -0.45 \end{bmatrix}$$

Finally, the third qubit has another two Pauli operators acting on it ($(-0.23)[X_2]$ and $(-0.45)[Y_2]$). This translates to:

$$(-0.23)[X_2] + (-0.45)[Y_2] \rightarrow \begin{bmatrix} -0.23 \\ 0.0 \\ 0.0 \end{bmatrix} \begin{bmatrix} 0.0 \\ -0.45 \\ 0.0 \end{bmatrix}$$

After combining all these mappings in order, the resulting sequence would be:

$$\begin{bmatrix}0.0\\-0.23\\0.0\end{bmatrix}\begin{bmatrix}-0.45\\0.0\\0.0\end{bmatrix}\begin{bmatrix}0.0\\0.0\\-0.23\end{bmatrix}\begin{bmatrix}0.0\\0.0\\-0.45\end{bmatrix}\begin{bmatrix}-0.23\\0.0\\0.0\end{bmatrix}\begin{bmatrix}0.0\\-0.45\\0.0\end{bmatrix}$$

This process is repeated for all input values, and each one of the generated sequences is then fed into the same LSTM network from the previous approach. The shape of the input in this approach is (3,1) as each input is a 3 dimensional vector. Moreover, the length of the sequences in this approach is much shorter compared to the sequences generated by the previous approach.

For instance, the average length of the sequences in this new approach is around 950, while the average length of the sequences in the old approach was about 160K. This increased the speed at which the model trains and increases the scalability of the solution. Because memory and computation power needed to process and store the new sequences is much lower than that required to do the computations in the old approach.

Finally, to fix a lot of the convergence issues and over fitting issues that the previous two approaches had, this approach added an Adam optimizer to the LSTM network. This optimizer had to be built from scratch to be compatible with the sparse LSTM network build in this project. The equations and pseudo code described in section 3.3 where used as a reference when the algorithm was implemented.

Chapter 5

RESULT ANALYSIS

5.1 The setup of the testing environment

In order to test this project a series of tests were carried out for each one of the approaches. In the early stages of the project the main focus was on being able to find a way to calculate the ground state energy for any given molecule. This was done using the quantum computer simulator provided by Pennylane and QChem libraries. Moreover, it was interesting to see how moving the atoms around the 3-dimensional Cartesian space affected the ground state energy and the Hamiltonian formulation.

In order to test this, the molecular structure of each molecule was studied, and the atoms of that molecule were moved around in a way that would maintain the molecular bound. For example, The water molecule (H_2O) has two Hydrogen atoms and one Oxygen atom. The Hydrogen atoms are located to both sides of the Oxygen. These atoms form a bound between them with an angle about 104.5° [90].

In order to maintain the structure of the molecular bound, only the distance between the different atoms was modified. Both Hydrogen atoms were moved around the space in a trajectory that would maintain the 104.5° angle they form with the Oxygen atom.

The z-axis of these molecules was fixed to 0 for the ease of calculation. Moreover, the Oxygen atom was also fixed in space at coordinate (0,0) in the now 2 dimensional molecular space. Then, the Hydrogen atoms were initially placed on both sides of the oxygen in positions that would create the 104.5° angle between the three atoms.

After placing the initial positions of the atoms, each Hydrogen atom was moved around the space along the line passing through the Hydrogen atom and the Oxygen atom. These two lines formed ensured that the angle between these three atoms would be maintained as the Hydrogen atoms moved around the space.

Assuming the lines are symmetrical with respect to the Y-axis, then the angle between either one of the lines and the Y-axis would be 52.25°. The positions of both lines and the atoms is illustrated in figure 5.1

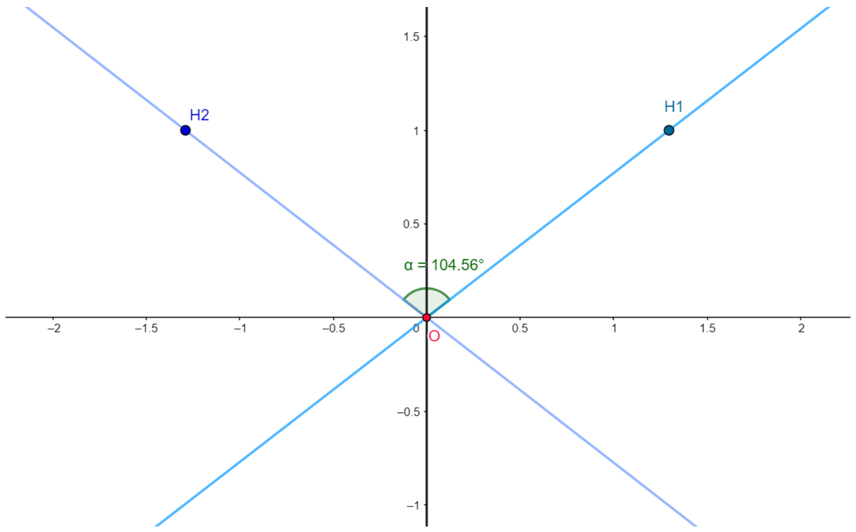


Figure 5.1: This figure illustrates the position of the two lines where the Hydrogen atoms (H1, H2) will move with respect to the Oxygen atom (O).

In order to find one of the equations of either one of the lines, then the coordinates of either H1 or H2 needs to be known (where H1 and H2 represent the Hydrogen atoms of the water molecule). To find these coordinates, a horizontal line is drawn with the equation $y = 1$, forming two right triangles between each one of the lines and the Y-axis.

Given that the line y=1 forms a right triangle (as shown in figure 5.2) with the line connecting H1 and O, then the coordinates of H1 can be known given the distance between H1 and the Y-axis. This distance can be found by looking at the sine of the angle between the Hydrogen line and the Y-axis (l).

$$\sin(52.25°) = \frac{l}{1} \tag{5.1}$$

$$\therefore l \approx 1.25, \tag{5.2}$$

meaning that the line passing through H1 and O will pass through the point (1.25, 1). Therefore, the line has the equation: $y = 0.77x$.

Using the symmetry discussed earlier, the line passing through H2 and O will have the equation $y = -0.77x$.

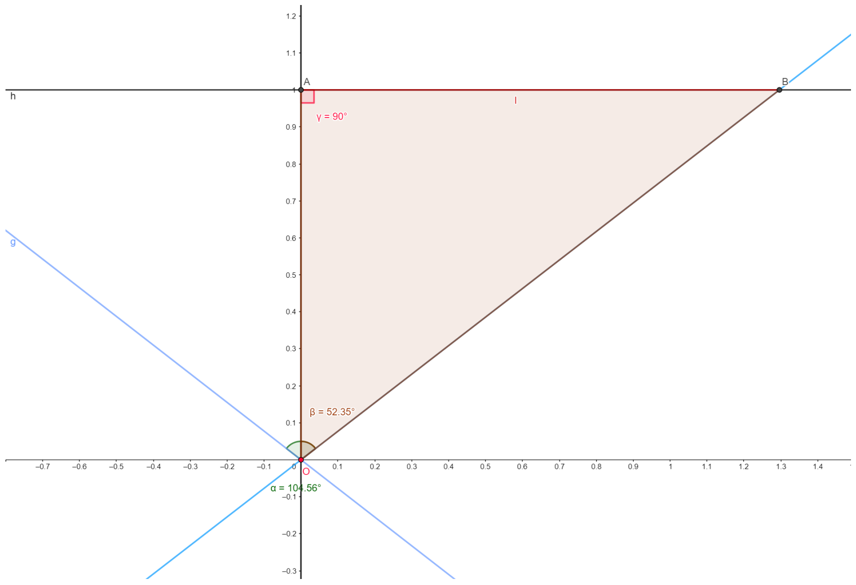


Figure 5.2: this figure shows the triangle formed by the line y=1, the line that passes through the first Hydrogen atom (H1) and the Oxygen atom (O), and the Y-axis

Knowing the lines that Hydrogen atoms need to follow in order to maintain the bound angle, multiple ground state energies were calculated. For instance, a different Hamiltonian can be formed for every combination of atomic coordinates as shown in figure 5.3. All these combinations with their energies were calculated and stored in files for later use. The movement of the atoms will result in an energy landscape (shown in figure 5.4) where it shows the change of the molecular ground state energy with relation to the distance between the molecule's atoms.

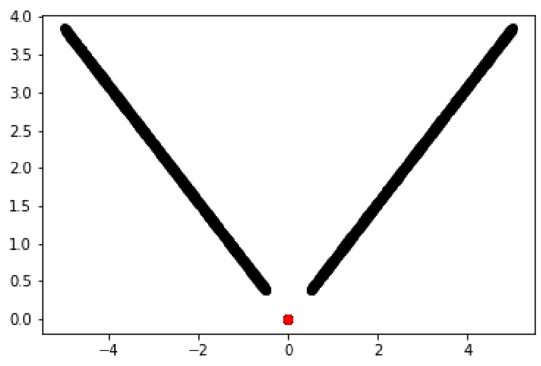


Figure 5.3: this figure shows the movement of the Hydrogen atoms with respect to the Oxygen atom

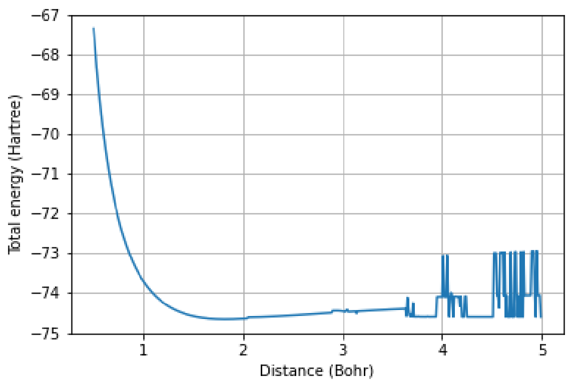


Figure 5.4: this graph shows the energy of the water molecule with respect to the change in distance between the Hydrogen atoms and the central Oxygen atom

5.2 Result Analysis

The previous process was repeated to generate the data for all the molecules used to build the dataset.

The Results of The Initial Approach

The information in this dataset was then used to train the initial model that used the Hamiltonian string with a char-by-char encoding as the input. The performance of this model was very bad overall. For instance, the average of the accuracy on the training data was 32% as shown in figure 5.1. Moreover, when this model was tested on the testing dataset it did not always return a valid float, as it sometimes returned values like "1..6666082033999". Furthermore, the model suffered a lot from over-fitting considering the limited size of the training dataset.

Dataset Size	Avg Training Accuracy
6563	0.34
578	0.32
342	0.31

Table 5.1: This table shows the training accuracy of different datasets trained with the initial model that used the Hamiltonian as string.

These findings are caused by the encoding of the input and output data. For example, after translating the Hamiltonian string into vector encoding, the input data is fairly random. Each of the Hamiltonian strings is quite long, especially after being converted to a char-by-char encoding. When the numbers and gates from the original Hamiltonian term are translated, the model misses a lot of patterns.

Furthermore, because the output was similarly encoded in this char-by-char encoding, it was not guaranteed that the result would always be a valid float. Because the model predicts the result char-by-char, some of the outputs included numerous floating points.

The Results of The Second Approach

The testing results began to improve after pivoting the method to use the double-body matrix as the input. The model's over-fitting, however, persisted. Another difference in this new approach is that instead of using the Keras LSTM, it employed a sparse LSTM that was created from scratch.

When testing the new approach, the training loss was decreasing as seen in Figure 5.5. Nevertheless, the model was very influenced by the last input it received in the training process and constantly returned the same output in the testing stage.

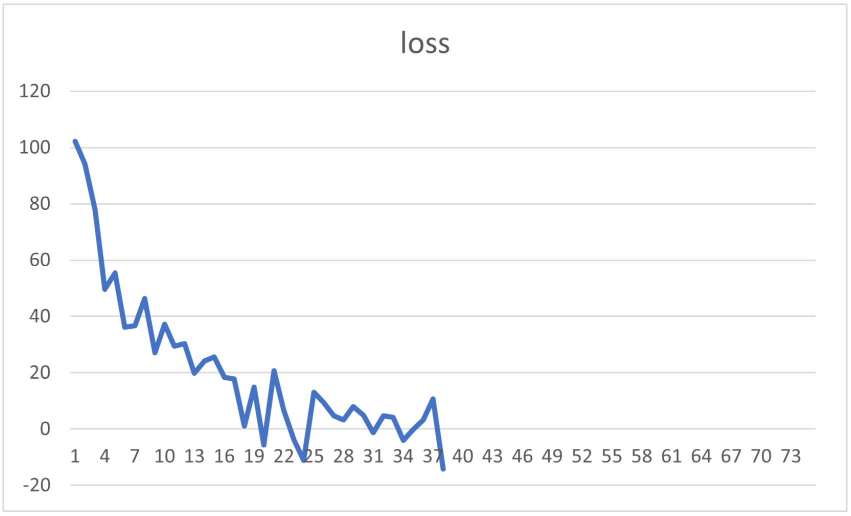


Figure 5.5: This figure shows the evolution of the average loss when training the Sparse LSTM with the double body matrix. This training was done with the molecules O_2, BeO, and FLi with a total of 150 data points.

The training data used in this attempt consisted of about 5 molecules (FLi, H_2, HO^-, O_2, and BeO) that were small enough so that their Hamiltonian can be calculated on a quantum simulator locally. Many combination of the previous molecules were used to train the model and test it. However, the model always seemed to suffer from over-fitting. This issue was originally due to some issues with the sparse LSTM. Even though fixing these issues slightly helped with the over-fitting, the model was still not learning properly. The Model was usually influenced by the last molecule it was trained with. For instance, if the model was lastly trained with HO^- the energies predicted by the model will be in the range of -70; whereas, if the last molecule was O_2 the predicted energies will be in a higher range of around -100.

For example, compare the results of predicting the BeO energies when the model was trained with FLi, O_2, and BeO (as shown in figure 5.6) to the same model after the HO^- molecule was added to the training dataset (as shown in figure 5.7). The model's projections shifted closer to the HO^-'s energy level, which is around -70.

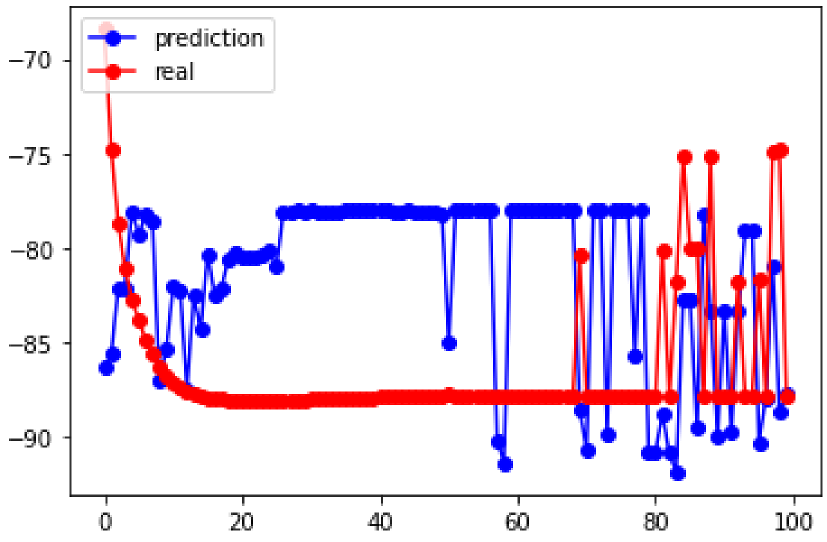


Figure 5.6: This figure shows the testing results of a model trained with FLi, O_2, and BeO molecules, and tested on data it has previously seen from the BeO molecule.

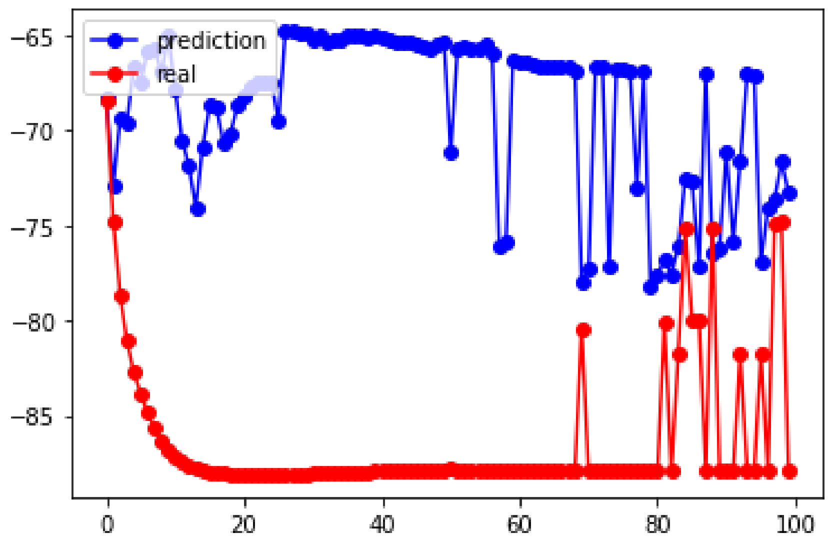


Figure 5.7: This figure shows the testing results of a model trained with FLi, O_2, BeO, and HO^- molecules, and tested on data it has previously seen from the BeO molecule.

Shuffling the dataset and updating the model's weights initialization helped boost the generalizability of this model a little. Furthermore, as seen in figure 5.8, the model's loss seemed to converge better than before. The predicted outputs, on the other hand, were only slightly different and usually circled around a single number, which was still heavily influenced by the training data.

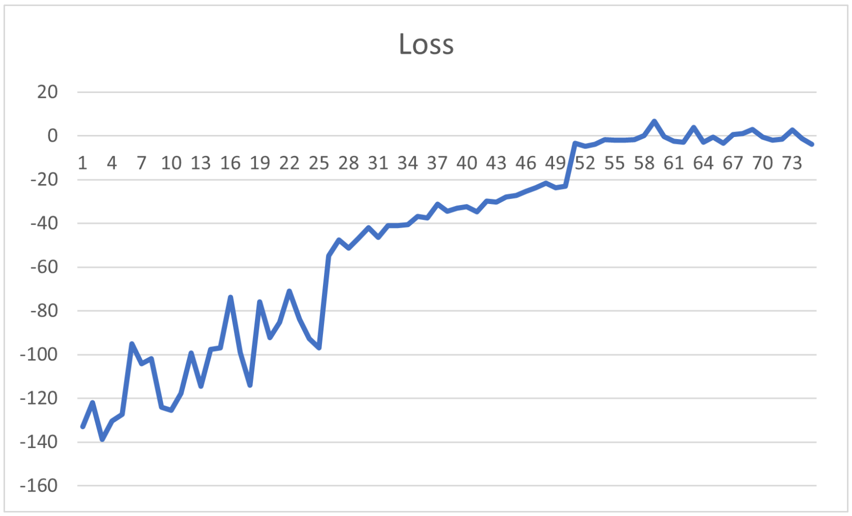


Figure 5.8: This figure shows the evolution of the average loss when training the Sparse LSTM with the double body matrix. This training consisted of 4 epochs, each with 25 batches of size 4

Nevertheless, after examining the results of many tests, it became clear that a lack of data was not the only problem preventing the model from performing better. The ground state energy of distinct molecules varies significantly, and even if the model were given more training data, the model's generalization will always be limited to values that it has seen before. For example, if the model was trained with values between -0.5 and -100, predicting the ground state energy of a molecule with a ground state energy of -200 will be difficult. To address this problem, a logarithmic scale was applied to the ground state energies during training, bringing the energy values closer together. This approach combined with the shuffled data improved the results of the model as shown in figure 5.9.

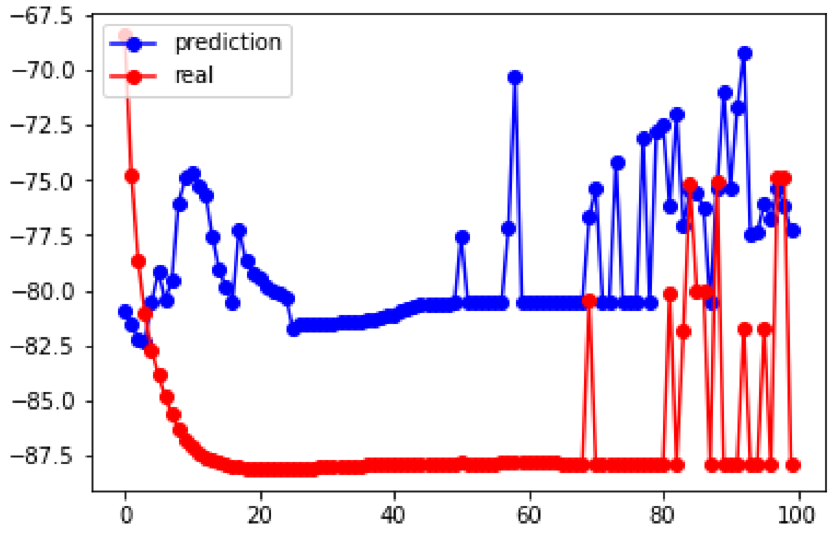


Figure 5.9: This figure shows the testing results of a model trained with FLi, O_2, BeO, and HO^- molecules, and tested on data it has previously seen from the BeO molecule. After applying the log_{10} transformation to the ground state energies during the training phase.

Although applying the logarithmic transformation to the model's output helped with the generalization problem, there were other drawbacks that made this approach unscalable. This problem arises as a result of the input matrix's structure. In fact, the size of the flattened double body matrices increases exponentially as the number of electrons and the active zone size increase. This is an issue for the project since it implies that it won't be able to deal with larger molecules.

The Results of The Final Approach

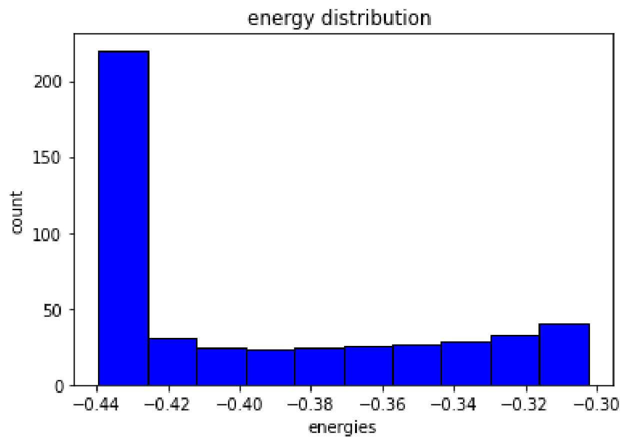
This earlier analysis led to the project's final approach, which is to return to the Hamiltonian string. However, instead of encoding the weights char-by-char, the string is encoded in a different method this time, allowing the weights to be represented as integers. By incorporating some of the concepts from the second technique, this solves some of the limitations of the original strategy.

This method was broken down into three phases. Make the Hamiltonian strings first, as described earlier in this section. Then, term by term, encode the string. Finally, train the same LSTM model used in the second approach using the new encoded strings. The primary goal of this approach's training was to understand the patterns in the quantum circuit that represented the Hamiltonian.

Furthermore, omitting the single body interactions from the Hamiltonian and concentrating just on the double body interactions provided a significant benefit. It not only improved the model's generalizability, but it also minimized the magnitude of the output energies as shown in figure 5.10. This allows various molecules with different energy levels to have similar outputs that the model can learn without the need of using any logarithmic or other transformations.

This first tests of this approach were done on data the model was trained with. For instance, after training the model with O_2, LiH, HO^-, FLi, and H_2; it was tested on O_2. The results of this experiment were very promising as shown in figure 5.11. The model seemed to have learned the data it was trained with, keeping in mind that the model was only trained for 2 epochs with around 2K training data point. Then to confirm that this model was not over-fitted and can perform good on data it has not been trained with, the O_2 molecule was removed from the training dataset. After that, the model was tested again with the O_2 molecule. The results of this experiment can be seen in figure 5.12. The results were somewhere similar to the first experiment which shows that the model was not memorizing the input data, but rather was able to extract patterns in the encoded Hamiltonian strings allowing it to have an overall good generalization with data it has not seen before.

This test was repeated after adding the C_2 molecule to the training, and the testing was done on O_2 which was not present in the training dataset. This model was only trained for one epoch, and the results can be seen in figure 5.13. AS seen by this results, the model is very sensitive still to training data. This is a side effect for the lack of data, meaning that possibly with larger amount of training data, higher accuracy can be archived.



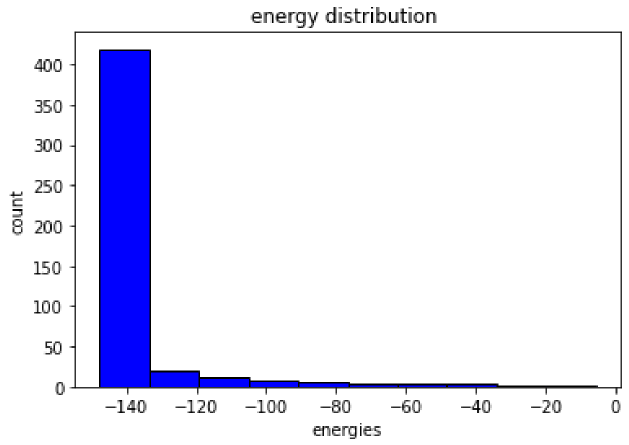


Figure 5.10: This figure shows the difference in the training data after removing the one-body interactions and concentrating on the double-body interactions. Removing the energy generated by the one-body interactions reduces the energy into a smaller interval. The graph on top shows the energies without the one-body interactions, while the graph on the bottom shows the energies with the one-body interactions.

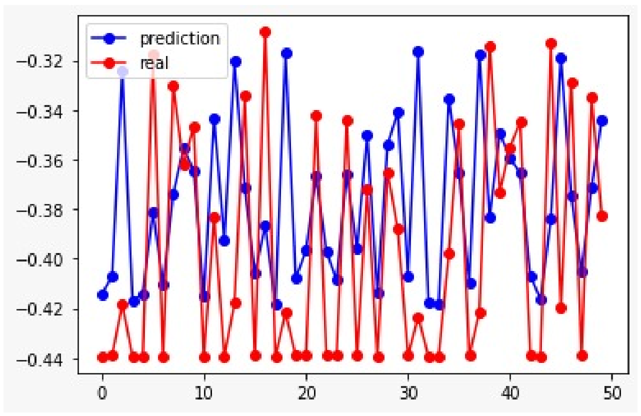


Figure 5.11: This figure shows the testing results when trying to predict the energies of the O_2 molecule that has been already seen in the training data. The model was trained with the newly encoded Hamiltonian string using the molecules O_2, LiH, HO^-, FLi, and H_2.

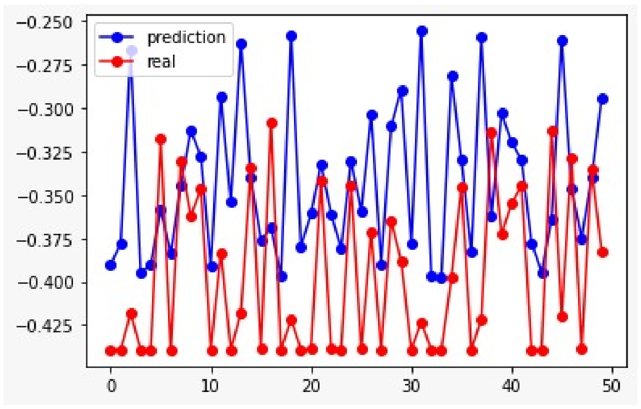


Figure 5.12: This figure shows the testing results when trying to predict the energies of the O_2 molecule without including it in the training dataset. The model was trained with the newly encoded Hamiltonian string using the molecules H_2, LiH, HO^-, FLi

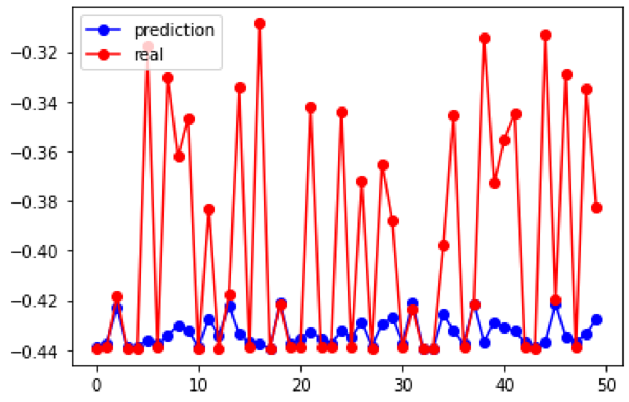


Figure 5.13: This figure shows the testing results when trying to predict the energies of the O_2 molecule without including it in the training dataset. The model was trained with the newly encoded Hamiltonian string using the molecules C_2, H_2, LiH, HO^-, FLi

5.3 Discussion

After seeing the results of the tests with the Oxygen molecule, the experiment was repeated with the Hydrogen molecule to ensure the reliability of the model. The results of this experiment can be seen in figure 5.14. The overall performance of the model seemed to be very promising as the results of the previous two experiments show.

However, the model has only been trained and tested on molecules with similar complexity, as all the training molecules consisted of two atoms with a linear bound

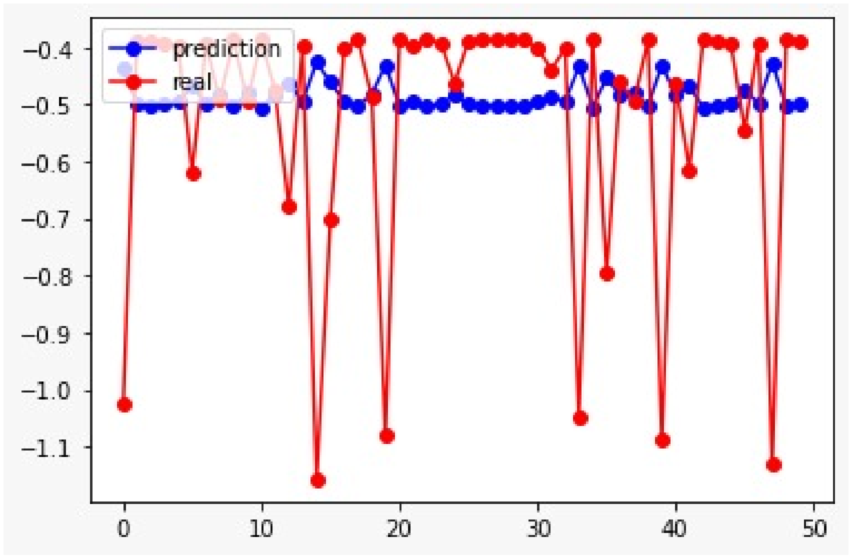


Figure 5.14: This figure shows the testing results when trying to predict the energies of the H_2 molecule without including it in the training dataset. The model was trained with the newly encoded Hamiltonian string using the molecules O_2, LiH, HO^-, FLi

and a somewhere similar number of active electrons. This is why another experiment was done on the model, but this time testing the same model with the water molecule (H_2O). However, as seen by figure 5.15 the results were not as promising as the earlier results. This is mainly because the water molecule has 3 atoms, thus has an extra complexity compared to the other molecules the model was trained with. This extra complexity means that when this Hamiltonian is passed through a network, that is trained on less complex Hamiltonians, the patterns will be different making it hard for the model to predict accurately the energy of the water molecule. In order to over come this issue the model needs to be trained with molecules that have a similar molecular structure.

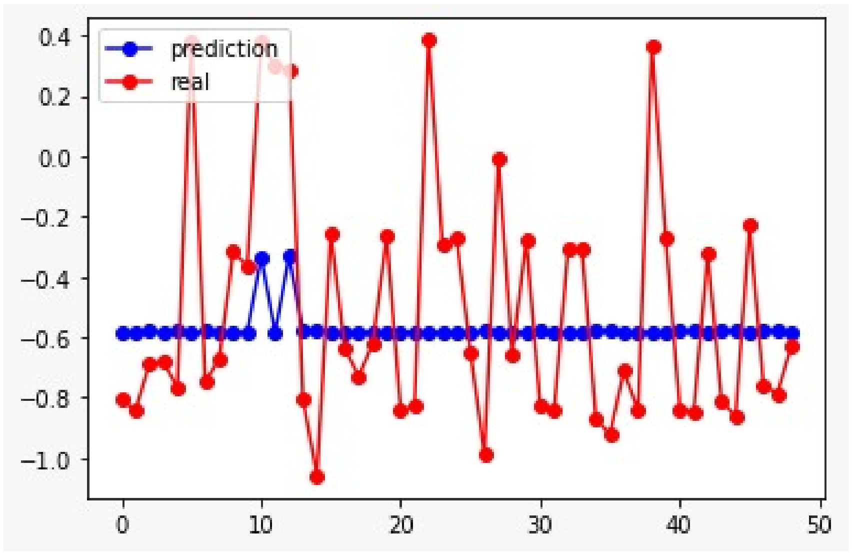


Figure 5.15: This figure shows the testing results when trying to predict the energies of an H_2O molecule. The model was trained with the newly encoded Hamiltonian string using the molecules O_2, LiH, HO^-, FLi, and H_2.

It is hard to confirm if this issue is a result of a flaw in the hypothesis or a data limitation. It could be that the model just hasn't seen enough data of different complexity to generalize for molecules with higher number of atoms. For instance, if a model was trained with molecules with different number of atoms, this model can see different patterns than the ones picked up by the model in this experiment. Moreover, the training data provided for the model was very limited, thus making it hard for the model to provide the generalizability expected.

Nevertheless, if the molecular Hamiltonian does have significant differences when comparing molecules with different number of atoms; then this approach will not be very scalable. Because in that case it will not be possible to build one model that would solve any given Hamiltonian circuit it is given. This hypothesis needs to be further examined before a final conclusion can be reached. Moreover, more training data needs to be provided for a variety of molecules.

One of the limitation this project faced was the lack of computing power. All the data was generated using a quantum computer simulator running locally on a machine with 16 GB of RAM and an Intel I7 8th generation processor. This meant that the any molecule that would require more than 20 qubits will not be possible to be simulated. As a result the biggest molecule that can be simulated on this machine was the water molecule. If access to more processing power, or a real quantum computer is provided, then maybe this hypothesis can be fully tested.

Chapter 6

PROJECT COST

The overall temporal cost of this project was around 600 hours. This time was mostly divided into three main categories: researching, implementing the solution, and writing the report.

The researching took the longs with about 300 hours. This researching involved learning the basics of quantum mechanics and quantum chemistry. Before starting to look into quantum chemistry, the main principles of quantum mechanics needed to be understood. This research started last year with the university's research group, and was mainly focused on quantum computers. However, the research done for this project built on top of the information collected during last year's research to build a clearer idea about these concepts.

Once the main principles of quantum mechanics were established, then the research moved to understanding the concepts of quantum chemistry. These concepts were harder to understand, so they took more time. This is mainly because of the lack of a chemistry background, which made understanding some concepts very difficult. The final part of this researching phase was to define a clear objective so that the development can start.

The development took around 200 hours, divided between the three different approaches this project tested. The first approach took around 45 hours of development before it was clear that this approach was flawed. Then, the second approach took about 100 hours of development. This is took longer than any of the other approaches because the run time of the models was very long, taking up to two days in some cases. Moreover, this approach involved building and debugging the LSTM network which took a lot of time. Finally, the realization of the issues with the scalability of the second approach led to the final approach this project tried. This final approach only took around 35 hours of development. This is mainly due to the fact that all the tools needed to try this approach were already developed and debugged during the second approach.

Once the results of the second approach were observed, the writing process of this report started. This report took about 100 hours of writing. Most of this time was spent reading books and finding sources. The research done in this project required technical language to accurately describe everything. This meant that every thing written had to be checked and referenced. The Gantt diagram in figure 6.1 shows the time spent in each section and the time periods these sections took place.

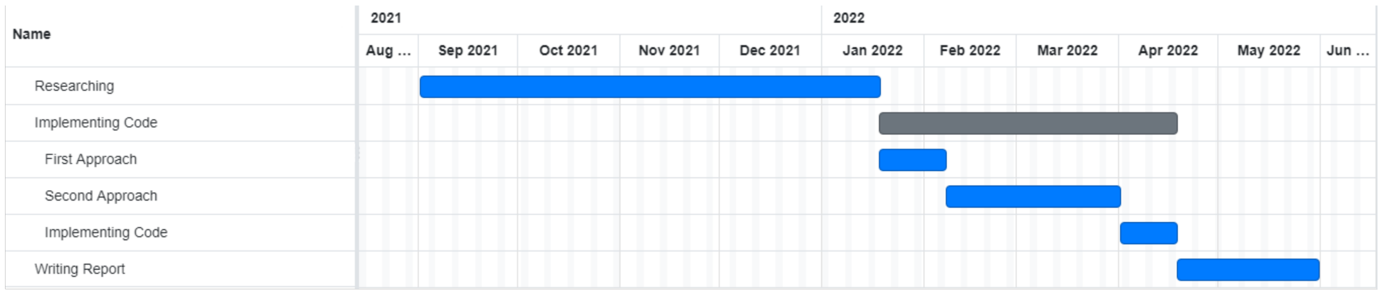


Figure 6.1: This Gantt diagram shows the time distribution in this project.

Chapter 7

CONCLUSIONS

The original objective of this research was to see if deep learning could be used to identify the Hamiltonian's ground state energy using the quantum circuit designed to find that state. The outcomes observed in this project proved that finding the answer to this question would require a lot more training data than the project was able to obtain. This is because with limited data the model was only able to achieve the expected results with molecules of similar complexity. However, this doesn't confirm that given larger amount of data would solve the problem. More testing is needed to reach a final conclusion for this hypothesis. It could be possible that given more data from different molecules of different complexity, the model becomes more generalized. However, Testing this would require generating a large amount of data that would take a lot of time, and would require a lot of computational resources.

The data used in this project, as mentioned earlier, was generated by quantum computer. This means that generating the data for larger molecules would be limited to the number of qubits current quantum systems have. Moreover, it would be hard to check that the model provides accurate results for larger molecules, as their ground state energy can not be calculated by a quantum computer to confirm these results. Moreover, it could be argued that if the quantum computers are able to produce the data needed to train the model with complex molecules. Then, the machine learning model will not be needed.

On the other hand, even if quantum computing is powerful enough to compute the ground state of complex molecules, it would be interesting to see if a machine learning model can learn to do the same with a similar accuracy. This could provide wide accessibility to advanced simulators that can outperform current classical simulators.

7.1 Future Work

One of the things that can be tried to improve these results, is choosing a different neural network for the training of the model. There are other sequential models that would fit the idea that this project was trying to explore. Moreover, it would be interesting to see how changing the type of networks, or even building a custom neural network could help the generalizability of this model.

Furthermore, a way to bypass the limitation of data could be by combining both classical and quantum methods to generate this data. Ideally all the data should be generated by a quantum computer. However, a way to test this hypothesis is to generate the circuits needed to compute the Hamiltonian on a quantum computer, then solve this Hamiltonian classically using DFT or similar methods. This can provide a large dataset that can then be used to test the scalability of this model.

On a different note, the knowledge gained in the project can be used to try and find another solution to the problem at hand. For instance, the data collected from the quantum computer can be used to train a model to perform error correction on the output of the DFT simulators. This work would have to consider the work done on this topic in the paper "Pushing the frontiers of density functionals by solving the fractional electron problem" [52]. The work done in this papery uses deep learning to improve the results of DFT. However, it would be interesting to see how having accurate calculations performed by quantum computers can affect these results.

Another possibility for the future is to try to develop a model that simplifies the Hamiltonian while having the least impact on the system's total energy. This idea would have to be refined, as it is important to decide on which stage of the calculation should this model be applied to. For instance, if the model tries to remove qubits from the Hamiltonian formulation, then this could be pointless as these qubits represent indistinguishable fermions. Meaning that it doesn't matter which qubit is removed it will always have same impact on the energy. Therefore, a machine learning model would not be needed to do that.

However, it could be interesting to see how the machine learning model can try to reduce the active space of a given molecule as much as possible. This will reduce the number of qubits needed to map the molecule, and the decision on the active space to be used is a critical decision that would provide different energy outputs depending on the chosen zone. This could be looked at as an optimization problem, trying to minimize the active zone while minimizing the energy loss.

It could be possible to try and map this problem to a variational quantum eigensolver, and test how a quantum computer would be able to help in this process. This would require a study on how the number of qubits needed to find this active zone would grow with the growth of the molecule. Moreover, other quantum computing techniques can be also used to try and solve this problem like quantum annealing.

Quantum annealing is a type of quantum computation that is built differently than how the quantum-gates based computers are built. Quantum annealing computer excel at solving optimization problems, and offer a larger number of qubits than the quantum-gate computers do.

BIBLIOGRAPHY

[1] Atchade Parfait Adelomou, Elisabet Golobardes Ribé, and Xavier Vilasis Cardona. “Formulation of the Social Workers’ Problem in Quadratic Unconstrained Binary Optimization Form and Solve It on a Quantum Computer”. In: *Journal of Computer and Communications* 08.11 (2020), pp. 44–68. DOI: 10.4236/jcc.2020.811004. URL: https://doi.org/10.4236/jcc.2020.811004.

[2] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer, Sept. 2018. ISBN: 978-3-319-94462-3. DOI: 10.1007/978-3-319-94463-0.

[3] Dorit Aharonov. “A simple proof that Toffoli and Hadamard are quantum universal”. In: *arXiv preprint quant-ph/0301040* (2003).

[4] Guillermo Alonso-Linaje and Parfait Atchade-Adelomou. *EVA: a quantum Exponential Value Approximation algorithm*. June 2021. arXiv: 2106.08731 [quant-ph].

[5] Frank Arute et al. “Quantum supremacy using a programmable superconducting processor”. In: *Nature* 574.7779 (2019), pp. 505–510.

[6] Alán Aspuru-Guzik et al. “Simulated quantum computation of molecular energies”. In: *Science* 309.5741 (2005), pp. 1704–1707.

[7] Parfait Atchade-Adelomou and Guillermo Alonso-Linaje. *Quantum Enhanced Filter: QFilter*. 2021. arXiv: 2104.03418 [quant-ph].

[8] Parfait Atchade-Adelomou, Elisabet Golobardes-Ribé, and Xavier Vilasis-Cardona. “Formulation of the Social Workers’ Problem in Quadratic Unconstrained Binary Optimization Form and Solve It on a Quantum Computer”. In: *Journal of Computer and Communications* 8.11 (2020), pp. 44–68.

[9] Parfait Atchade-Adelomou, Elisabet Golobardes-Ribé, and Xavier Vilasıs-cardona. “Using the Variational-Quantum-Eigensolver (VQE) to Create an Intelligent Social Workers Schedule Problem Solver”. In: *International Conference on Hybrid Artificial Intelligence Systems*. 2020, pp. 245–260.

[10] Parfait Atchade-Adelomou et al. “qRobot: A Quantum Computing Approach in Mobile Robot Order Picking and Batching Problem Solver Optimization”. In: *Algorithms* 14.7 (2021). ISSN: 1999-4893. DOI: 10.3390/a14070194. URL: https://www.mdpi.com/1999-4893/14/7/194.

[11] Parfait Atchade-Adelomou et al. *quantum Case-Based Reasoning (qCBR)*. 2021. arXiv: 2104.00409 [cs.AI].

[12] S. Arun Balaji and K. Baskaran. “Design and Development of Artificial Neural Networking (ANN) system using sigmoid activation function to predict annual rice production in Tamilnadu”. In: *ArXiv* abs/1303.1913 (2013).

[13] Albert P Bartók et al. "Machine-learning approach for one-and two-body corrections to density functional theory: Applications to molecular and condensed water". In: *Physical Review B* 88.5 (2013), p. 054104.

[14] Vivek S. Bawa. *Basic architecture of RNN and LSTM*. Jan. 2017. URL: https://pydeeplearning.weebly.com/blog/basic-architecture-of-rnn-and-lstm.

[15] Feliks A Berezin. "General concept of quantization". In: *Communications in Mathematical Physics* 40.2 (1975), pp. 153–174.

[16] Ville Bergholm et al. "Pennylane: Automatic differentiation of hybrid quantum-classical computations". In: *arXiv preprint arXiv:1811.04968* (2018).

[17] Reinhold Bertlmann. *CHAPTER 4. TIME–INDEPENDENT SCHRODINGER EQUATION*. URL: https://homepage.univie.ac.at/reinhold.bertlmann/pdfs/T2_Skript_Ch_4.pdf.

[18] Jacob Biamonte et al. "Quantum machine learning". In: *Nature* 549.7671 (2017), pp. 195–202.

[19] Arkady Bolotin. "Computational Complexity and the Interpretation of a Quantum State Vector". In: *arXiv preprint arXiv:1304.0508* (2013).

[20] Sergey B Bravyi and Alexei Yu Kitaev. "Fermionic quantum computation". In: *Annals of Physics* 298.1 (2002), pp. 210–226.

[21] Yudong Cao et al. "Quantum chemistry in the age of quantum computing". In: *Chemical reviews* 119.19 (2019), pp. 10856–10915.

[22] Yudong Cao et al. "Quantum chemistry in the age of quantum computing". In: *Chemical reviews* 119.19 (2019), pp. 10856–10915.

[23] Brent Carswell, Barbara D MacCluer, and Alex Schuster. "Composition operators on the Fock space". In: *Acta Sci. Math.(Szeged)* 69.3-4 (2003), pp. 871–887.

[24] Guillaume Charton Françoisand Lample. *Using neural networks to solve advanced mathematics equations*. 2020. URL: https://ai.facebook.com/blog/using-neural-networks-to-solve-advanced-mathematics-equations/.

[25] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[26] Paul Debleena et al. *Artificial intelligence in drug discovery and development - PMC*. 2021. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7577280/.

[27] Amol Deore et al. "The Stages of Drug Discovery and Development Process". In: *Asian Journal of Pharmaceutical Research and Development* 7 (Dec. 2019), pp. 62–67. DOI: 10.22270/ajprd.v7i6.616.

[28] Stancu Dorian and Popa Marin. "Developing A Quantum Circuit Simulator API". In: *ACTA Universitatis Cibiniensis* 67 (Sept. 2015). DOI: 10.1515/aucts-2015-0084.

[29] Jacob D Durrant and J Andrew McCammon. "Molecular dynamics simulations and drug discovery". In: *BMC biology* 9.1 (2011), pp. 1–9.

[30] Pablo Echenique and José Luis Alonso. "A mathematical and computational review of Hartree–Fock SCF methods in quantum chemistry". In: *Molecular Physics* 105.23-24 (2007), pp. 3057–3098.

[31] FDA. *The Drug Development Process*. Jan. 2018. URL: https://www.fda.gov/patients/learn-about-drug-and-device-approvals/drug-development-process.

[32] Zbigniew Ficek and Stuart Swain. *Quantum interference and coherence: theory and experiments*. Vol. 100. Springer Science & Business Media, 2005.

[33] Allan Franklin and Slobodan Perovic. *Appendix 5: Right Experiment, Wrong Theory: The Stern-Gerlach Experiment (Stanford Encyclopedia of Philosophy)*. 2019. URL: https://plato.stanford.edu/entries/physics-experiment/app5.html.

[34] Jennifer Joana Gago et al. "Sequence-to-Sequence Natural Language to Humanoid Robot Sign Language". In: July 2019.

[35] Aidan Gomez. *Backpropogating an LSTM: A numerical example*. Apr. 2016. URL: https://blog.aidangomez.ca/2016/04/17/Backpropogating-an-LSTM-A-Numerical-Example/.

[36] CG Gray, G Karl, and VA Novikov. "Progress in classical and quantum variational principles". In: *Reports on Progress in Physics* 67.2 (2004), p. 159.

[37] Robert B. Griffiths. *Hilbert Space Quantum Mechanics*. Jan. 2014.

[38] Harper R. Grimsley et al. *An adaptive variational algorithm for exact molecular simulations on a quantum computer*. July 2019. URL: https://www.nature.com/articles/s41467-019-10988-2.

[39] Gongde Guo et al. "KNN model-based approach in classification". In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2003, pp. 986–996.

[40] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[41] Anne Marie Helmenstine. *What is ground state in chemistry?* URL: https://www.thoughtco.com/definition-of-ground-state-604422.

[42] Jesús Hernández-Trujillo and Richard FW Bader. "Properties of atoms in molecules: atoms forming molecules". In: *The Journal of Physical Chemistry A* 104.8 (2000), pp. 1779–1794.

[43] Burak Himmetoglu. “Tree based machine learning framework for predicting ground state energies of molecules”. In: *The Journal of chemical physics* 145.13 (2016), p. 134101.

[44] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-term Memory”. In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[45] Matthew Honnibal and Ines Montani. “spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing”. To appear. 2017.

[46] John D Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[47] D. Durrant Jacob and McCammon J. Andrew. *Molecular dynamics simulations and drug discovery | BMC Biology | Full Text*. 2011. URL: https://bmcbiol.biomedcentral.com/articles/10.1186/1741-7007-9-71#Sec4.

[48] Richard Jozsa and Noah Linden. “On the role of entanglement in quantum-computational speed-up”. In: *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 459.2036 (2003), pp. 2011–2032.

[49] Bekir Karlik and A Vehbi Olgac. “Performance analysis of various activation functions in generalized MLP architectures of neural networks”. In: *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011), pp. 111–122.

[50] Hiroki Kawai and Yuya O. Nakagawa. “Predicting excited states from ground state wavefunction by supervised quantum machine learning”. In: *Machine Learning: Science and Technology* 1.4 (Oct. 2020), p. 045027. DOI: 10.1088/2632-2153/aba183. URL: https://doi.org/10.1088/2632-2153/aba183.

[51] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[52] James Kirkpatrick et al. “Pushing the frontiers of density functionals by solving the fractional electron problem”. In: *Science* 374.6573 (2021), pp. 1385–1389.

[53] Thomas Kluyver et al. “Jupyter Notebooks – a publishing format for reproducible computational workflows”. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90.

[54] Alexander A Klyachko. “The Pauli exclusion principle and beyond”. In: *arXiv preprint arXiv:0904.2009* (2009).

[55] Agustinus Kristiadi. *Deriving LSTM gradient for backpropagation*. Aug. 2016. URL: https://agustinus.kristia.de/techblog/2016/08/12/lstm-backprop/.

[56] Simons Lab. *Chapter 2. Second Quantisation*. URL: https://www.tcm.phy.cam.ac.uk/~bds10/tp3/secqu.pdf.

[57] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[58] Qing-Song Li et al. "A unified framework of transformations based on Jordan-Wigner transformation". In: *arXiv preprint arXiv:2108.01725* (2021).

[59] Wenjie Liu et al. "A Novel Quantum Visual Secret Sharing Scheme". In: *IEEE Access* PP (July 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2931073.

[60] Jarrod R McClean et al. "OpenFermion: the electronic structure package for quantum computers". In: *Quantum Science and Technology* 5.3 (2020), p. 034014.

[61] Jarosaw Meller et al. "Molecular dynamics". In: *Encyclopedia of life sciences* 18 (2001).

[62] Ricardo Mendoza. *Entendiendo las Redes Neuronales Artificiales | by Ricardo Mendoza | Medium*. URL: https://medium.com/@ricardojmv85/entendiendo-las-redes-neuronales-artificiales-1e0f4523eae6.

[63] Kyle Mills, Michael Spanner, and Isaac Tamblyn. "Deep learning and the Schrödinger equation". In: *Physical Review A* 96.4 (2017), p. 042113.

[64] Tanja van Mourik, Michael Bühl, and Marie-Pierre Gaigeot. "Density functional theory across chemistry, physics and biology". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 372.2011 (2014), p. 20120488. DOI: 10.1098/rsta.2012.0488. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsta.2012.0488. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2012.0488.

[65] William S Noble. "What is a support vector machine?" In: *Nature biotechnology* 24.12 (2006), pp. 1565–1567.

[66] Christopher Olah. *Understanding LSTM networks*. Aug. 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[67] Atchade-Adelomou Parfait, Elisabet Golobardes-Ribé, and Xavier Vilasis-Cardona. "Formulation of the Social Workers' Problem in Quadratic Unconstrained Binary Optimization Form and Solve It on a Quantum Computer". In: *Journal of Computer and Communications* 8.11 (2020), pp. 44–68. DOI: 10.4236/jcc.2020.811004.

[68] Pennylane. *Circuit Ansatz — PennyLane*. 2019. URL: https://pennylane.ai/qml/glossary/circuit_ansatz.html.

[69] Alberto Peruzzo et al. *A variational eigenvalue solver on a photonic quantum processor*. July 2014. URL: `https://www.nature.com/articles/ncomms5213?origin=ppub`.

[70] John Preskill. "Quantum computing in the NISQ era and beyond". In: *Quantum* 2 (2018), p. 79.

[71] Yanjun Qi. "Random forest for bioinformatics". In: *Ensemble machine learning*. Springer, 2012, pp. 307–323.

[72] Bharath Ramsundar et al. *Deep Learning for the Life Sciences*. `https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837`. O'Reilly Media, 2019.

[73] Ibrahim Saideh. "Entanglement in high dimensional quantum systems". PhD thesis. July 2019.

[74] Jacob T Seeley, Martin J Richard, and Peter J Love. "The Bravyi-Kitaev transformation for quantum computation of electronic structure". In: *The Journal of chemical physics* 137.22 (2012), p. 224109.

[75] Ali Shaib et al. "Efficient Noise Mitigation Technique for Quantum Computing". In: *arXiv preprint arXiv:2109.05136* (2021).

[76] C David Sherrill. "An introduction to Hartree-Fock molecular orbital theory". In: *School of Chemistry and Biochemistry Georgia Institute of Technology* (2000).

[77] David Sherrill. *Basis Sets*. Mar. 2001. URL: `http://vergil.chemistry.gatech.edu/courses/chem4681/background/node5.html`.

[78] David Sherrill. *Slater Determinants*. May 2002. URL: `http://vergil.chemistry.gatech.edu/notes/hf-intro/node4.html`.

[79] Michihiko Sugawara. "Numerical solution of the Schrödinger equation by neural network and genetic algorithm". In: *Computer Physics Communications* 140.3 (2001), pp. 366–380.

[80] Qiming Sun et al. "PySCF: the Python-based simulations of chemistry framework". In: *Wiley Interdisciplinary Reviews: Computational Molecular Science* 8.1 (2018), e1340.

[81] van Mourik Tanja, Bühl Michael, and Gaigeot Marie-Pierre. *Density functional theory across chemistry, physics and biology | Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. 2014. URL: `https://royalsocietypublishing.org/doi/10.1098/rsta.2012.0488`.

[82] PennyLane dev team. *Adaptive circuits for Quantum Chemistry*. Sept. 2021. URL: `https://pennylane.ai/qml/demos/tutorial_adaptive_circuits.html`.

[83] Jules Tilly et al. “Computation of molecular excited states on IBM quantum computers using a discriminative variational quantum eigensolver”. In: *Physical Review A* 102.6 (2020), p. 062425.

[84] Jules Tilly et al. “The Variational Quantum Eigensolver: a review of methods and best practices”. In: *arXiv preprint arXiv:2111.05176* (2021).

[85] Pavan Vadapalli. *Solving Basic Math Equation Using RNN [With Coding Example] | upGrad blog*. 2020. URL: `https://www.upgrad.com/blog/solving-basic-math-equation-using-rnn/`.

[86] Jessica Vamathevan et al. “Applications of machine learning in drug discovery and development”. In: *Nature reviews Drug discovery* 18.6 (2019), pp. 463–477.

[87] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[88] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: `10.1038/s41592-019-0686-2`.

[89] Vei Wang et al. “VASPKIT: A user-friendly interface facilitating high-throughput computing and analysis using VASP code”. In: *Computer Physics Communications* 267 (2021), p. 108033. DOI: `https://doi.org/10.1016/j.cpc.2021.108033`.

[90] Andrew Wolff and Martin. *10.2: Hybrid Orbitals in Water - Chemistry LibreTexts*. Mar. 2020. URL: `https://chem.libretexts.org/Courses/Pacific_Union_College/Quantum_Chemistry/10%3A_Bonding_in_Polyatomic_Molecules/10.02%3A_Hybrid_Orbitals_in_Water`.

[91] RG Woolley and BT Sutcliffe. “Molecular structure and the born—Oppenheimer approximation”. In: *Chemical Physics Letters* 45.2 (1977), pp. 393–398.

[92] Yang Xian. *chapter4: Introduction to Molecular QM*. May 2008. URL: `https://theory.physics.manchester.ac.uk/~xian/qm/chapter4.pdf`.

[93] Jiaying Yang, Ahsan Javed Awan, and Gemma Vall-Llosera. “Support Vector Machines on Noisy Intermediate Scale Quantum Computers”. In: (Sept. 2019). DOI: `10.13140/RG.2.2.17956.63360`.

[94] Noson S. Yanofsky and Mirco A. Mannucci. *Quantum Computing for Computer Scientists*. Aug. 2008. ISBN: 978-0-521-87996-5. DOI: `10.1604/9780521879965`.

[95] Naike Ye, Zekai Yang, and Yuchen Liu. “Applications of density functional theory in COVID-19 drug modeling”. In: *Drug Discovery Today* 27.5 (2022), pp. 1411–1419. ISSN: 1359-6446. DOI: `https://doi.org/10.1016/j.drudis.2021.12.017`. URL: `https://www.sciencedirect.com/science/article/pii/S1359644621005687`.