
Neural Nets for Hartree-Fock

NEURAL NETS FOR HARTREE-FOCK

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Bachelor of Arts

in

Physics

by

Timothy D. Strang

DARTMOUTH COLLEGE

Hanover, New Hampshire

June 2, 2025

Examining Committee:

James Whitfield, Chair

Chandrasekhar Ramanathan

Paul Robustelli

Abstract

Hartree-Fock is one of the cornerstone numerical methods used to approximate the ground-state quantum wavefunctions of atomic many-body systems, utilizing several physical approximations before iteratively refining an ansatz function to achieve minimal energy. Multiple traditionally coded programs exist to carry out these calculations, but convergence to a global minimum is not guaranteed and run times may scale poorly with system size. One solution may be to use a quantum computer to simulate quantum many-body systems. As part of the Whitfield Group, a long-term goal is to identify and benchmark areas such as these that may present opportunities for a quantum speedup. However, to conclusively identify a quantum computational advantage, all possible methods of classical computation must be shown to be less efficient than the quantum calculations. The goal of this thesis is to probe several neural net based approaches to the Hartree-Fock algorithm. In addition to laying the groundwork for future quantum computational benchmarks, all methodologies employed are assessed for immediate practical applicability. A variety of neural nets were trained to reproduce the results of traditionally-coded techniques, then checked for fidelity and speedups. A net was successfully trained to predict traditionally-calculated Hartree-Fock energies on a database of real, small molecules with high fidelity and drastically reduced CPU

utilization times. Additional nets were trained on a database of randomly generated molecules to predict Hartree-Fock and CISD energies, yielding reasonable fidelities and a similarly drastic speedup, at the front-end expense of the lengthy training process. Overall, these results should be viewed as a promising initial probe into the much broader problem space.

Acknowledgements

First and foremost I'd like to thank my advisor James Whitfield and the rest of the Whitfield Group for all the support they've given me over the past few years. My work with the Group, especially on this thesis, has helped me grow intellectually in ways I could never have accomplished on my own.

Secondly, I would like to thank my parents for not only supporting me through college, but also for never failing to encourage me to pursue my passions. I wouldn't be here without them.

Contents

Abstract	ii
Preface	iv
1 Introduction	1
1.1 The Hartree-Fock Method	1
1.1.1 Ab Initio Derivation	2
1.1.2 Hamiltonian Approximations	5
1.1.3 Matrix Formulation	9
1.1.4 HF Algorithm	10
1.2 Machine Learning: Motivation and Application	11
1.2.1 Neural Net Basics	14
1.3 Overview and Context	20
1.4 Statement of Thesis	21
2 Methods	22
2.1 Initial Trial: Real Molecules	23
2.1.1 Training	24
2.2 Secondary Trials: Random Molecules	27
2.2.1 Custom Hyperparameter Optimizer	29

2.2.2	Random Molecules	37
3	Results and Analysis	41
3.1	Initial Trial: Real Molecules	41
3.2	Secondary Trial: Random Molecules	47
3.2.1	Convergent Energy Trial	47
3.2.2	Correlation Energy Trial	50
4	Conclusion	56
4.1	Future Research	59
4.1.1	Hyperparameter Optimizer Improvements	59
4.1.2	New Datasets	63
4.1.3	Charting Trends	63
4.2	Closing Remarks	64

Chapter 1

Introduction

1.1 The Hartree-Fock Method

Since Schrödinger's groundbreaking wavefunction formulation of quantum physics in the early 1900's, physicists, chemists, and computer scientists alike have been attempting to numerically solve his namesake differential equation in the vast majority of circumstances where analytical methods fail. One of the most successful early approaches, which has become the basis for much of modern quantum chemistry, is the Hartree-Fock procedure. This procedure iteratively determines molecular electronic structure up to several key approximations, resulting in a set of molecular orbital functions and energies. For this introduction, I will discuss both the nature of these approximations and two formulations of the procedure itself.

1.1 The Hartree-Fock Method

1.1.1 Ab Initio Derivation

Wavefunction Form

We begin by discussing the approximate nature of a many-body fermionic wavefunction. Using the properties of differential equations, we can view this many-body wavefunction as a construction of simpler, building-block wavefunctions, the spatial components of which are called basis orbitals and denoted $\phi(\mathbf{r})$. In crystal lattices, basis orbitals tend to be periodic plane waves, whereas in molecular settings they are traditionally the familiar atomic electron “shells” from introductory chemistry. The simplest way to attempt construction of a many-body wavefunction from a set of orbitals is by simple multiplication, i.e.

$$\psi = \prod_i \phi_i(\mathbf{r}_i) \quad (1.1)$$

We make use of a similar form that holds for most scenarios of interest when factoring in the 1/2 spin degree of freedom in fermions, notationally:

$$\chi(\mathbf{x}) = \phi(\mathbf{r})\sigma(\omega) \quad (1.2)$$

Where $\sigma(\omega)$ represents the spin component of the wavefunction and the hybrid χ is called a “spin-orbital”. We might now naively write out the many-body wavefunction of the molecule as:

$$\psi = \prod_i \chi_i(\mathbf{x}_i) \quad (1.3)$$

This was Hartree’s original method (Hartree, 1928), but as pointed out independently by Slater and Fock several years later (Slater, 1930), (Fock, 1930), violates

1.1 The Hartree-Fock Method

quantum antisymmetry requirements for systems of fermions. A mathematical work-around is to employ Slater determinants, modifications of the simple product that automatically guarantee antisymmetrization. In place of Eq. 3, we now have:

$$\psi = \det \begin{bmatrix} \chi_1(\mathbf{x}_1) & \dots & \chi_N(\mathbf{x}_1) \\ \dots & \dots & \dots \\ \chi_1(\mathbf{x}_N) & \dots & \chi_N(\mathbf{x}_N) \end{bmatrix} \quad (1.4)$$

Where N is the total number of fermions. Since each particle can only occupy one basis orbital per Slater determinant, we have M choose N possible Slater determinant ψ 's in our basis state space. Operating in this space, we can analytically determine the ground state of the system by minimizing the energy with respect to a linear combination degree of freedom:

$$\Phi = \sum_J c_J \psi_J \quad (1.5)$$

Where c_J is a complex coefficient to be determined and J is the collective indexing set of all possible Slater determinants ψ_J . The process of energy minimization in this manner is called “full configuration interaction” (FCI), or “exact diagonalization”, and is the brute force method for calculating electronic properties. Should our basis set be complete in the state space, then this procedure would be exact. Unfortunately, optimization over an infinite-dimensional state space is, in general, considered intractable. Indeed, for any basis set of size M reasonable for the vast majority of molecules, the space of M choose N Slater determinants is already too large to be computationally tractable.

1.1 The Hartree-Fock Method

Single Determinant Form

This is the fundamental problem at the heart of quantum chemistry, and scientists have been brainstorming work-arounds using varying degrees of approximation since the early days of quantum mechanics. One of the first approximative method to be proposed was the Hartree-Fock algorithm (Hartree, 1928), devised to provide a basic picture of electronic structure while remaining reasonably tractable on small molecules. The mathematical trick behind Hartree-Fock is the restriction of the problem from complete diagonalization over the space of all possible Slater determinants to the minimization of energy with respect to a single Slater determinant. Effectively, the degree of freedom over which optimization occurs is just swapped from linear combinations of N-particle states to rotations of the orbital basis space. The result is the optimized Hartree-Fock wavefunction

$$\Psi_{HF} = \det \begin{bmatrix} \chi'_1(\mathbf{x}_1) & \dots & \chi'_N(\mathbf{x}_1) \\ \dots & \dots & \dots \\ \chi'_1(\mathbf{x}_N) & \dots & \chi'_N(\mathbf{x}_N) \end{bmatrix} \quad (1.6)$$

Where χ'_i represent the new, rotationally optimized spin-orbitals, formed from the original basis space. Mathematically, this approach is equivalent to a mean-field approximation for each electron; in other words, when determining the wavefunction for one electron, the probability distributions of all other electrons are approximated to be distributions of classical charge.

1.1 The Hartree-Fock Method

1.1.2 Hamiltonian Approximations

In order to find the ground state, the variational theorem states that we must minimize the electronic energy expectation value. This is quantified by action of the electronic Hamiltonian operator on $|\Psi\rangle$. As is often the case, we can separate this Hamiltonian into a summation of operators corresponding to each source of energy. The first approximation we make is the Born-Oppenheimer approximation, which treats molecular nuclei as fixed bodies, due to the negligible relative mass of electrons. Under this assumption, we may ignore any terms regarding the inter-nucleonic potential as an arbitrary additive constant to the energy of the system, leaving us with only three meaningful terms in our Hamiltonian:

$$H \approx T_e + V_{eN} + V_{ee} \quad (1.7)$$

Where T_e represents the kinetic energy operator on the electrons, V_{eN} represents the nuclear-electronic potential, and V_{ee} represents inter-electronic potentials, expressible as a sum of individual two-electron terms. Due to the inherent differences in calculations involving one- and two-electron terms, it is often convenient to define separate operators for each. Here we can write:

$$h_i = -\frac{1}{2}\nabla_i^2 - \sum_A \frac{Z_A}{r_{iA}} \quad (1.8)$$

Where the first term is simply the non-relativistic kinetic energy operator and the second term sums the Coulomb potential in convenient choice of units from each nucleon. The restriction of our operators to the strictly non-relativistic is another key approximation present in most computational methods for quantum chem-

1.1 The Hartree-Fock Method

istry, but is usually assumed to be relatively unimportant compared to the mean-field. The two-electron Hamiltonian terms, also Coulomb operators, take the form:

$$v_{ij} = \frac{1}{r_{ij}} \quad (1.9)$$

representing the electrostatic interaction of electrons i and j . Here we see the Born-Oppenheimer approximation at work, as the nucleons are treated as a fixed source of external potential for each electron in Eq. 8, whereas the electron-electron terms depend on the orbitals of both fermions in Eq. 9. After some mathematical simplification is done to the general expectation value equation:

$$\langle \Psi | H | \Psi \rangle = \bar{E} \quad (1.10)$$

we obtain the following expression, equivalent up to our assumptions:

$$h(x_1) |\chi_i(x_1)\rangle + \sum_j [\langle \chi_j(x_2) | \frac{1}{r_{12}} | \chi_j(x_2) \rangle |\chi_i(x_1)\rangle - \langle \chi_j(x_2) | \frac{1}{r_{12}} | \chi_i(x_2) \rangle |\chi_j(x_1)\rangle] = \epsilon_i |\chi_i(x_1)\rangle \quad (1.11)$$

Eq. 11 is in general a system of coupled partial differential equations wherein ϵ_i represents the energy associated with the i^{th} electron's Hamiltonian. Written more heuristically,

$$\epsilon_i = \langle i | h | i \rangle + \sum_j [ii|jj] - [ij|ji] \quad (1.12)$$

Where here the first term on the right, again, represents our single electron energy, and the sum expresses the inter-electronic potential. The two terms in the sum are the electron-electron Coulomb potential and the completely non-classical fermionic exchange potential. The exchange potential arises completely from the

1.1 The Hartree-Fock Method

antisymmetry of $|\Psi\rangle$; thus, Hartree-Fock is exact in its treatment of exchange potential.

What is less clear here is the presence of the implicit "mean-field" approximation mentioned earlier. Whereas FCI optimizes over a global molecular basis, i.e. optimizes a superposition of ensemble states, Hartree-Fock does not. Our degree of freedom has been restricted to the altering of the χ' single-particle orbitals. Once these orbitals are determined, they are combined in a specific manner — the Slater determinant. What this does conceptually is ignore correlations between global electronic states, which individually can have equal or slightly larger energy than the HF ground state but in superposition may actually yield a lower energy molecule. This is technically only one-half of the true electronic "correlation energy", the other being the fermionic exchange potential, which is already accounted for. Regardless, the exclusion of this "Coulombic correlation" energy is usually considered Hartree-Fock's largest source of error, leading it to systematically underestimate ground state energies and rendering the method incapable of describing some physical phenomena entirely, like London dispersion. Other algorithms exist to augment or replace Hartree-Fock in situations where the correlation energy or its effects are deemed important. Those that take place on top of Hartree-Fock, using the converged HF determinant as an ansatz and then loosening the mean field constraint and optimizing, are generally named after the number of electron excitations considered. Single-double CI, or CISD, for example, considers the space of determinants that include one or two excited electrons each, in addition to the HF determinant. This is the only post-Hartree-Fock method relevant to

1.1 The Hartree-Fock Method

this thesis, but a wide array exist (Mag, 2009), and vary depending on the desired level of accuracy. Of course, each additional increment of accuracy comes with a drawback particular to the method used — usually in the form of computational complexity. There is no cheating Schrödinger.

Returning to Eq. 11, we can go a different route with our notation. As we will see, this new formulation has some useful advantages for practical calculations. We define two operators:

$$J_j = \langle \chi_j(x_2) | \frac{1}{r_{12}} | \chi_j(x_2) \rangle \quad (1.13)$$

$$K_{ij} = \langle \chi_j(x_2) | \frac{1}{r_{12}} | \chi_i(x_2) \rangle \quad (1.14)$$

Called the Coulomb and exchange operators, respectively. Thus, our previously monstrous Eq. 11 now reads:

$$(h_i + \sum_j J_j - K_{ij}) | \chi_i(x_1) \rangle = \epsilon_i | \chi_i(x_1) \rangle \quad (1.15)$$

Or, in compacted notation,

$$f_i | \chi_i \rangle = \epsilon_i | \chi_i \rangle \quad (1.16)$$

Where f is known as the Fock operator, and coordinate indexes are implied. In this form, we see most clearly the base eigenfunction equation that we are attempting to solve. Note that f_i is a single-particle Hamiltonian, but its structure depends uniquely on the orbitals of all other particles. Unless each $| \chi_i \rangle$ is perfectly degenerate under its particular Fock operator, we do not in general obtain an energy eigenstate for the full molecular ensemble, highlighting again a lack of correlation

1.1 The Hartree-Fock Method

considerations in procedures of this kind. There may exist some simplifications of the Fock operator depending on symmetries in the system, for example in periodic crystalline systems or molecules where all spatial orbitals are double-filled. For the rest of this section, however, we will continue with the most general case.

1.1.3 Matrix Formulation

Solving the eigenfunction equation given by the Fock operator in unrestricted function space is made easier with the aid of computer driven integration, but in general remains intractable — hence the use of a finite basis. This turns a large part of the calculation into a linear algebra problem, which is much more easily solved. To start, we denote explicitly the set of basis spin-orbitals as $\{\tilde{\chi}\}$. We then create ansatz occupied orbitals using linear combinations of the basis,

$$\chi_i = \sum_{\nu} C_{\nu i} \tilde{\chi}_{\nu} \quad (1.17)$$

Where $C_{\nu i}$ is simply some real coefficient. So we may now, using Eq.'s 16 and 17, write out:

$$\sum_{\nu} C_{\nu i} \langle \tilde{\chi}_{\mu} | f | \tilde{\chi}_{\nu} \rangle = \epsilon_i \sum_{\nu} C_{\nu i} \langle \tilde{\chi}_{\mu} | \tilde{\chi}_{\nu} \rangle \quad (1.18)$$

Which has a useful factoring out of real coefficients. Since these inner products are the sums of discretely indexed terms, it is natural to organize them into matrices as follows:

$$S_{\mu\nu} = \langle \tilde{\chi}_{\mu} | \tilde{\chi}_{\nu} \rangle \quad (1.19)$$

$$F_{\mu\nu} = \langle \tilde{\chi}_{\mu} | f | \tilde{\chi}_{\nu} \rangle \quad (1.20)$$

1.1 The Hartree-Fock Method

Where now we see that

$$\sum_{\nu} F_{\mu\nu} C_{\nu i} = \epsilon_i \sum_{\nu} S_{\mu\nu} C_{\nu i} \quad (1.21)$$

Which is just the formula for generic matrix multiplication if we treat C as a matrix with axes ν and i and ϵ as a diagonal matrix over i . This is called the Roothaan equation after the creator of the matrix method (Roothaan, 1951), and is commonly stated as

$$FC = SC\epsilon \quad (1.22)$$

Where F is known as the Fock matrix, owing to its dependence on the Fock operator f , and S as the “overlap matrix”, as its entries are inner products on a function space. Thus, in an orthonormal basis, S approaches the identity, and the Roothaan equation becomes a regular matrix eigenvalue equation. While usage of orthonormal bases is the historical norm, it is by no means necessary, especially with modern computational tools.

1.1.4 HF Algorithm

With all the structure in place, it is now possible to iteratively optimize the spin-orbitals. The process naturally takes a different form for the Roothaan equation than for regular Hartree-Fock. We begin by discussing the latter.

We begin either with a guess of the electronic orbitals, or by simply calculating the first one in the naked nucleonic potential under Born-Oppenheimer. We then proceed in calculating each new orbital by factoring in all those already calculated, and repeat until each new iteration corrects the previous energy by less than

1.2 Machine Learning: Motivation and Application

some predetermined cutoff value. Because of this process, HF is known as a "self-consistent field" method, since we terminate when the orbitals are (approximately) mutually consistent in their energies. Note that convergence is not necessarily guaranteed by this process alone. Each update of a single spin-orbital affects the two-body components of every other electron's Fock operator, meaning even minor changes require laborious integration in the subsequent step.

In the Roothaan formulation, we solve all electronic minimization problems at once, using either eigenvalue or generalized eigenvalue equation solvers to diagonalize the Fock matrix with minimal ϵ sum. This diagonalization will generate a new coefficient matrix C' , which represents the altered spin-orbitals with respect to the chosen basis. We begin the process again with the new C' , generating and diagonalizing a new Fock matrix until $|\Delta C|$ between steps is below a predetermined threshold. Note that updates to the Fock matrix between iterations are generated by the Fock operator's dependence on occupied states. This dependence is sometimes obscured by Eq.'s 21 and 22 but is, unfortunately, unavoidable.

1.2 Machine Learning: Motivation and Application

The Hartree-Fock method has been around much longer than the modern computational techniques now used to solve it. Indeed, the same electro-chemical theories that the HF algorithm was designed to model were crucial to the development of technologies like the transistor, which we use today in undergraduate theses to carry out more and better HF calculations. In the early days of quantum theory, computers referred to human beings who had to sit down and do the

1.2 Machine Learning: Motivation and Application

tedious integrations and eigenvalue optimizations themselves, rendering long iterative methods like Hartree-Fock intractable on all but the most trivial molecules. As machine computers slowly became accessible in academic settings after the advent of the transistor, opportunity was quickly seized to apply the new technology to quantum chemistry (Miller et al., 1957). Up until very recently, the field of computational quantum chemistry has been dominated by traditionally-coded eigenvalue solvers and integrators. These have vastly expanded the region of molecule space able to be probed numerically by scientists, and more modern open-source quantum chemistry program suites like PySCF have even made some of the calculations possible for amateurs. However, traditionally coded techniques have a hard theoretical limit on efficiency, due to uncertainties in convergence and the exponential scaling of problem complexity with molecule size. Even under clever assumptions and using our most efficient mathematical and computational tricks, supercomputers are often still necessary. With Moore's Law for traditional computers approaching a possible end as transistors shrink to the atomic scale (Powell, 2008), demand is high for creative new techniques.

In recent years, technology has matured enough to bring two alternative forms of computing into the mainstream discourse. The first is quantum computation, which, if achieved on a practical level, has been theorized to have particular potential for simulating quantum systems (Fey). The reasoning behind this is simple: classical computers, like humans, are simply not optimized to process the complicated differential equations and many-body coupled interactions of the quantum world. Quantum computers on the other hand, either by analogue or using binary

1.2 Machine Learning: Motivation and Application

qubits, have these same quantum properties built in to every calculation. We already know that such properties can be harnessed to improve certain classically difficult problems, like the factorization of large integers and efficient database searching. As larger, more reliable quantum computers enter the fold, experts believe that quantum chemistry will be one of the first fields to see widespread application.

Another alternative computational method experiencing a recent spike in popularity is machine learning, especially neural nets. Though the first neural net was actually fabricated not long after computers became available in the laboratory (Rosenblatt), the enormous parameter spaces of neural nets that must be optimized in order to compete with traditional coding rendered them impractical for several decades. In modern times, however, computers have access to exponentially increased functionality in areas crucial to the resource-intensive procedures used to train neural nets. Increased RAM allows computers to manipulate the parameter space of deep neural nets much more effectively; GPU's are custom designed to handle the intricate linear algebra involved in training algorithms; and the rise of big data in industry provides researchers with the massive databanks needed for most practical applications.

In relation to quantum chemistry, neural nets may serve two purposes. The first is the obvious one: solving the Schrödinger equation is extremely complicated to do, even using approximation algorithms. Thus, a switch to more organic, adaptive styles of computation like neural nets could be the solution. The second purpose

1.2 Machine Learning: Motivation and Application

is indirect, in that all coding methodologies available to classical computers must be tried and tested before we can accurately benchmark possible advantages that may be unique to quantum computation, which as mentioned above is expected to be especially potent in the field of quantum simulation. In accordance with this two-fold goal, this thesis aims both to test neural nets for possible improvements over existing computational methods, and to provide a more complete standard against which quantum algorithms may be judged. The next section is dedicated to a brief introduction to the mechanics of neural nets as they relate to this thesis.

1.2.1 Neural Net Basics

Structure and Propagation

Neural nets were given their name based on their original biological inspiration: the human brain. In rough analogy with biological brains, neural nets are arrays of “neurons” arranged in a series of layers, each sharing a unique connections certain neighboring neurons, laid out in a particular pattern designed to accomplish a particular target function. From a computer science perspective, a neuron is just a specific location for a float reserved in the net’s allotted RAM.

The first and last layers of neurons are special: they are the input and output layers, respectively. The input layer is self-explanatory, the float stored in the j^{th} neuron’s memory slot corresponding to the j^{th} entry of the input vector. Of course, not all interesting inputs are naturally vectors, but most datasets can be converted with relative ease. For example, an input vector may be the value of activation for each pixel in an image, or a number represented in binary. From there, a propagation function takes as inputs the values stored by each input neuron, and through a

1.2 Machine Learning: Motivation and Application

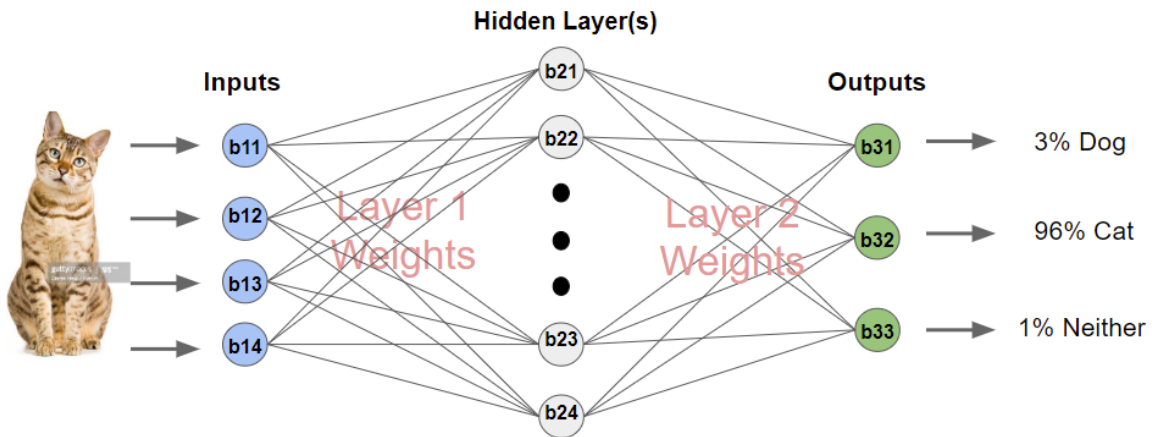


Figure 1.1: A typical neural net. Circles represent neurons, and lines their connective functions. This net is reasonably well trained to differentiate images of cats and dogs, as we see from the outputs. Image of the cat supplied by Getty Images Embed.

predetermined mathematical procedure determines the floats that will be stored in the next layer of neurons. Often, these floats, known as “neuron activations”, are restrained to be between zero and one and can be interpreted as a percentage of the neuron’s signal strength.

The propagation function also takes several other inputs. These are the *parameters* of the layer, and are differentiated into the groupings of “weights” and “biases”. Whereas neuron activations are input-dependent, parameters are properties of the neural net itself, and are only adjusted during training. *Weights* are parameters associated with two neurons of adjacent layers, while *biases* are assigned to each single neuron. Biases are, roughly speaking, the “baseline importance” of a neuron, and weights characterize the interactions between neural layers. The simplest propagation function is just a matrix multiplication of the form: <layer n

1.2 Machine Learning: Motivation and Application

activation vector $\rangle = [\text{weight matrix}] \langle \text{layer n-1 activations} * \text{biases} \rangle$. Most more advanced propagation functions are a variation on this formula, usually with the simple addition of some normalization factor. The propagation process repeats until all neurons up to the final output layer have received their activations. Like the input layer, the output layer represents a vector of floats based on its neurons' activations, which may or may not be then converted into some other form of data, like an image or a graph. This is the net's guess or prediction for the given input. For example, the neural net shown in Figure 1.1 is trained to differentiate between images of cats and dogs, with input activations corresponding to pixel brightness and output activations corresponding to the strength of the net's "suspicion" as to which category the image belongs.

Training and Backpropagation

Theoretically, one could do the math out by hand and custom pick the weights and biases for a particular neural net that will yield the desired outputs over a specific dataset. This would be a form of traditional coding — i.e., one where the human programmer determines an explicit manner in which a desired task can be accomplished, and then sets to work telling the computer how to do it. Not only would this be the world's least user-friendly programming language, it would also be vulnerable to all of the same pitfalls machine learning was invented to avoid. For example, it would be nearly impossible for a programmer to sit down and write a program in C++ that could recognize images of dogs and cats up to 99% accuracy. How do you explain to a computer what a dog is? What constitutes a cat in a particular sea of black-and-white pixel activations, but a fish in another?

1.2 Machine Learning: Motivation and Application

Where do you even start? In a technical sense, the basic function of a neural net is to learn a particular transformation between two vector spaces of fixed dimension. In order to be useful, the net must somehow be trained to approximate the desired transformation without the programmer's explicit knowledge of the mathematical underpinnings of the transformation in question. The process by which this is accomplished is called backpropagation, and it lies at the heart of all neural networks.

For this thesis, we employ only supervised machine learning. This type of learning requires a large, fully-labeled dataset, i.e., a collection of inputs with predetermined associated outputs depending on the function we desire our net to serve. Continuing with the image recognition example, a well-made dataset might include 10,000 image files of dogs and cats, each with an associated label text file reading "dog" or "cat". In this case, the labels were most likely attached to the images by a group of humans beforehand. The acquisition of fully-labelled, sufficiently large datasets is a fundamental challenge for programmers looking to produce a supervised neural net. However, assuming that we do have such a dataset, the next step is to partition it. A standard split takes a random 80% of the total dataset, with labels, and marks it off for training, while the remaining 20% of data points are kept reserved for testing finalized nets. The problem with utilizing the same set of labelled inputs for both training and testing is that it opens the door for overfitting. Specifically, we have no guarantee that a trained net has not created a dataset-specific program, memorizing the training inputs and correlating them with remembered labels, until we test it on new material. Overfitted nets will in

1.2 Machine Learning: Motivation and Application

general perform extremely well on their training space, but suffer a sharp decline in accuracy when tested on unfamiliar inputs.

So, assuming that we have at our disposal a well-made dataset with the proper partitions, we are ready to begin training our net. Training tends to be the most time consuming part of the process, and execution of computationally-dense backpropagation algorithms is the basis of most challenges for net-based programming. To start, a net is primed with randomized parameters, and then fed each of the inputs in the training partition of the dataset one by one. Each time an input is entered, propagation occurs, resulting in an output. Since propagation is a function of the parameters, outputs are initially more or less random, and rarely correlate well with the labels associated to particular inputs. Next comes backpropagation, during which a loss function — usually some simple statistical metric like mean squared error — determines a “score” by which the net can judge its own performance. If the loss is high, the backpropagation algorithm recognizes that the net’s parameters require significant adjustment. Layer by layer, the backpropagation function retraces the propagation function’s steps, altering the parameter space slightly in a manner calculated to shift outputs closer to the labels assigned to each input. Over many many generations of this cycle, we hope to see convergence in the parameter space corresponding to an optimized net that, when given any input in the test set, will return an output approaching the label with reasonable accuracy.

The discussion above was intentionally vague on the actual mechanics of the back-

1.2 Machine Learning: Motivation and Application

propagator. This is because the search for more efficient backpropagation techniques is an important branch of ongoing research. The simplest, and historically first, backpropagation scheme is direct stochastic gradient descent, or SGD. All backpropagators treat the net's parameters as a vector space of extremely high dimension. After receiving and propagating an input, a net running SGD will determine the gradient of the loss function with respect to the parameter space, then shift each parameter along the negative gradient by an amount proportional to both the loss and a pre-assigned learning rate. Learning rates are usually a manually entered value between zero and one, and correct choice of learning rate is task-specific and somewhat arbitrary. Larger learning rates may jump over important features like minima, but also require fewer datapoints for training, whereas lower learning rates have the opposite properties. Neither is universally preferable. Regardless of learning rate, basic SGD suffers from a tendency to get caught in local minima, when convergence to the global loss minimum is the net's long-term goal. To address this, more advanced techniques, like annealing, often supplement SGD when it is used, and a whole host of other backpropagators are available out of the box in most neural net packages.

All of the non-parameter variables mentioned above — learning rate, momentum, choice of backpropagator — are examples of the large class of variables known as hyperparameters. Hyperparameters, unlike parameters, are not optimized over the course of training. Rather, they have to do with the architecture of the net and the training algorithm itself. As such, suitable choosing of hyperparameters is often a subtle process, which nonetheless may be crucial for the success or failure of

1.3 Overview and Context

the training. This is another fundamental challenge for would-be neural net programmers. Additional hyperparameters include the shape of the neural network, as well as the possible inclusion of nonlinear layers to the net architecture. Nonlinear layers tend to be used as statistical measures to enhance the probability of parameter convergence during training, but are removed once the net is complete. Several are used in the nets trained for this thesis, and will be explained in their relevant sections.

1.3 Overview and Context

For this thesis, we use a variety of techniques involving neural nets to carry out the Hartree-Fock algorithm on datasets of small molecules. The motivation, as outlined in more detail in Section 1.2, is two-fold: 1) identify possible areas in the problem space where machine learning is more effective than traditional coding, and 2) work towards a more comprehensive benchmark for quantum computational methods. The first of these comes with a caveat: given resource limitations for undergraduate researchers, we cannot reasonably expect to compete with the cutting edge of accuracy in our neural nets. It would be redundant to document machine learning on chemical or analytically calculated energies (Rupp et al., 2012a). Thus, we only attempt to match the results achieved by traditional coding methods — for this thesis, the PySCF (Sun et al., 2018) open-source Python package — and then identify areas of increased capability or efficiency. For the sake of establishing a controlled benchmark for computational speedup, all calculations for every method used were run on a local machine (11th Gen Intel(R) Core(TM) i7 @ 2.80 GHz, 15.7 GB usable RAM) in the Ubuntu virtual environment. Neural nets

1.4 Statement of Thesis

were trained to calculate several characteristic molecular energies based on PySCF calculations, and then compared against the traditional program for speed and accuracy. This process spanned several PySCF methods, as well as two separate datasets of molecules — one real and one randomly generated — and incorporated a broad range of computational techniques to boost net performance and training.

1.4 Statement of Thesis

In the course of this thesis, we argue that relatively small neural networks are capable of reproducing PySCF's calculations of Hartree-Fock energy for real, small, closed-shell molecules in a fraction of the runtime, and can most likely be adapted to handle open-shell molecules, unrestricted Hartree-Fock, and CISD calculations with comparable accuracy. This thesis should be read as an initial probe rather than a final result in pursuit of long-term goals (1) and (2) as stated in section 1.3.

Chapter 2

Methods

2.1 Initial Trial: Real Molecules

The first challenge in any machine learning project is identification of a viable dataset. Thankfully, modern computing has allowed the quantum chemistry community to compile a staggering amount of data on a range of small molecules, some of which are available open-source on the internet. For the first trial, I utilized the QM7 dataset (Rupp et al., 2012b), a subset of the much larger GDB-13 dataset, which contains almost one billion stable and synthesizable molecules (Blum and Reymond, 2009). QM7 includes just 7,165 of these, consisting of the set of all GDB-13 molecules with 23 or fewer atoms. The heaviest atom included in any QM7 molecule is silicon. The database includes a compilation of several data files for each molecule: a list of atomic numbers corresponding to the molecular nuclei, the xyz coordinates of each nucleus (relative to an arbitrary origin), and a Coulomb matrix. Coulomb matrices are a hybrid description of a molecule's structure, combining nuclear charges and positions into a single characteristic matrix which is invariant under rotations and translations of the spatial coordinate system. Coulomb matrices have several desirable properties, but are also larger by a factor of 437 when compared to the unprocessed atomic number and xyz data. Thus, they were ignored in favor of a more bare-bones approach for this thesis. Note that, theoretically, the Coulomb matrix contains only information that is already obtainable from the first two file types, which fully characterize the molecule. Potential use arises primarily in situations where coordinate invariance is particularly difficult to establish post facto. For the purposes of this thesis, nuclear charges and positions were separately formatted into csv files for data pre-processing with NumPy (Harris et al., 2020), then split into two randomized partitions in the stan-

2.1 Initial Trial: Real Molecules

ard 80/20 ratio.

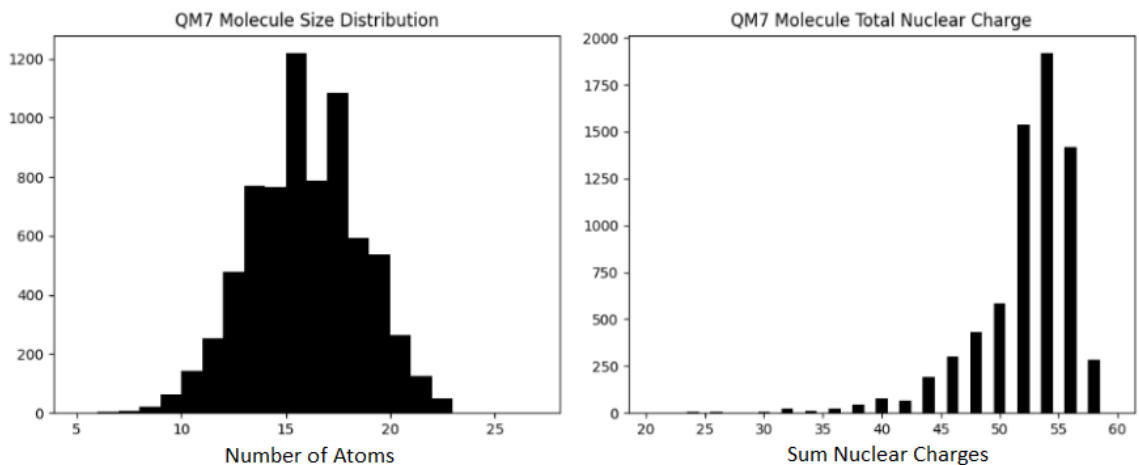


Figure 2.1: A profile of the types of molecules included in the QM7 RHF-compatible subset based on size and total nuclear charge.

2.1.1 Training

All neural net functionality in this thesis was implemented using the PyTorch suite of tools. Training of the initial trial was done in the straightforward manner, by running a training loop on a variety of nets with tweaked hyperparameters and assessing performance at intervals manually. The error metric chosen for this trial was straightforward mean squared error, and was implemented using the `torch.nn` module's `MSELoss` method.

Final Net

The final net that was selected in this manner had two hidden layers consisting of 500 and 250 neurons, respectively (skip to Figure 3.1 for a visual schematic).

2.1 Initial Trial: Real Molecules

Several other hyperparameters were also tweaked. Firstly, input data was pre-processed. Nuclear charge and position matrices were laid out in the format: Atom 1 charge, Atom 1 \hat{x} position, Atom 1 \hat{y} Position, Atom 1 \hat{z} Position, Atom 2 charge, ... etc. with appropriate zeros padding the end for molecules with fewer than 23 atoms. The axis labels are, of course, arbitrary, and determined by the order of the coordinates in the QM7 position file. Then, the total nuclear charge of the molecule was calculated, along with the "length" of the molecule, i.e. the total number of nuclei it contains. These two numbers were appended to the end of inputs before being fed to the net, and were intended to function to give the net as much information as possible without significantly increasing the length of input vectors. With this measure in place, input vectors were 94 elements in length: the $4 \times 23 = 92$ elements needed to characterize nuclear structure, followed by our two additional values.

It was also found to be beneficial for backpropagation to include two nonlinear layers in the net's architecture. These consisted of an initial "dropout" layer followed by a ReLU function layer, both supplied by the `torch.nn.functional` module. Dropout layers are a statistical device that randomly zero a small percentage of the neuron activations they receive, while propagating the majority on unaltered to the next layer. ReLU functions are some of the simplest and most common nonlinear additions neural nets, zeroing negative activations while allowing positive activations to propagate as usual. The utility of both kinds of layers in certain problem spaces is documented in the literature (for a different application, see (Dahl et al., 2013)), and is not the study of this thesis. The last property we will note about non-

2.1 Initial Trial: Real Molecules

linear layers is that some types, especially dropout layers, are typically switched off post-training when backpropagation is no longer taking place. This was handled automatically by calling the PyTorch `.eval` method. For the backpropagator, I used the AdamW optimizer in lieu of basic SGD. AdamW is one of the latest in a series of optimization functions intended to improve convergence to the global minimum, and was selected by trial and error throughout the experimentation process. The theory and mechanics of AdamW are explained in depth by its creators in (Loshchilov and Hutter, 2017) and do not pertain to this thesis beyond the level of documentation.

Lastly, in order to obtain labels for the dataset, restricted Hartree-Fock was run using the PySCF `scf.RHF` method. This is the most basic and least accurate form of Hartree-Fock, and was selected for its simplicity as a starting point. In restricted Hartree-Fock, or RHF, two major approximations are added on top of the baseline HF assumptions discussed in section 1.1.2. First, all spatial orbitals are assumed to be doubly occupied. Thus, RHF is a “closed shell” method, and is compatible only with molecules of even total nuclear charge. This limited the QM7 database, as out of all 7,165 molecules included, only 5,601 had even nuclear charge. Variations of Hartree-Fock that do not feature this approximation are significantly more complex, and are usually referred to explicitly as open shell methods. The second RHF approximation differentiates the method from unrestricted Hartree-Fock, or UHF, which is also closed shell. While both methods deal only with pairs of same-shell electrons with opposite spin — referred to as α and β electrons in the documentation — UHF does not require that the pair share a single spatial wavefunction. This

2.2 Secondary Trials: Random Molecules

allows for some extra degrees of freedom in the resulting Slater determinant which can lead to higher accuracy, especially in cases where magnetic contributions to the Hamiltonian become non-negligible. However, this increased capability again comes at the cost of added computational complexity. Lastly, the selected basis for all PySCF calculations in this thesis was STO-3G, meaning atomic spatial orbitals in the LCAO molecular orbital basis were composed of a linear combination of three primitive Gaussian functions. For example, Eq. 2.1 shows the usual form of an S-subshell orbital in the basis.

$$\langle \psi_s | r \rangle = \sum_{i=1}^3 c_i \left(\frac{2\alpha_i}{\pi} \right)^{3/4} e^{-\alpha_i r^2} \quad (2.1)$$

c_i and α_i are variables used to distinguish between shells, and are generally tabulated beforehand. STO-nG in general is a common scheme for constructing tractable LCAO orbitals, with n referring to the number of primitive Gaussians allotted to each orbital. Of these, STO-3G is the most common in the literature, which is why it was selected for this thesis.

2.2 Secondary Trials: Random Molecules

Following implementation of the first neural net on the QM7 dataset, several natural routes were available to extend research further. A handful of nets were trained by hand in the same manner as the initial net, attempting to predict PySCF results on a variety of molecule sets, including both randomly generated molecules and QM7 subsets. These, in general, had little success, but this is most likely due in part to the fractional amount of time spent probing their respective hyperpa-

2.2 Secondary Trials: Random Molecules

parameter spaces when compared to the initial trial¹. However, these avenues were not explored rigorously enough to be included in the main body of the thesis, and are left as possible future extensions to the project.

The reason for the abandonment of these trials was the increasingly apparent intractability of manual hyperparameter optimization when training a large number of nets. This is a common issue in the field of machine learning. On one hand, hyperparameter fine tuning regimens can multiply the computational drain of the already intensive training loops. On the other hand, correct choice of hyperparameters can be extremely important in deciding the success or failure of a particular net. Minor shortcuts to manual optimization do exist, most importantly via the manual “pruning” of nets during training. Pruning refers to the practice of cutting off the training process early for nets that do not appear to be converging effectively to a global minimum, which can in theory save a great deal of time. On the other hand, it requires tedious personal monitoring of the training process and a good deal of subjectivity. Since larger nets can take several days to train, effective manual pruning like that employed in the first QM7 trial is not generally tractable for large projects.

There is no perfect solution to this dilemma, but the fledgling practice of automatic hyperparameter optimization can remove at least some of the human labor and subjectivity from the equation. Automatic hyperparameter optimization is exactly what it sounds like, and can be thought of as a sort of second-order ma-

¹See Section 4.1 for a brief discussion of one of these, where a net was trained to predict diatomic separation curves.

2.2 Secondary Trials: Random Molecules

chine learning. In general, an automatic hyperparameter optimizer will establish a multi-dimensional hyperparameter space, each point in which represents a neural net with unique hyperparameters — though importantly, the input and output layer sizes must necessarily remain unchanged. It will then implement some predetermined algorithm to train and test nets represented at a variety of points within this space, pruning unpromising trials as it goes. Several hyperparameter optimization suites exist, but like many machine learning packages, tend to be geared for neural classifiers and image recognition rather than numerical tasks like Hartree-Fock. After some experimentation with several such packages, it was deemed more efficient to code a basic hyperparameter optimizer tailored specifically for quantum chemistry than to attempt to modify existing packages for the purposes of this thesis. The remaining three nets discussed in the main body of this thesis were trained using this homemade program, which was designed to be as general as possible. I now offer a high-level viewpoint of its functionality.

2.2.1 Custom Hyperparameter Optimizer

The custom hyperparameter optimizer is a suite of Pandas- and NumPy-based scripts which facilitate a general-purpose interface between datasets, net initializers, and training loops. To begin, it's helpful to establish some terminology. In analogy with the parameter-hyperparameter distinction, I will refer to a *trial* as the training of a single net, whereas the hyperparameter optimization of many nets for a particular task will be called a *hypertrial*. Similarly, a batch is a set of training data points that the backpropagator handles simultaneously, while a *hyperbatch* denotes a small collection of nets whose trials will be handled simultaneously by the opti-

2.2 Secondary Trials: Random Molecules

mizer.

The program is designed to handle arbitrary datasets, and takes them as a high-level input. These are deposited into the Datasets folder, and must contain a csv file with a special name containing a list linking label files to corresponding data points. Hypertrial data is stored in a separate folder, and each must have its own directory containing files which mark the desired task, training set, and “transcendental” parameters for the run. Transcendental parameter is a term created specifically for the code of this thesis to avoid convoluted variable names, and refers to a property that must remain fixed over the runtime of a hypertrial, though some may be re-selected in subsequent runs. In a certain sense, they are the “hyper-hyperparameters” of the hypertrial.

Transcendental parameters are stored in a Pandas-readable csv file in the hypertrial folder, and are integral to the hyperparameter optimizer’s performance in much the same way that hyperparameters are important to individual neural nets. The most important transcendental parameters to touch on for this thesis are those pertaining to neural net shapes, sizes, and training durations. Shape and size are broken down into three independent properties that can be used to fully specify a net: number of neurons, number of layers, and pattern. Each of these can either be given a constant value or a link to a selection function in a separate file in the transcendentals csv. If given a function link, the net initializer will call that function before building each net in the hypertrial to determine what value this property will take for that particular trial. It is important to note that while larger nets have

2.2 Secondary Trials: Random Molecules

in general a higher ceiling of accuracy on most tasks than smaller nets, this is not necessarily the case if we restrict the training duration. Larger nets have larger parameter spaces over which to optimize, and may converge slower over the same training data compared to a smaller net for certain tasks. This can be either an advantage or a disadvantage depending on the goal of the programmer. One seeking accuracy might put no ceiling on the training duration, or allow dynamic allocations through the `transcendentals.csv` by linking a function that allocates a certain number of training batches to each net that is proportional in size to its parameter space. On the other hand, if computational resources are valuable, one might wish to put a hard cap or static value on training duration regardless of net size, and force the algorithm to find the most accurate possible net within a compact time-frame.

The pattern variable (in the code, the name “shape” is used, but we have avoided it so far for clarity) is slightly unique. At the time of writing, the program contains functionality to produce four different net shapes: ribbed, rectangular, diamond, and bottle. Figure 3.1 gives a visual example of a rectangular (NN35-4), a ribbed (NN0-2), and a bugged version of a bottle (NN43-0) architecture. Any selection function associated to the shape variable in the `transcendentals.csv` is responsible for populating a vector whose elements represent the “desirability” of each different pattern type, based on the performance of previous nets in the hypertrial. The vector is then normalized, and the fractions in each slot are interpreted by the net initializer as probabilities to choose a particular pattern. If future programmers ever return to the program’s code, the addition of more patterns and/or hybrid-

2.2 Secondary Trials: Random Molecules

pattern functionality would be a relatively straightforward improvement to the code.

The other transcendental parameters currently supported by the program are choice of loss function, error metric, input and output layer size, dataset (from the Dataset folder), optimizer, batch size, hyperbatch size, resolution, and training duration. Training duration is a very important transcendental, which as discussed above can be adjusted to be dynamic or static depending on the programmer's intentions. Batching is briefly explained at the start of this section, and batch size is a standard hyperparameter for all training functions. There is a wide body of literature on the subject of batching and the theory behind it for interested readers (see (Rumelhart et al., 1986) for a seminal paper touching on many neural network fundamentals), but I will not discuss it further here. Moving on, resolution is an arbitrary parameter used for monitoring hypertrial progress and hyperbatch size and will be touched on briefly in the training subsection. Dataset, loss function, and error metric are all self-explanatory and should not be dynamically assigned, although nothing technically prohibits one from doing so. This leaves input and output layer size. These must be constant throughout the hypertrial, else nets will not be comparable. They are only included in the transcendentals csv to provide access to their values for the net initializer.

Hypertrial Initialization

Once the dataset is properly processed and stored and the hypertrial specifics are filed away, the actual running of the hypertrial is initiated from a third folder.

2.2 Secondary Trials: Random Molecules

This folder contains all the main task management scripts for the program, and should not need to be modified for most applications. In addition to some basic test scripts which ensure that hypertrial and dataset information has been properly loaded, this folder contains an initialization script, a kernel script, a net configuration script, and a training loop template. When booted up, the initialization script should be called, where the hypertrial name is input by the user and the hypertrial and dataset folders are checked for proper formatting. From there, a copy of the transcendental csv is pulled in to the local directory. After all this, the kernel script is called, which will function as the central manager for the subsequent duration of the hypertrial. Essentially, the kernel dictates the interplay between net initialization and the training process, as well handling data records of the hypertrial via a Pandas dataframe.

Hypertrial Training

A hypertrial in the program consists of an iterative training of a fixed number of hyperbatches, set to fifty as a standard. Dynamic testing and hypertrial termination may be another straightforward next step for the program which was not included in the current version for timing reasons. Each hyperbatch contains a number of nets with unique hyperparameters, determined by the net initializer and hypertrial transcendentals. Though not mentioned when initially describing transcendental parameters, it is important that all functions associated to parameters via the transcendentals csv depend only on information contained in the log dataframe created by the kernel, as this will be their only input. For all the nets trained in this thesis, size variables, batch size, and training duration were selected

2.2 Secondary Trials: Random Molecules

```
import numpy as np
import pandas as pd
import random
import math

def gradient_descent(df, err, lr):
    current = df[-1]
    curErr = err[-1]
    change = current
    length = math.floor((len(err)-1)/2)
    for i in range(length):
        val1 = current-df[-1-i]
        if val1 != 0:
            val1 = 1/val1
            val2 = curErr - err[-1-i]
        else:
            val2 = 0
        change = change-val1*val2*np.random.rand(1)*lr/length
    output = change
    return output
```

Figure 2.2: Code snippet of the gradient descent function. Inputs were kept general, `df` representing a Pandas Series object taken from the log DataFrame for the hyperparameter being optimized and `err` another Pandas Series representing previous net errors. `lr` is a positive float, corresponding to a parameter-specific learning rate.

using a discrete version of gradient descent, whose code is shown in Figure 2.2 and mathematics are represented by Eq.'s 2.2-2.3. Training duration alone was made not to vary within a single hyperbatch, in order to ensure compatibility with the pruning algorithm outlined in the paragraph below.

$$\frac{\partial L}{\partial x} \approx \sum_i \frac{L(x) - L(x_i)}{x - x_i} * \beta \quad (2.2)$$

2.2 Secondary Trials: Random Molecules

For the purposes of Eq. 2.2, L represents the loss function, x is the current value of the hyperparameter being optimized, and $\{x_i\}$ represent hyperparameter values from the last half of nets trained in the hypertrial so far. β is a combined parameter, representing the hyperparameter's learning rate times a random factor between zero and one and divided by the total number of indices included in the sum. For both hypertrials included in this thesis, learning rates were set to 10^{-4} for all hyperparameters but depth, which was set to 10^{-5} . This was done based on initial experimentation with the code, where layer counts rapidly outpaced the neuron counts needed to adequately populate them. Hyperparameters were then chosen via addition of discretely approximated partials to current variable values:

$$(a', b', c', \dots) = (a, b, c, \dots) - \left(\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}, \frac{\partial L}{\partial c}, \dots \right) \quad (2.3)$$

In the usual manner of gradient descent. Generalizing the function in Figure 2.2 to handle vectors and applying it to the shape hyperparameter is another straightforward addition to the code for a future programmer; shape was randomly selected for the hypertrials discussed in this thesis. Finally, hyperbatch size was set to five and resolution to fifty, although this was based on no real theory beyond the general utility of round numbers.

Between each hyperbatch training, the net configuration script creates serial-numbered files containing the particular Net class for each trial based on selected hyperparameters, and pastes them into a copy of the training loop template. When this

2.2 Secondary Trials: Random Molecules

is completed, the net configuration script terminates, and the kernel resumes. Its next step is to run each of the newly-created training files one by one, training over N batches before swapping, where N is the value of the resolution hyperparameter. This process is continued until the hyperbatch is fully trained according to training duration specifications. Upon completion, the kernel logs relevant data and stores the training files in a repository within the hypertrial folder for future reference.

The last part of the training phase that needs to be touched upon is the automated pruning algorithm. To reiterate, pruning is the discarding of nets that have become stuck in local minima or are otherwise not converging to a desirable accuracy in a reasonable time frame during the training process. Since training is, in general, a time-intensive process, automated hyperparameter optimizers that can train hundreds of nets in a single run require a method for recognizing trials likely to end in failure before wasting a full complement of resources training them. In this program, pruning is handled at a high level by the kernel script by gathering a cluster of nets — a hyperbatch — and training them concurrently in competition with each other. For a hyperbatch of K nets, the total training duration for each net is divided into $K-1$ equal sections. As nets train in the manner outlined above, the kernel prunes the worst-performing net after each of these sections. This is determined via loss function on the training data to keep the testing data pure and avoid overfitting. In this way, after $K-1$ pruning iterations, all but one net in the hyperbatch is left as the “winner”. As K approaches infinity, and assuming equal training duration for each net, this pruning scheme will result in a halving of the

2.2 Secondary Trials: Random Molecules

total runtime of the program.

2.2.2 Random Molecules

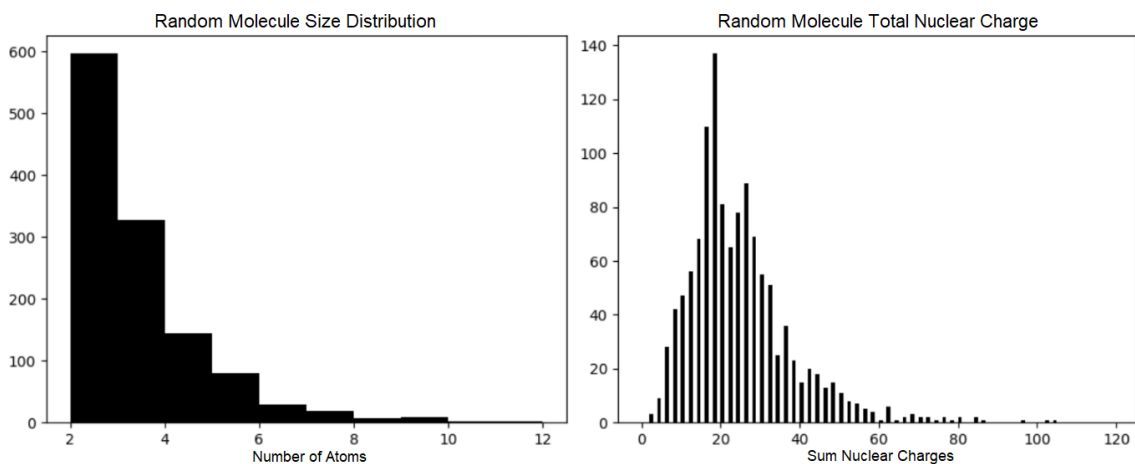


Figure 2.3: The randomly generated test set, post-processing. As we see there was a quick dropoff in convergence rates as molecules scaled with length, leading to a skew in the overall dataset. Additionally, the closed-shell constraint made only even atomic charge molecules valid for the hypertrials.

Both of the hypertrials subsequently discussed in this thesis were trained on a set of randomly generated “fake” molecules rather than the QM7 dataset. This technique served two purposes: 1) using computer generated molecules allows for arbitrarily large training sets, and 2) random molecules of a given size represent a more general problem space than specially selected real molecules. Molecules were generated in an analogous format to the QM7 xyz and nuclear charge matrices, with a randomly chosen molecular length (number of nuclei) between 2 and 24 and a randomly chosen atomic number for each nucleus between 1 and 17. These files were then saved in exactly the same manner and notation as the QM7 files so that they could be processed using pre-existing code. For the purposes of the long-

2.2 Secondary Trials: Random Molecules

term project, three label types were generated for each molecule in the dataset: electronic (SCF) energy, CISD correlation energy, and electronic RDMs. Training was not completed on the last of these within the time frame of the writing of this thesis, although it is an ongoing project and a natural extension of the work outlined here.

PySCF for Random Molecules

The PySCF method chosen to analyze these molecules was UHF, or unrestricted Hartree-Fock. As discussed in section 2.1.1, this allows for an extra degree of freedom in for simulated spin-orbitals which can lead to a higher degree of accuracy, although it does not lift the closed-shell constraint. Thus, random molecules generated were forced to contain an even total nuclear charge during the generation process. In addition, the CISD post-HF program was utilized to calculate the partial Coulomb correlation energy for all of our randomized molecules, which is presented as a separate correction term which can be added to the underlying SCF energy for increased accuracy.

Unfortunately, this came with a major caveat. Since we were only looking for SCF energy estimates from our QM7 net, it was acceptable to settle for a non-converged RHF energy when calculating labels. This was not generally ideal, as it means training took place on less-than-accurate energy estimates, but for the purposes of recreating PySCF's performance it was sufficient. Additionally, convergence in UHF and RHF is relatively rare over the datasets tested, both real and randomly generated, and thus on the already restricted QM7 dataset there simply would not

2.2 Secondary Trials: Random Molecules

have been a sufficient number of converged energies to train a neural net on. To rectify this and generate a fully converged dataset, each random molecule had to be created, have the UHF procedure run on it, and then checked for convergence before proceeding. Not only did this process take an incredible amount of time and computational resources to reach a final molecule count of 8,010 (originally planned to be 10,000, the program was cut off due to sufficiency and time constraints), but it also skewed the composition of molecules that made it through to the dataset considerably. Figure 2.3 shows the distribution of molecule lengths in the final dataset. Note that the molecule generator script produced molecules of random length within the target interval, and so without the convergence criterion, the distribution should be roughly flat. Instead, since smaller molecules have exponentially less complex electronic structures than larger molecules, they were proportionately more likely to converge to a stable value under Hartree-Fock. Thus, we see a large skew towards smaller molecules in the finalized dataset.

Neural Nets for Random Molecules

Once the random dataset was created and labelled, training could commence. Overall, 6,792 molecules were trained on, leaving 1,218 for the testing partition, for a total percentage split of 85/15. Fifty hyperbatches of five nets a piece were trained and tested for each of the hypertrials. One predicted the SCF energies, and the next predicted the CISD correlation energy corrections. Both took as input only the raw nuclear position and charge matrices, combined and flattened into vector form. Each used only AdamW as an optimizer, and MSELoss as both a loss function and error metric. The other relevant transcendental parameters — dataset,

2.2 Secondary Trials: Random Molecules

input size and output size — were necessarily predetermined and static. For final results, each trained net was fed all 1,218 molecules in the testing partition, and average and median errors compared to PySCF labels were recorded and analyzed. The two nets in each trial with the lowest mean and median errors over the test set, respectively, were then selected for further analysis. These are the nets referenced by serial number in the data, and are the subject of all relevant analysis presented in this thesis.

Chapter 3

Results and Analysis

3.1 Initial Trial: Real Molecules

In terms of error percentage, the most successful net produced in this thesis was the initial QM7 manually-trained 500 x 250 hidden-neuron learner. This learner was trained against PySCF's RHF energies on 4,481 of the 5,601 RHF-compatible QM7 molecules for standard training/testing split of 80/20. On the testing subset, the fully trained net mimicked PySCF energy outputs with an average error of 12.62% and a median error of 10.70%. It is important to note here that the vast majority of molecules did not have convergent energies under PySCF, and so the PySCF energy errors compared to actual chemical energies for these molecules are likely high, even if we restrict our comparison to the mean-field energy limit. This claim is justified by the fact that non-converged energies are generated by PySCF by printing out whatever the current calculated energy for the molecule is after 50 cycles of the iterative process. Since this energy is by definition not converged, subsequent cycles will not, in general, agree. Overall, this is to say, we expect the

3.1 Initial Trial: Real Molecules

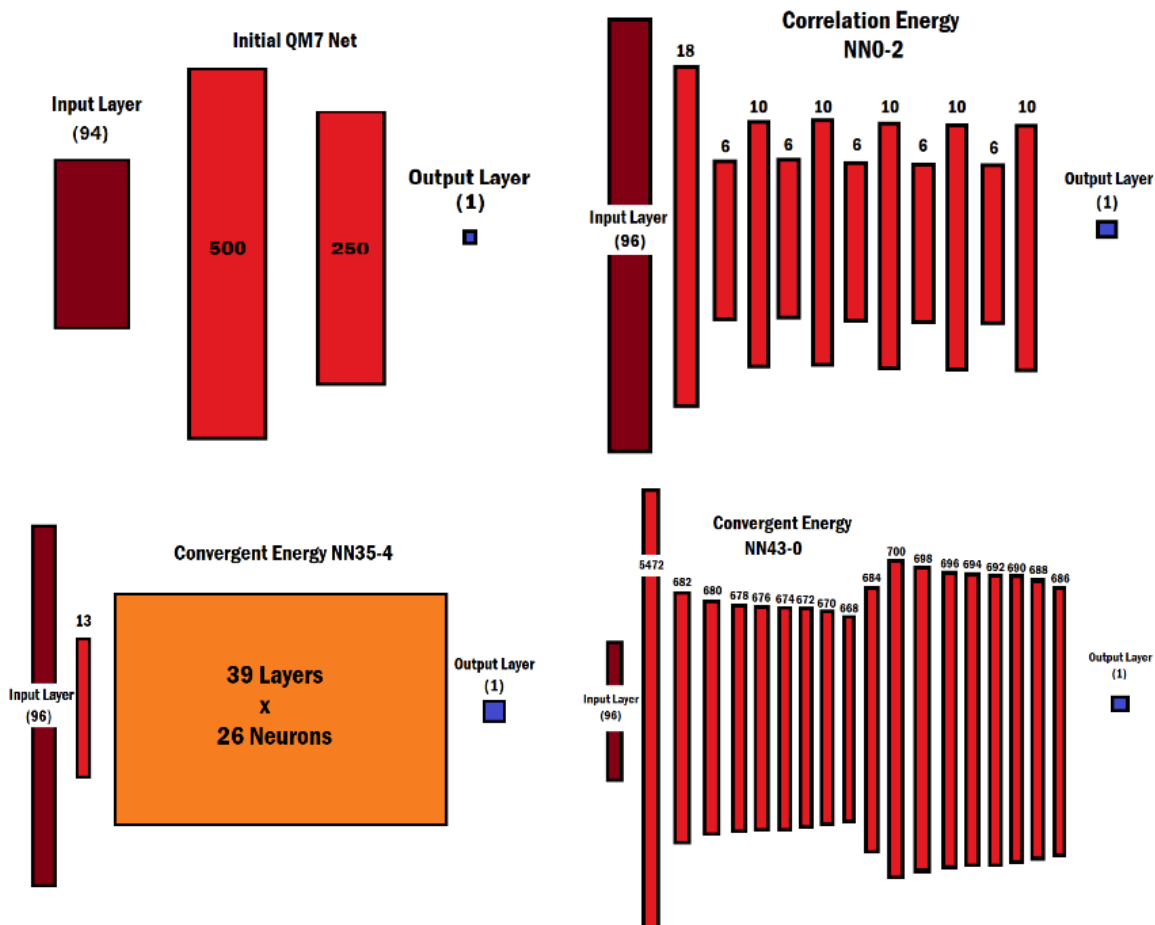


Figure 3.1: Net architectures for all four nets discussed in this thesis.

10.70% median error to be significantly diminished, if not negligible, when compared to the PySCF non-convergent HF error margin for the data which the net was trained. Numerical analyses of the exact magnitude of such PySCF-specific internal errors were passed over for this thesis in favor of establishing a broader task domain for our nets, but will form an important part of long-term study as net fidelity approaches a maximum.

In addition to reproducing RHF results, the trained net offers two distinct advan-

3.1 Initial Trial: Real Molecules

tages over explicitly coded PySCF algorithms. The first advantage is the drastic reduction of comparative CPU utilization. We see from Figure 3.1 that the net required less than one half of one percent of the total CPU utilization time of PySCF on the vast majority of QM7 RHF-compatible molecules. Numerically, this distribution corresponds to a mean reduction in CPU utilization of 99.81%, with a median reduction of 99.89%. The mechanics of this speedup are fundamental to neural nets, and are a major reason why some expect machine learning to challenge the efficiency of quantum techniques. Put shortly, neural networks front load their drain on computational resources during the training phase. Once the complex, time consuming backpropagation routine has calculated the many-dimensional gradient and updated parameters accordingly, forward propagation in the processing of new data involves only the basic operations of matrix algebra to determine net output. So, if even a single researcher is able to train a net with acceptable fidelity and a computational speedup compared to traditional methods, all he or she must do is to publish the parameter state file of the fully trained net, and all future researchers will be able to bypass the initial costly training phase forever. Thus, the front-loaded resource drain of training a neural net becomes negligible if the net is successful in the long-term, although net success is by no means guaranteed during training.

The second major improvement over traditional code is the versatility of the neural net in terms of molecule domain space. Whereas the PySCF RHF algorithm could not be run on all 7,165 molecules in the QM7 dataset, neural nets can in theory produce outputs based on any input that is vectorizable and of the proper size. Thus,

3.1 Initial Trial: Real Molecules

the trained net was able to give energy estimates for all molecules in the dataset,

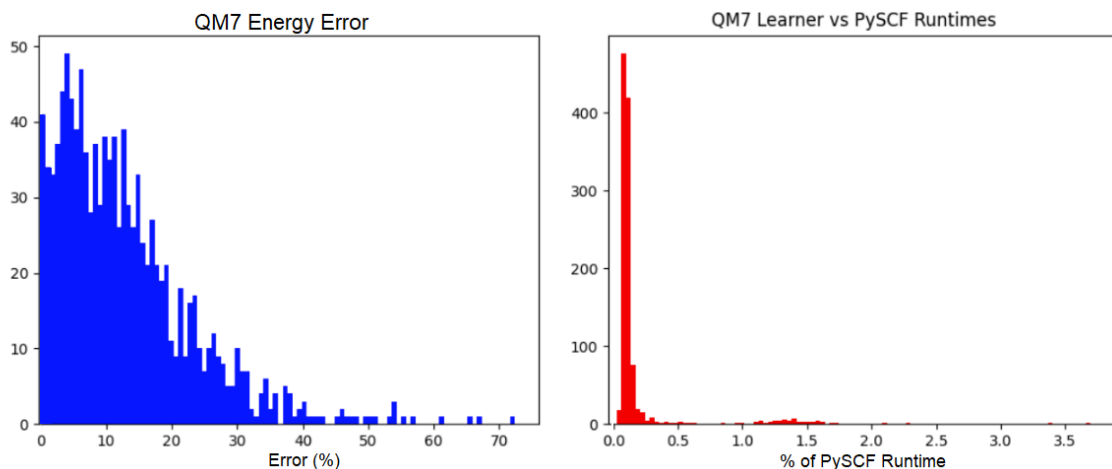


Figure 3.2: Compiled data on the QM7 neural net performance in comparison to the PySCF RHF method. Data was taken for both methods across the full testing partition, and CPU runtimes are computed using the `process_time` python builtin.

although no error metrics could be established as the extra molecules could not be given RHF energy labels. Extrapolations like this are not generally as trustworthy as the results cited explicitly in this paper, but can often be useful approximations when other methods are too costly or difficult. One particularly example application for neural net extrapolation is to assess nonphysical systems, like molecules with fractional nuclear charge. Physicists have a long history of utilizing nonphysical systems to model phenomena they cannot simulate precisely, and fractional nuclear charges have direct applications for electronic structure problems involving partial Coulombic shielding (E. and D.L.), (E. et al.).

One last thing worth noting about the net trained in this trial is its runtime invariance to molecular input size. A small trend towards longer runtimes versus nuclear charge sum is visible in Figure 3.3, but this seems to simply be a reflection

3.1 Initial Trial: Real Molecules

of the RHF-compatible distribution of Figure 2.1. Since propagation is at its core a sequence of matrix algebra where matrix dimensions are predetermined by the net's architecture, we would not expect to see any significant difference in runtime based on molecule sizes. This, however, is not the case for explicit methods like PySCF. As can be seen in Figure 3.4, RHF runtimes scaled almost linearly with both molecular length and total nuclear charge. This makes sense given the increased complexity of the Hartree-Fock method for larger molecules; in fact, with a larger sample size, we would expect the trend to appear increasingly exponential. This represents another advantage of neural nets, but is partially undercut by the hard cap placed on input molecule length for our net.

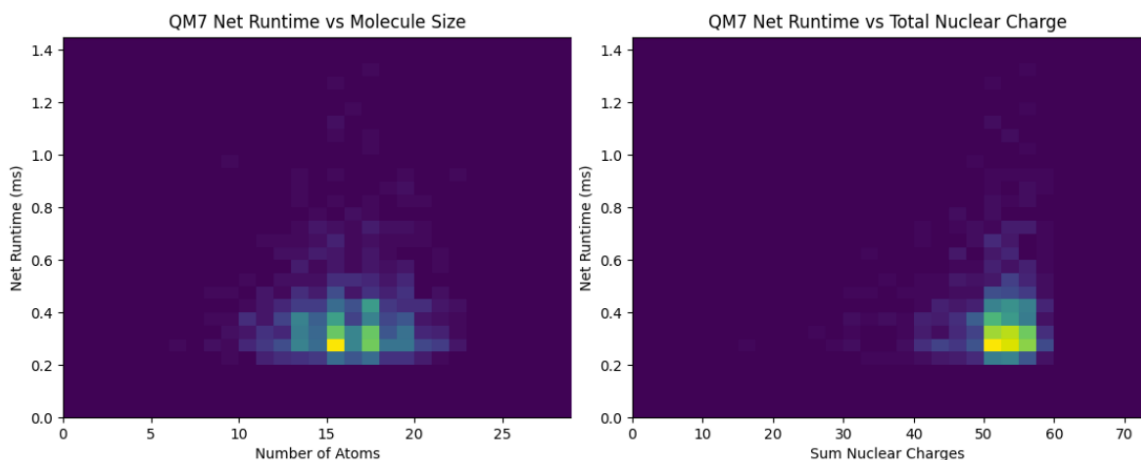


Figure 3.3: A set of 2D histograms comparing neural net CPU runtimes with molecular size metrics.

The QM7 net was explicitly tested against PySCF on one extrapolation: the dissociation curve of diatomic fluorine. Fluorine was selected since it had enough electrons to have interesting structure, while still being light enough to have its

3.1 Initial Trial: Real Molecules

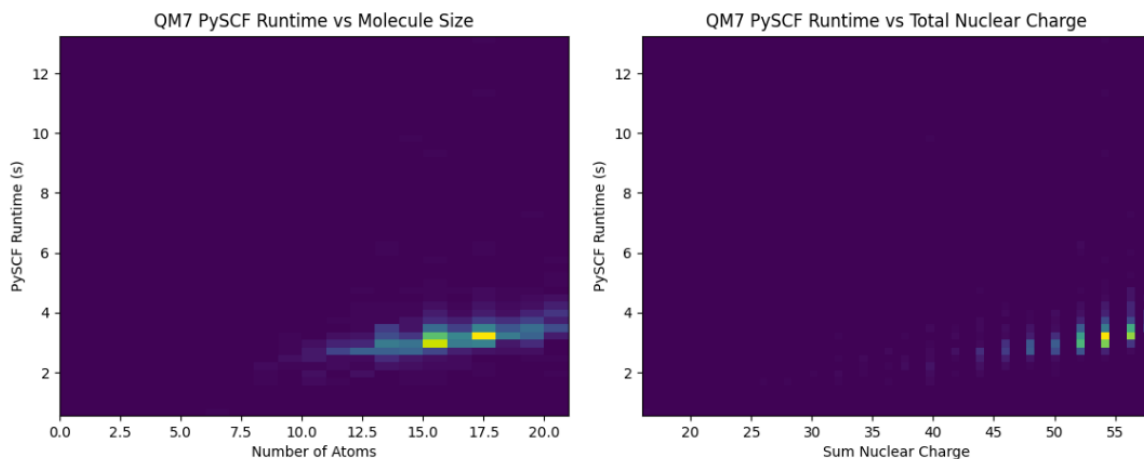


Figure 3.4: A set of 2D histograms comparing PySCF RHF CPU runtimes with molecular size metrics.

ground state included in the QM7 set. This, however, still represents an extrapolation from the training data, since the net was not trained on partially dissociated molecules. Side-by-sides of the PySCF and neural net curves are shown in Figure 3.5. Clearly, the net does not extrapolate well to dissociation curves. This is a major

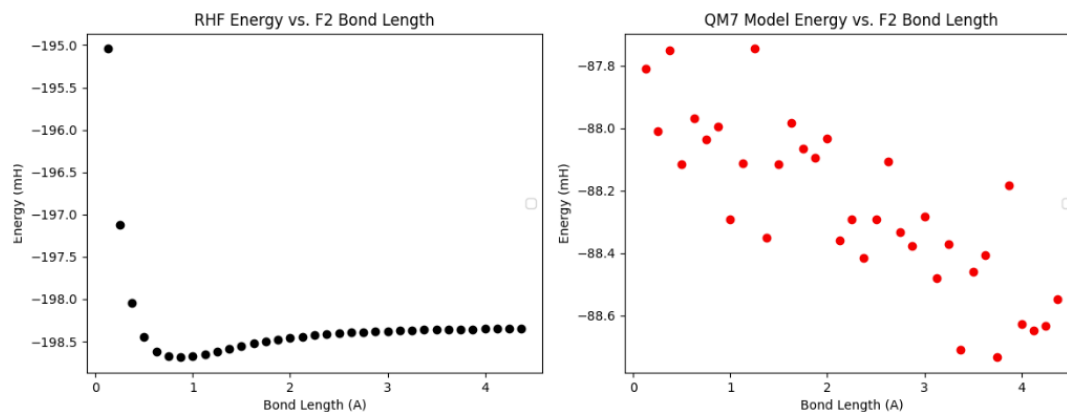


Figure 3.5: PySCF RHF vs the QM7-trained net for F_2 dissociation curves.

blow to its utility, as simulation of dissociation curves is an important benchmark for quantum chemical simulations. Interesting to note is the fact that F_2 appears to

3.2 Secondary Trial: Random Molecules

be a molecule with particularly high error for the net: the ground state, calculated by PySCF to be at around an Angstrom of nuclear separation, is included in the domain space of the graph, and so not all of the $> 50\%$ error can be blamed on extrapolation of the training data. Lastly, we see an important limitation of this kind of neural net on clear display. Specifically, a simple MSE loss function does not incentivize explicit incorporation of conceptual physical properties, like conservation of energy and the tenets of continuous deformation. Explicit methods like PySCF, whatever their inaccuracies, have certain of these properties hard-coded into their methodology. Thus, we see a fairly clean looking dissociation curve representative of a continuous function on the left side of Figure 3.5, while the neural net produced a shotgun-like pattern optimized for average accuracy over physical fidelity. Possible fixes to this are discussed in the Appendix, but as it stands, the difficulty of getting neural nets accurate enough to model phenomena like dissociation curves that require a grasp discrete physical principles represents a major advantage for explicit methods.

3.2 Secondary Trial: Random Molecules

3.2.1 Convergent Energy Trial

Out of all trials in the random molecule convergent UHF energy hypertrial, the nets with lowest mean and median errors were nets NN35-4 and NN43-0, respectively (notation of NN[hyperbatch]-[net number]). The error distributions over test molecules for each of these nets are shown in Figure 3.6, alongside CPU utilization relative to PySCF. As we see, the histograms of both nets share several key

3.2 Secondary Trial: Random Molecules

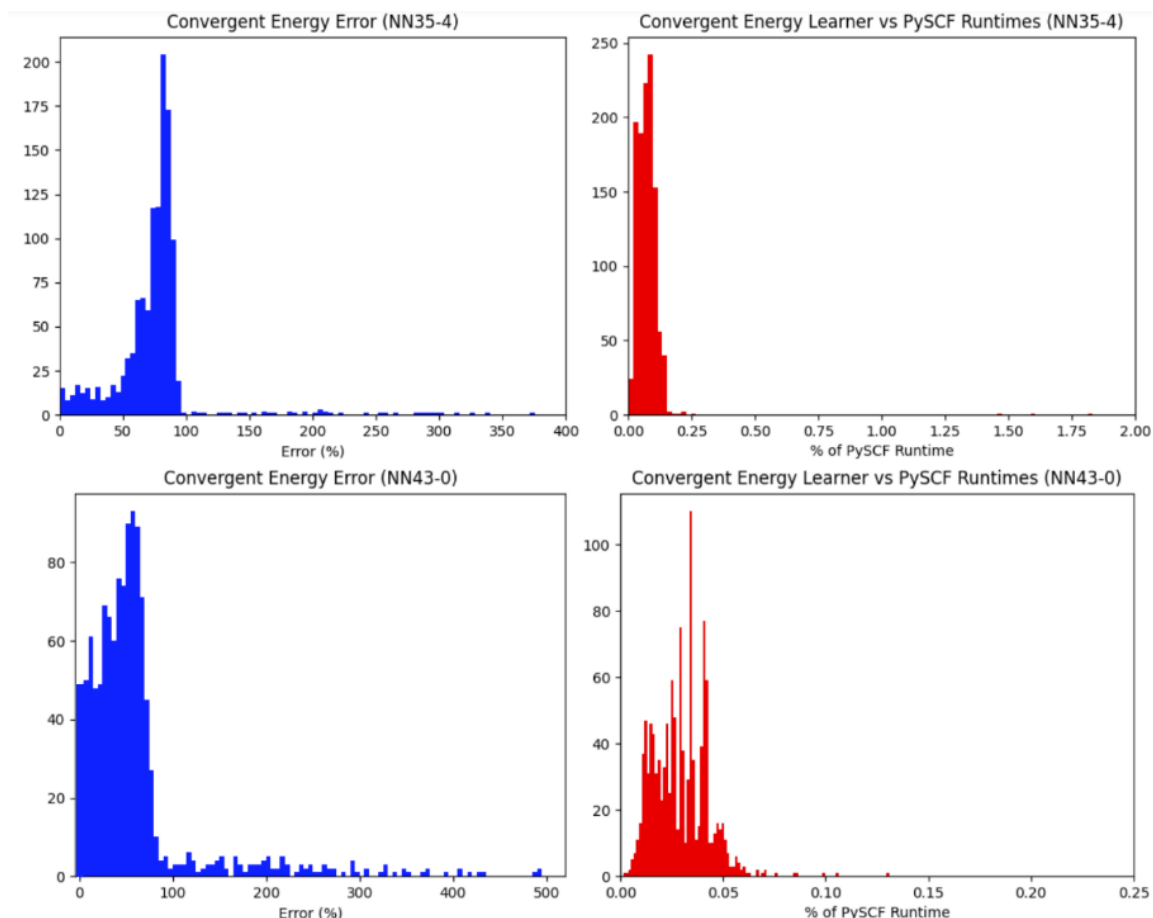


Figure 3.6: Compiled data for the convergent UHF energy neural nets of lowest mean error (NN35-4) and lowest median error (NN43-0) respectively. Data gathering techniques are identical to those used for the QM7 net.

features. I will begin by discussing the error distributions.

The first feature to note is that both nets produced error distributions with large peaks at slightly below 100% error. These are followed in both cases by long tails of sharply diminished magnitude spreading out several hundred percentages in the positive direction. This tail is responsible for a very high average error in both nets, with the lower of the two values being 98.66% average error for NN35-4. Al-

3.2 Secondary Trial: Random Molecules

though the tails do significantly skew the mean errors for both graphs, we still consider their magnitudes sufficient to render them a significant feature, and so these means are regarded as valid and not the result of outliers. Depending on the goals of a hypothetical user of these nets, these tails may be tolerable and decreased median value may be the more desirable metric. The best net in this regard was NN43-0, with median error 50.27%, significantly lower than the mean.

Analysis of the signed error of each of the nets in Figure 3.7 gives slightly more insight into their inner workings. Both nets, it seems, have been trained to be par-

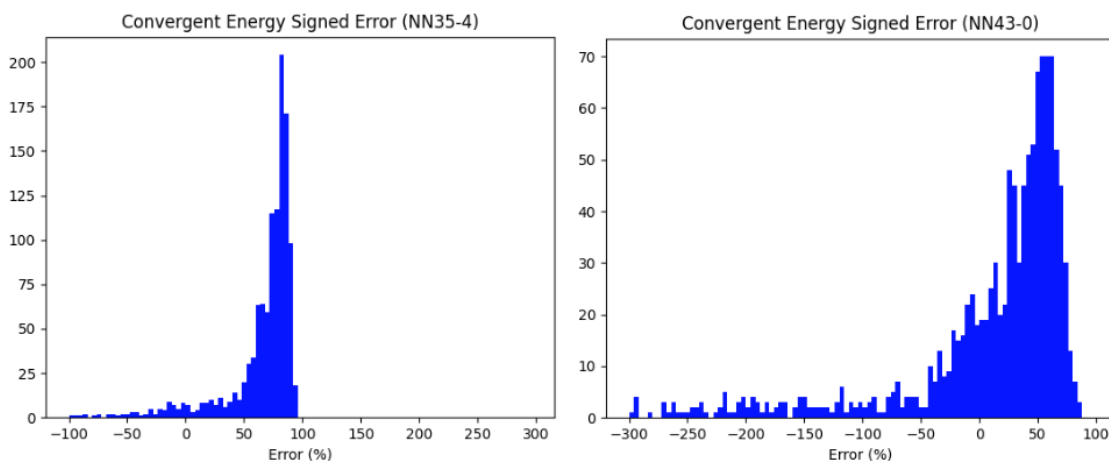


Figure 3.7: Signed error for the convergent energy UHF nets versus PySCF.

ticularly cautious in their energy estimates. Sharp peaks in signed error occur at slightly less than 100% and slightly more than 50% for NN35-4 and NN43-0, respectively. This means that, especially in the case of NN35-4, the net nearly always guessed energies of negligible magnitude when it was unsure. This was a common local minimum to fall into when training the initial trial by hand: a net that always spits out zero will often have better mean error than a net than a poorly trained

3.2 Secondary Trial: Random Molecules

net that “takes more risks” so to speak, even though the task has not really been accomplished. It is difficult to say whether NN43-0 was in the process of entering or exiting the same local minimum, but this is clearly a large area of concern that a future iteration of the optimizer will have to overcome. In the initial QM7 trial, promising nets that fell into this minimum were “shaken”, by training them on unrelated data for several batches before returning to the training set. A similar protocol could perhaps be included in automated form to the program here.

Next, we examine the relative CPU utilizations of these nets. As we see in Figure 3.2 again, the vast majority of molecules were processed in less than .05% of the compute time it took PySCF for equivalent samples. We also see from Figure 3.9 at the end of the chapter that runtimes did not show any noticeable trend as molecules scaled in size, although this is not as clear an advantage over PySCF for this dataset of particularly small molecules, as we see in Figure 3.10. It is also important to note that the molecules included in this dataset were filtered by the convergence criterion for explicit compatibility with PySCF. In other words, CPU utilization data in Figure 3.6 is actually biased in PySCF’s favor, since non-convergent molecules will by definition be processed for a longer duration in the PySCF algorithm than convergent molecules of the same size and complexity (see the previous section’s discussion of PySCF’s iterative cutoff).

3.2.2 Correlation Energy Trial

The correlation energy hypertrial produced a net with results roughly comparable to those of the convergent energy hypertrial. One particular difference to be

3.2 Secondary Trial: Random Molecules

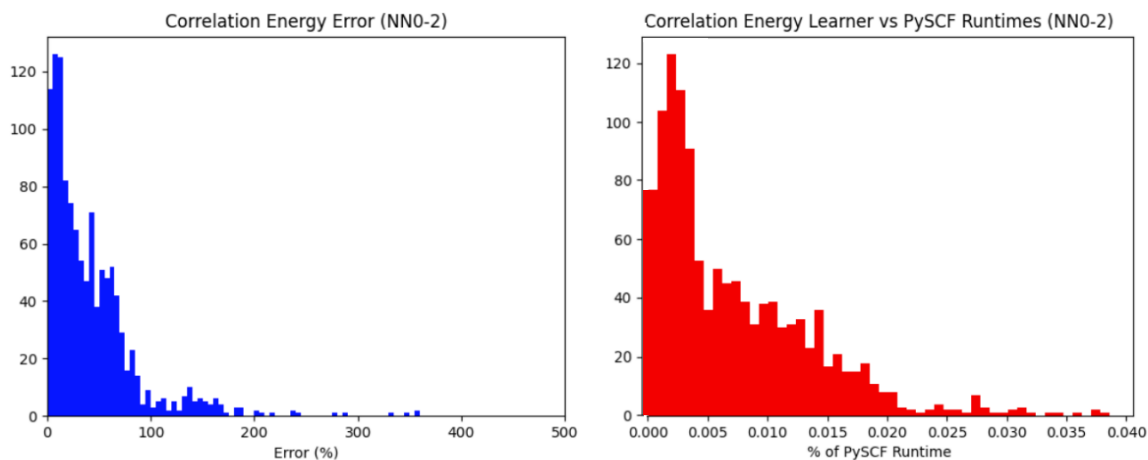


Figure 3.8: Compiled data for the correlation energy neural net of lowest mean and median error. Data gathering techniques are identical to those used for the QM7 and convergent energy nets.

noted immediately is that in this hypertrial, a single net, NN0-2, tested lowest for both mean and median errors after training. This is peculiar, and not what one might expect: 49 hyperbatches were optimized and trained after the winning net had been created, tested, and logged away, and not a single one of them produced any nets with comparable accuracy. With only 108 total hidden neurons and 2,580 parameters, NN0-2 is by far the smallest net considered in this thesis writeup.

This is explicable in two possible realities: 1) that the default hyperparameters were, by pure chance, already optimal for the task, or 2) that the hyperparameter updating algorithm to trace a path towards minima. In the second case, it may be true that net architectures rapidly overgrew their training durations through the course of hyperparameter optimization, resulting in the first cluster of more compact nets performing better under the given constraints. This is the current working theory, as similar issues were encountered during the fine tuning phase

3.2 Secondary Trial: Random Molecules

of program development, and may not have been adequately patched. In addition, it seems extremely unlikely that the first explanation should hold, especially given that calculation of CISD correlation energies is a strictly more computationally complex task than calculation of the UHF energy, and we see that the best nets from that hypertrial came near the end of the run-through, and were much larger.

We also see from a visual analysis of the figures that, much like the two successful convergent UHF energy nets, the correlation energy net has a strong left skew in its errors. It also shares the long but non-negligible tail feature, trailing off along several hundred percentages. In fact, the full length of the correlation energy net's error tail is not represented in Figure 3.8. Because of the nature of PySCF CISD calculations, which give the relatively small energy corrections to UHF rather than the energy sum, label energies for the correlation hypertrial are occasionally identically zero. These extreme cases had to be screened out during error calculation to avoid divisions by zero. However, molecules remained in the test set with arbitrarily small, nonzero correlation energies. For these types of molecules, random statistical differences with the calculated PySCF correlation energy due to imperfections in the net were magnified in the percent error calculation by almost-vanishing denominators, giving extraordinarily large outlier errors. Because of this, the actual mean value for error on the net — not including cases of zero correlation energy — was 72,640%.

In these cases, statistics tells us that medians are the preferred representation of center. Median error for NN0-2 including cases of vanishing correlation energy

3.2 Secondary Trial: Random Molecules

was a modest 30.30%. This does not approach the 10.70% median error of the QM7 net, but beats the convergent energy nets by a considerable margin. Combining these statistics with the relative compactness of NN0-2 and the computational complexity of calculating CISD correlation energies explicitly, we suggest that projections of correlation energy corrections are the most promising potential areas for neural net application probed in this thesis. Future refinements of the techniques used here might as a starting point focus on reducing the median error to less than ten percent, either by hand or through enhancements to the hyperparameter optimizer.

3.2 Secondary Trial: Random Molecules

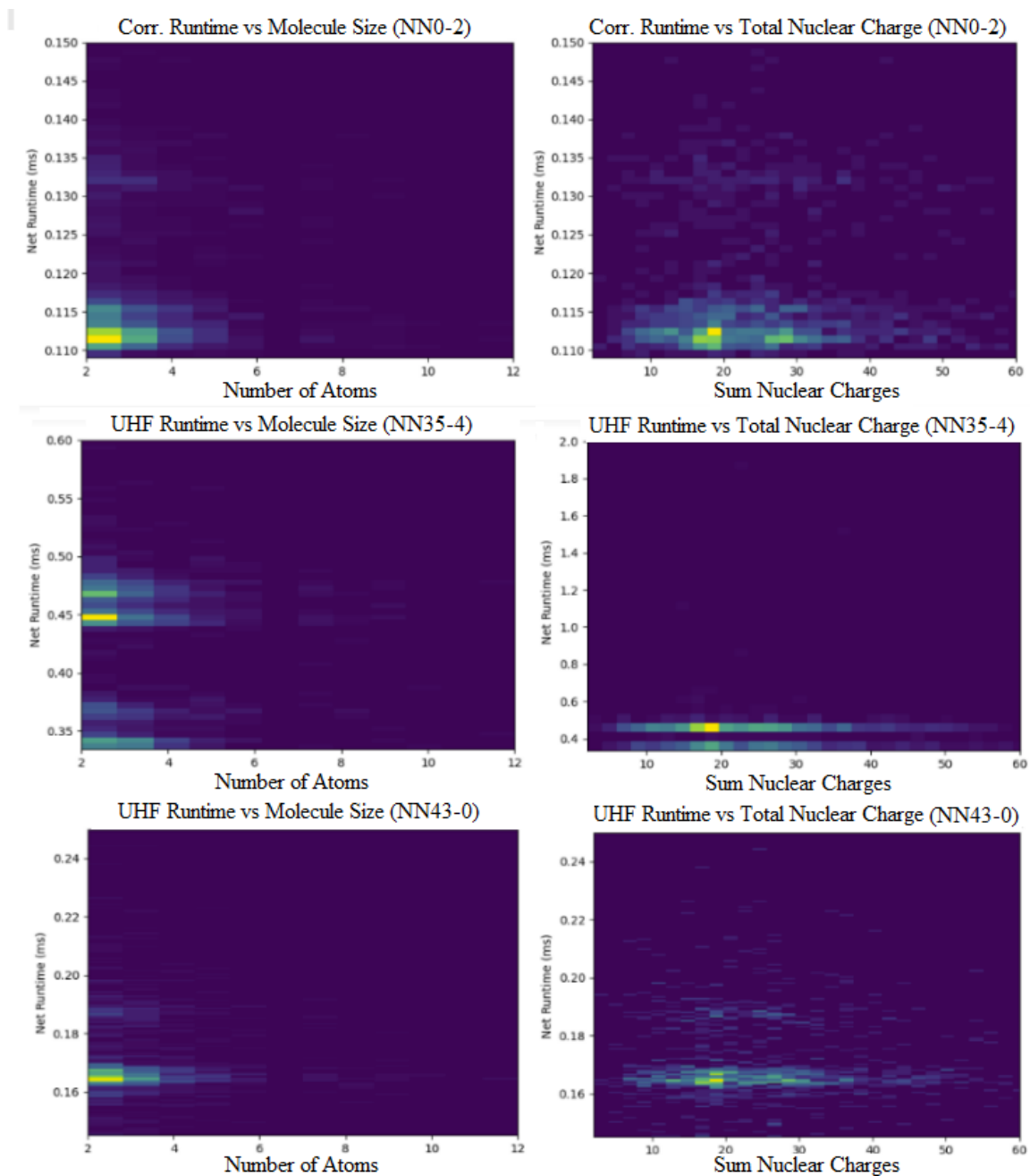


Figure 3.9: A set of 2D histograms comparing neural net CPU runtimes with molecular size metrics for each net discussed.

3.2 Secondary Trial: Random Molecules

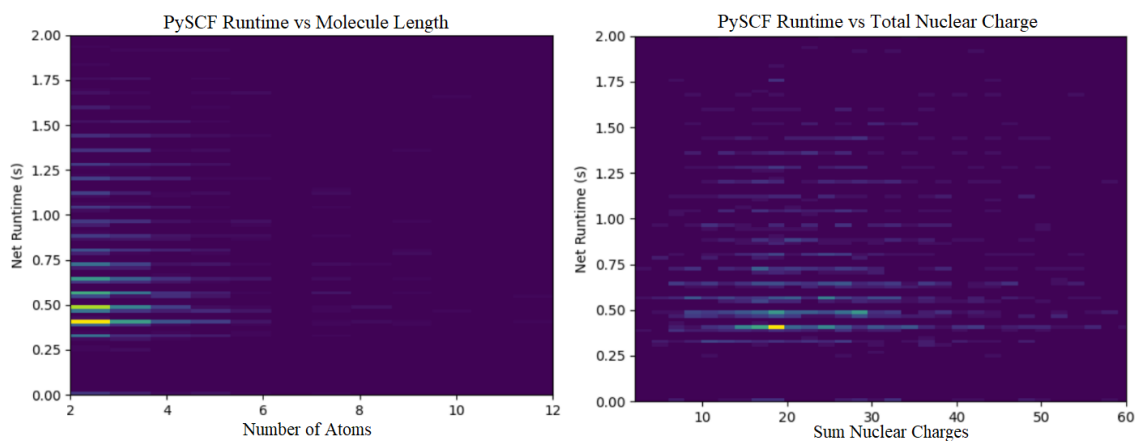


Figure 3.10: CPU runtimes for PySCF UHF method compared to molecular size.

Chapter 4

Conclusion

The data gathered in this thesis is best viewed as a preliminary exploration into a promising direction for computational quantum chemistry and performance benchmarking. In particular, data from the manually-trained QM7 net shows that neural nets are capable of roughly reproducing PySCF RHF data on small real molecules, and that this approach may be extrapolated to a far broader domain space. Further study will be needed to determine the exact accuracy of particular extrapolations, but potential applicability to nonphysical approximation cases is currently unique to the machine learning approach. CISD correlation energy prediction also shows potential as an area of near-term application, with our initial probe yielding only moderate statistical error and a surprisingly compact net architecture. Further work on the hyperparameter optimization algorithms may result in nets that whittle down this error significantly.

With this being said, two major drawbacks still diminish the applicability of neural nets to quantum chemistry. The first is the technique's inherent lack of task

Conclusion

versatility. While Section 3.1 discusses several ways in which neural nets may be able to execute tasks on more general domain spaces than their explicit counterparts, they are restricted to performing a single, narrowly-defined task every time. The QM7 net trained in this thesis may be adept at predicting the RHF energy for small molecules, for example, but it has done no explicit physical calculations, and therefore cannot give any additional information about the molecule beyond an estimated SCF energy. Compare this to PySCF: if we wish to run a post-Hartree-Fock method after performing initial HF calculations, we can do so without starting from scratch. We can also generate the one, two, three, and four electron reduced density matrices, orbital occupations, and a variety of other characteristics, like dipole moment. All this is done with reduced computational overhead using data borrowed from the initial calculation. For neural nets, each one of these tasks would require an entirely new net and training session.

The second area where neural nets falter is related, and has to do with counterfactuals. If we were to use a neural net to estimate the energy of a diatomic molecule, it might reliably produce a value within a small range of the PySCF prediction, and require significantly less computational overhead in the long term. However, if we were to slowly pull the two atoms apart, and use each method in turn to predict molecular energy at a number of points along the process, we would see the neural net's error randomly distributed about each point sampled. This holds true even for nets trained on partially dissociated molecules; straightforward loss functions only explicitly minimize statistical error averages, and do not contain provisions to decide in what manner these errors manifest across the data. This is

Conclusion

the reason for their susceptibility to local minima like the self-zeroing of NN35-4. Thus, unless the average error in the net was extremely low, we would be left with a rough dissociation curve traced out by the PySCF predictions surrounded by a chaotic haze of guesses created by the net. Each of these predictions may be fairly accurate, but the net's lack of understanding of the underlying physical principles renders it incapable of recognizing important conceptual phenomena like energy conservation and continuous deformations.

Given these considerations, direct applications to quantum chemistry for neural nets of the kind tested in this thesis may be limited to large datasets, where average statistical accuracy and low computational overhead are prioritized. This may not be the case if physical priors are effectively incorporated into loss functions during training, but no conclusions can be drawn on this matter from the data presented in this thesis. On the other hand, limitations like this might open a window of opportunity for quantum computation. While neural nets may offer speedups on large datasets, they still struggle to simulate certain aspects of the underlying physics. Conversely, explicitly coded classical programs like PySCF may provide a more faithful physical model, but they suffer in terms of computational resource use. Meanwhile, quantum computers are both explicitly coded — and thus faithful physical models — and also potentially much faster than traditional methods. Thus, we may hypothesize based on the results of this thesis that researchers investigating speedups in quantum simulations on quantum computers should look first into problems such as the calculation of dissociation curves that require simultaneous computational efficiency and fidelity to core physical principles.

4.1 Future Research

This thesis, as has been discussed, is best viewed as a probe into the extremely large problem space of quantum chemistry. It would be entirely intractable for a single person or even a single research group to fully explore all possible avenues of application for machine learning in this field. However, based on the explorations performed over the course of this thesis, it is possible to single out a few next steps in the larger project.

4.1.1 Hyperparameter Optimizer Improvements

For anyone wishing to explore the general application of neural nets to a broad field of problems, refined hyperparameter automation is crucial. The hyperparameter optimizer discussed in this thesis is functional but also somewhat basic. Several features were planned to be included but cut due to time constraints, and several new extensions were identified in the process of writing the code. For someone wishing to hone their Python skills, adding these features may be an interesting and relatively straightforward place to start.

GUI

Solely for organizational purposes, it was hoped that the hyperparameter optimization program might include live-updated graphs, detailing how well each net in a hyperbatch is performing in its training. Additionally, a live-updating graph of the results of each hyperbatch in the hypertrial would be a useful tool for troubleshooting, allowing potential users a much clearer view into the progress (or lack thereof) that their hypertrial is making. At the current stage, the user is notified

4.1 Future Research

only of the loss metric on each net after a cycle of training, and whether or not a net has been pruned.

Mid-Hypertrial Testing

Hand in hand with this, a new in-training test function could be added to future versions. In particular, nets might be tested on a small subset of the test set at intervals during training, and their results recorded. Since no backpropagation has occurred over the testing set, this technique avoids the overfitting concern. Meanwhile, we get a much better metric of the net's actual performance than supplied by loss data from the training set. This creates the opportunity to selectively save the state dictionaries of partially trained nets in a separate file. Specifically, the nets with the best performance in these test set checkpoints may automatically save a "backup" state dictionary, in case further training accidentally negatively effects net performance. This can happen randomly, due to global minimum-seeking routines in the backpropagator, but it can also happen systematically. For example, NN43-0 may have had a higher mean error at a prior stage in its training, but it also may not have yet fallen into the self-zeroing local minimum. Thus, for the specific purposes of the researcher, an under-trained NN43-0 may have actually been of greater utility than the one discussed here. Partially functional versions of this feature do exist in the current code but are still buggy and disorganized. It may be wise for future researchers to start fresh.

4.1 Future Research

Generalized Loss Functions

The third and perhaps most exciting additional feature is increased compatibility for physically-motivated loss functions. One edition of a net with such a loss function was trained without the aide of the hyperparameter optimizer. The idea was to train a neural net to take as input the nuclear composition of a diatomic molecule and output a vector corresponding to a certain section of a Laurent series approximating that molecule's dissociation, in an attempt to circumvent the shortcomings of our initial net when applied dissociated molecules (See Figure 3.5). Random diatoms were generated, notably without xyz position coordinates associated, since the net was to predict energies over a distribution of nuclear separations. During training, the net propagated the inputs as usual, and the Laurent series was modelled and scored via mean squared difference to a random subset of the PySCF-generated dissociation curve. This label curve was acquired by passing the diatom through PySCF with a random assortment of internuclear separation values, giving a rough shape for the overall curve. We see the results of the final trained net in Figure 4.1. Clearly, this approach will, at the very least, take some fine tuning from a future programmer. Additional applications of creative loss functions are detailed in a classical setting in (Greydanus et al., 2019), where Hamiltonians for a handful of physical systems were predicted via neural net to create dynamic simulations. A static adaptation of this approach may well be applicable to the Hartree-Fock procedure.

4.1 Future Research

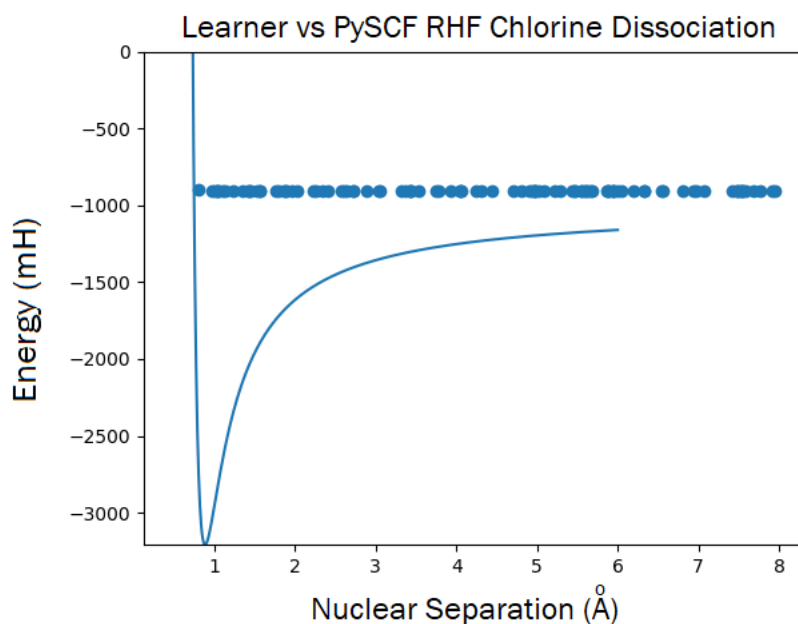


Figure 4.1: Predicted Chlorine Dissociation Curve. The best graph produced by the Laurent series diatomic dissociation learner. PySCF RHF energy values for a random sampling of bond lengths shown as dots, overlaid against the net's predicted function.

4.1 Future Research

4.1.2 New Datasets

Another natural extension of the project would be to train nets to perform tasks not attempted in the course of this thesis. PySCF alone offers a variety of other calculations that could be used to create novel dataset labels, given a group of molecules for inputs. These calculations include RDM prediction and dipole moment estimation, along with information about orbital occupancies and Fock matrix formulation, among others. Any of the above would be a viable target for neural net training, although much more computing power than that used for this thesis would be necessary to handle the parameter spaces needed for some of these tasks. Finally, open-shell and DFT methods could be probed, which could also open the door for more general molecular sampling.

4.1.3 Charting Trends

Lastly, and possibly alongside all of the above, future researchers may benefit from an increased focus on explicit characterization of the regions of the problem space that are particularly difficult for traditional coding. For example, it may be enlightening to probe which molecules tend to converge in PySCF, and after what period of time. This could then be used to isolate a subset of molecular structure problems to focus future machine learning efforts on. If machine learning seems to struggle comparably in the same areas, then quantum computation may be the answer.

4.2 Closing Remarks

The long-term goal pursued in this thesis is, at its core, the cataloging of various computational methods for quantum chemistry. This encompasses an extremely broad horizon of possibilities, only a few of which could be included in this thesis. As such, the door is wide open for all future researchers to build off or even supplant the data supplied here. On a personal note, I have enjoyed this project immensely, both because of how relevant the research felt and because of the freedom of exploration that comes with such a general motivation. Going forward, I would highly recommend the continuation of this project or a similar one to any motivated researcher, undergraduate or otherwise, looking to hone their coding skills. With a project like this, the problem space is never fully explored, always leaving ample room for creative minds to exploit the advantages of machine learning in new and promising ways.

Bibliography

Simulating physics with computers. *International Journal of Theoretical Physics*, 21:467–488.

(2009). *Post-Hartree–Fock Methods*, chapter 8, pages 133–139. John Wiley Sons, Ltd.

Blum, L. C. and Raymond, J.-L. (2009). 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.*, 131:8732.

Dahl, G. E., Sainath, T. N., and Hinton, G. E. (2013). Improving deep neural networks for lvcsr using rectified linear units and dropout. pages 8609–8613.

E., C. and D.L., R. Atomic screening constants from scf functions.

E., C., D.L., R., and W.P., R. Atomic screening constants from scf functions. ii. atoms with 37 to 86 electrons.

Fock, V. (1930). Note on hartree’s method. *Zeitschrift für Physik*, 61:126–148.

Greydanus, S., Dzamba, M., and Yosinski, J. (2019). Hamiltonian neural networks. *CoRR*, abs/1906.01563.

BIBLIOGRAPHY

- Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.
- Hartree, D. R. (1928). The wave mechanics of an atom with a non-coulomb central field. part ii. some results and discussion. *Mathematical Proceedings of the Cambridge Philosophical Society*, 24(1):111–132.
- Loshchilov, I. and Hutter, F. (2017). Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101.
- Miller, J., Friedman, R. H., Hurst, R. P., and Matsen, F. A. (1957). Electronic energy of lih and beh. *The Journal of Chemical Physics*, 27(6):1385–1387.
- Powell, J. R. (2008). The quantum limit to moore’s law. *Proceedings of the IEEE*, 96(8):1247–1248.
- Roothaan, C. C. J. (1951). New developments in molecular orbital theory. *Rev. Mod. Phys.*, 23:69–89.
- Rosenblatt, F. The perceptron – perceiving and recognizing automation. Technical report.
- Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating errors. *Nature*, 323.

BIBLIOGRAPHY

- Rupp, M., Tkatchenko, A., Müller, K.-R., and von Lilienfeld, O. A. (2012a). Fast and accurate modeling of molecular atomization energies with machine learning. *Phys. Rev. Lett.*, 108:058301.
- Rupp, M., Tkatchenko, A., Müller, K.-R., and von Lilienfeld, O. A. (2012b). Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters*, 108:058301.
- Slater, J. C. (1930). Note on hartree's method. *Phys. Rev.*, 35:210–211.
- Sun, Q., Berkelbach, T. C., Blunt, N. S., Booth, G. H., Guo, S., Li, Z., Liu, J., McClain, J. D., Sayfutyarova, E. R., Sharma, S., Wouters, S., and Chan, G. K.-L. (2018). Pyscf: the python-based simulations of chemistry framework. *WIREs Computational Molecular Science*, 8(1):e1340.