Machine Learning
for
Condensed Matter Physics

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

James Nelson

A thesis submitted for the degree of
Doctor of Philosophy
School of Physics
Trinity College Dublin

2021

Plagiarism Declaration

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the Library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

_____

James Nelson

Abstract

This thesis is about the application of machine learning (ML) methods to a variety of problems in condensed matter physics. As condensed matter physics has large computational and experimental datasets readily available - or in the computational case it is often easy to generate them - it is an ideal field for ML. Here ML can be used to speed up existing calculations and routines, or more ambitiously, make new discoveries. Specifically, in this thesis, I look at using ML to: efficiently solve lattice models, predict the Curie temperature of ferromagnets and discover new molecules. Solving lattices models exactly, is generally only possible for small systems, as the dimensionality of the wavefunction increases exponentially with the system size. In this context, we apply ML to: construct exact density functional theory maps, which bypass the calculation of the wavefunction, and learn the propagator, which evolves the system in time. For a ferromagnet to have practical applications it must have a large Curie temperature, however it is very difficult to predict the Curie temperature from first principles. Using a dataset of experimental measurements, we create a Curie temperature predictor, with a mean absolute error of around 50 K. Finally, we look at using ML for the task of discovering new molecules that have certain properties. We demonstrate a ML architecture, capable of learning molecular distributions.

List of Publications

1. "Machine learning density functional theory for the Hubbard model"
   J. Nelson, R. Tiwari, S. Sanvito
   Physical Review B 99 (7), 075132

2. "Predicting the Curie temperature of ferromagnets using machine learning"
   J. Nelson, S. Sanvito
   Physical Review Materials 3 (10), 104405

3. "Machine-learning semi-local density functional formalism for many-body lattice models at zero and finite temperature"
   J. Nelson, R. Tiwari, S. Sanvito
   In production

4. "On machine learning for quantum evolution"
   J. Nelson, L. Coopmans, G. Kells, S. Sanvito
   In production

5. "Generating 3D molecules with generative adversarial networks"
   J. Nelson, A. Lunghi, S. Sanvito
   In production

Conferences & Workshops

- International Workshop on Machine Learning for Materials Science 2018, Helsinki, Finland
  "Predicting the Curie Temperature Using Machine Learning" (Poster)

- Young Researcher's Workshop on Machine Learning for Materials Science 2019, Helsinki, Finland
  "Machine Learning Many-Body Models" (Talk)

- Big Data Summer 2019, Platja d'Aro, Spain
  "Predicting the Curie Temperature Using Machine Learning" (Poster)

- DPG Spring Meeting 2020, Dresden, Germany
  Cancelled Due to Coronavirus (Talk)

- Peter Grünberg Institut for Quantum Theory of Materials, 2020, Julich, Germany
  Cancelled Due to Coronavirus (Talk)

Acknowledgements

Firstly, I would like to thank my supervisor, Prof. Stefano Sanvito, for the guidance he has given me over the past four years. Stefano has always been very generous with his time and has created a great working atmosphere in the Computational Spintronics Group. Throughout my time in the group, I have been amazed at his insight and knowledge over a broad range of topics which reflect the diversity of this thesis.

It has been a pleasure to work alongside many great colleagues in Trinity. In particular, I would like to thank, Dr. Rajarshi Tiwari for all of his technical support and for generating the Hubbard model data used in Chapter 3 and Dr. Alessandro Lunghi for his mentorship and guidance on the work in Chapter 6. Finally, I had the pleasure of sitting beside Dr. Mario Galante and Dr. Emanuele Bosoni, the group was never the same after they left.

I have had the chance to collaborate with two groups outside Trinity during my research. I would like to thank Prof. Stefano Curtarolo for having me over to his group in Duke University, North Carolina for two weeks. Within that group I would like to thank Dr. Cormac Toher for his help in adding structural data to the Curie temperature database. In the Dublin Institute for Advanced Study (DIAS) I would like to thank Luuk Coopmans and Dr. Graham Kells for their guidance on the work in Chapter 4, additionally Luuk helped set up the data generator for the Heisenberg model dynamics.

I would like to thank my family and friends for their support during the PhD. In particular Kathy, who has kept me sane this past year. Finally and most importantly, I would like to thank my parents, without whom, none of this would have been possible.

‘Life in this world,’ he said, ‘is, as it were, a sojourn in a cave. What can we know of reality? For all we see of the true nature of existence is, shall we say, no more than bewildering and amusing shadows cast upon the inner wall of the cave by the unseen blinding light of absolute truth, from which we may or may not deduce some glimmer of veracity, and we as troglodyte seekers of wisdom can only lift our voices to the unseen and say, humbly, “Go on, do Deformed Rabbit . . . it’s my favorite.” ’

- Terry Pratchett, Small Gods

Contents

Introduction

With the successes of machine learning (ML) over the last decade in tasks such as image recognition and language translation it is unsurprising that recently it has started to be used in the physical sciences. In particular, it has found application in condensed matter physics, due to the prevalence of large datasets and the ease of data generation. Here, it has been utilized in tasks ranging from: creating efficient approximations of computationally heavy routines, finding new discoveries and adding deeper insight into the physics behind phenomena.

Reflecting this diversity, this thesis is comprised of four different investigations of ML in condensed matter physics. Due to this varied nature of the PhD, it has been a challenge to structure the thesis coherently and fluidly. The first two chapters detail the necessary background information, the next four contain the results and finally additional material is presented in the appendices. In particular the thesis is structured as follows:

Chapter 1 serves as an introduction to ML. It begins by explaining the fundamentals of ML and supervised learning. After this, various algorithms are described, with neural networks receiving the most attention, as they were the algorithm of choice for the majority of my research. Finally, the chapter ends with a brief exploration of unsupervised learning.

Chapter 2 gives the necessary physics background information. Starting from the physics of electrons, it goes on to describe second quantization and lattice models. The Hubbard model, explored in Chapter 3, and the Heisenberg model, used in Chapter 4, are derived and explored. Finally the chapter ends with a discussion on the scaling of the wavefunction of these models, motivating the work of the next two chapters.

Chapter 3 shows results from using supervised learning to apply density functional theory to the Hubbard model. As the size of the wavefunction increases exponentially with the number of sites in a lattice model, it is only possible to solve them exactly for small systems. However, instead of using the wavefunction to describe the system, one can use the ground-state density,

which scales only linearly with the system size. We show, by using the Hubbard model, that maps between the ground-state density and ground-state properties of the system, can be constructed using ML. Thus, ML provides a method for performing exact lattice density functional theory. Extending this, we use a local approximation of these maps, which allows us to train the ML model on small systems, which are computationally cheap to generate, and predict the properties of larger lattices. Finally, we show that not only can we predict ground-state properties of the systems, but also finite temperature quantities.

Chapter 4 is similar in spirit to the previous chapter, although here the goal is to use ML to propagate quantum systems in time. Using the Heisenberg model, we investigate what ML can achieve in this regard. We find that when the system is represented by the wavefunction or the density matrix, learning the exact propagator is straightforward, given a sufficient amount of data. However, when we use a compressed representation or the reduced density matrix, predicting the future state becomes non-trivial, and the memory of previous states is required. We explore the properties of this non-Markovian dynamics and of the memory, especially the effects related to the time-step size and localization.

Chapter 5 is about using ML to predict the Curie temperature of ferromagnets. For a ferromagnet to have practical applications it must have a large Curie temperature. However, it is very difficult to predict the Curie temperature from first principles. Taking a data-driven approach, we construct a Curie temperature predictor using experimental data. Although our model only uses the chemical composition of the material to make a prediction, its mean absolute error is only 50 K. In addition to this, we can also reliably estimate the uncertainty of our predictions. Further, we investigate whether incorporating structural information can improve the performance.

Finally, Chapter 6 deals with using ML for the ambitious task of creating models that can discover new molecules that have certain properties. Treating this as a generative modelling task, given a dataset of molecules we design a model, using generative adversarial networks, that can create similar molecules to the ones in the dataset. After introducing the model, we show how we can accurately reproduce thermal distributions of single molecules. Finally, we move on to the more challenging problem of modelling datasets with multiple molecules.

Chapter 1

Machine Learning

> "Machines take me by surprise with great frequency."
> - Alan Turing [1]

1.1 Introduction

Machine learning (ML) is the science of using computer algorithms to learn from previous data. Here learning can be defined as:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [2].

In some sense this is just curve fitting, given some data we find a function that can predict all the data-points accurately. However, by combining large amounts of data with powerful models impressive results can be achieved that go beyond mere curve fitting and arguably demonstrate intelligence. This can be seen in some of the most impressive showcases of ML, which include:

- Mastering strategy games such as chess, shogi, and go [3]
- Writing paragraphs of text given some prompt [4]
- Human-level performance in medical imaging [5]
- Recognising human speech [6]

In this Chapter, I lay out the fundamentals of machine learning beginning with the theory behind supervised learning. I then proceed to discuss specific

algorithms with an emphases on neural networks as they were used the most in my research. I then go on to discuss how we may estimate the uncertainty of the models and finish with a brief introduction to unsupervised learning. For more detail on: general machine learning see Ref. [2, 7], neural networks see Ref. [8] and artificial intelligence see Ref. [9].

1.2 Supervised Learning

In supervised ML, the aim is to estimate a function f that will map an input $\mathbf{x}$ to an output $\mathbf{y}$

$$f : \mathbf{x} \to \mathbf{y} \tag{1.1}$$

given a set of data. The input, $\mathbf{x}$, is called the feature vector [1] and it contains information that is relevant to the prediction of $\mathbf{y}$, which is called the target vector. Note that $\mathbf{x}$ could also be a matrix - images for example - or a tensor, but here I will assume without loss of generality that it is a vector[2]. Supervised learning can be subdivided into two classes: regression and classification.

In regression, $\mathbf{y}$ is a real valued scalar or vector. An example of regression would be predicting house prices. Here $\mathbf{x}$ would encode the attributes of the house, i.e. age of house, number of bedrooms, etc. and y would be the price of the house.

While in classification y represents a class. If we only have two classes - binary classification - then y can be a scalar with 0 representing one class and 1 the other. If we have more than two classes then y is generally a vector where the i^{th} class is denoted with a 1 at position i and zeros elsewhere, this is called one-hot encoding. As an example of classification we might have a set of images each containing either a dog, cat or bird. Then we could represent the classes in the one-hot format with Dog=(1,0,0), Cat=(0,1,0) and Bird=(0,0,1) and $\mathbf{x}$ would be the pixels of the image.

When a model has been trained on a set of data, just because it achieves a low error does not mean it can predict new data accurately. Figure 1.1 shows an example of this, here the model corresponding to the green line can predict the training data well but it is inaccurate for the data it has not seen before. This is known as overfitting, here the model just learns correlations in the training set and not the true function. The opposite to overfitting is underfitting where the model is not complex enough to represent the true function and thus has a high training error, in Figure 1.1 this is the orange line.

[1] In physics it is often called the descriptor and in deep learning the representation.

[2] A matrix or tensor can always be reshaped into a vector.

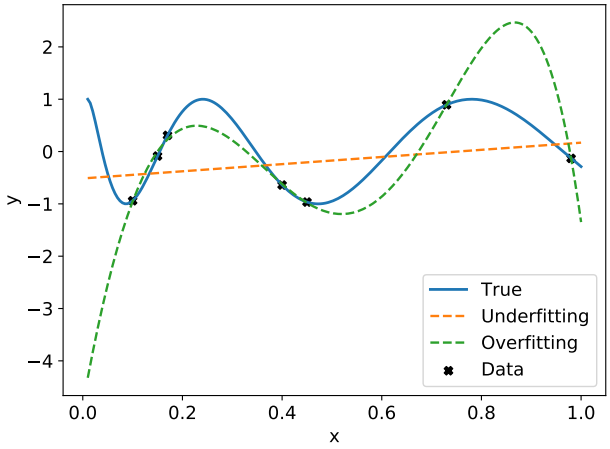


Figure 1.1: Here I show an example of overfitting and underfitting. The true function is in blue with the training set in black xs. The orange line is underfitting and is not sufficiently complex to represent the true function. While the green line is overfitting and differs widely from the true function outside the training set.

To understand overfitting and underfitting in more detail suppose we have a scalar ML model $f(\mathbf{x};\theta)$ where θ are parameters learnt from some given set of data. If we randomly select the training data our expected squared error at a given point $\mathbf{x}$ with true value y is

$$\begin{aligned} E_\theta[(y - f(\mathbf{x};\theta))^2] &= (y - E_\theta[f(\mathbf{x};\theta)])^2 + E_\theta[(E_\theta[f(\mathbf{x};\theta)] - f(\mathbf{x};\theta))^2] \\ &= \text{bias}[f(\mathbf{x};\theta)]^2 + \text{var}[f(\mathbf{x};\theta)]. \end{aligned} \tag{1.2}$$

The first term on the right-hand side is the squared bias and represents the error of the mean prediction while the second term is the variance and measures the disagreement between the models. To see how this relates to overfitting and underfitting consider first the underfitting case. Here since the model will have low capacity it will not change much each time we train on a random set, thus it will have low variance. But if the function we are trying to learn is complex then we will have high bias as the model is not complex enough to learn it. For overfitting the situation is reversed with low bias and high variance.

To avoid overfitting and to obtain a non-biased estimate of the performance of the model the dataset is randomly split into three mutually exclusive subsets: a training set, a validation set and a test set [7]. The training set is used to determine the parameters of the model, generally denoted by θ, by finding the values of θ that minimize the error as measured by some performance function.

To prevent overfitting, hyperparameters, denoted by λ, are introduced. These are model parameters, which are determined by the validation set. For a given choice of λ we find the best set of θ values by minimizing the performance metric over the training set. Then we use the validation set to choose the best set of hyperparameters and the best model - we may have several different choices of model - by seeing, which choice has the lowest performance metric over the validation set. Finally the test set is used for estimating the generalization error of the model, this is the expected error of our model over the underlying probability distribution of the dataset, p_{data}. Here p_{data} returns the probability of a feature vector and corresponding target for the given dataset, $p_{\text{data}} = p_{\text{data}}(\mathbf{x}, y)$. The reason we use the test set to estimate the generalization error is because the training and validation errors are biased due to the fact that they have been used to fit parameters.

There is no exact way to divide the dataset into training, validation and test sets. A larger training set will mean better model parameters, a larger validation set will result in a better estimation for the hyperparameters and finally the accuracy of the generalization error depends on how large the test set is. A reasonable choice might be 50%-25%-25% but there is no theoretical justification for this.

It is important to always bear in mind that the three subsets should be drawn from the same probability distribution and that the test set error is an estimate of the expected error over this distribution. For example suppose we learn some one dimensional function in the range $[0, 1]$. If we achieve a low test error, this only tells us about samples in that range, we have no guarantee that the model will be accurate in the range $[3, 5]$ for example.

Often we are in the position where we have limited data and dividing it into three datasets would make each dataset too small. One strategy used in this situation is to combine the training and validation sets into a single set (which we will refer to as the training set) and use K-Fold Cross Validation to choose the best model. Here the training set is split into K subsets, for each set the model is trained on the other (K-1) sets and tested on that set, the cross-validation error is then the average of all the errors. Then the best model (the model with the lowest cross-validation error) is trained on the entire training set. Finally the test set is used to estimate this model's accuracy.

1.2.1 Ridge Regression

To make these concepts less abstract I will now introduce a concrete example, ridge regression.

For a feature vector with p components, $\mathbf{x} = (x_1, x_2, ..., x_p)$, a linear regression model [3] predicts

$$f(\mathbf{x}) = c + \sum_{j=1}^{p} b_j x_j = c + \mathbf{b} \cdot \mathbf{x} \tag{1.3}$$

where we have the bias, c, and the weights, $\mathbf{b}$, which together comprise the model parameters $\theta = (c, \mathbf{b})$. It is convenient to use the mean squared error as our loss function

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - c - \mathbf{b} \cdot \mathbf{x}^{(i)})^2 \tag{1.4}$$

as this can be solved analytically by minimizing $J(\theta)$ with respect to θ. Here N is the number of data-points in the training set. The optimum value of c is given by

$$\frac{\partial}{\partial c} J(\theta) = 0 \quad \rightarrow \quad c = \frac{1}{N} \sum_{i=1}^{N} y^{(i)}. \tag{1.5}$$

To solve for $\mathbf{b}$ we rewrite $J(\theta)$ in a more compact form by defining the $N \times p$ matrix X, whose i^{th} row is $\mathbf{x}^{(i)}$, $X = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)})^T$ and letting $\bar{\mathbf{y}} = \mathbf{y} - c$. Then we have $J(\theta) = \frac{1}{N}(\bar{\mathbf{y}} - X\mathbf{b})^T(\bar{\mathbf{y}} - X\mathbf{b})$, with optimum value of $\mathbf{b}$

$$\nabla_{\mathbf{b}} J(\theta) = \mathbf{0} \quad \rightarrow \quad \mathbf{b} = (X^T X)^{-1} X \bar{\mathbf{y}}. \tag{1.6}$$

Here the $p \times p$ matrix $X^T X$ measures how correlated the features are [4], with $(X^T X)_{ij} = \sum_{k=1}^{N} x_i^{(k)} x_j^{(k)}$.

One method of preventing overfitting is to add a regularization term to the cost function [5]

$$\begin{aligned} J(\theta, \lambda) &= \frac{1}{N} \sum_{i=1}^{N} (y^{(i)} - c - \mathbf{b} \cdot \mathbf{x}^{(i)})^2 + \lambda \sum_{j=1}^{p} b_j^2 \\ &= \frac{1}{N} (\bar{\mathbf{y}} - X\mathbf{b})^T (\bar{\mathbf{y}} - X\mathbf{b}) + \lambda \mathbf{b}^T \mathbf{b}. \end{aligned} \tag{1.7}$$

When this regularization term is added to the cost function the model is called ridge regression. To see why this prevents overfitting imagine that λ was a very large value, then the value of $\mathbf{b}$ that minimized $J(\theta, \lambda)$ would be effectively

[3] Note that linear Regression is not limited to purely linear functions, for example we can learn polynomials of any degree q

$$f(x) = c + \sum_{n=1}^{q} x^n b_n.$$

[4] It is closely related to the covariance matrix $C_{ij} = \frac{1}{N} \sum_{k=1}^{N} (x_i^{(k)} - m_i)(x_j^{(k)} - m_j)$ where $m_i = \frac{1}{N} \sum_{k=1}^{N} x_i^{(k)}$.

[5] Note that we do not penalize the bias, c.

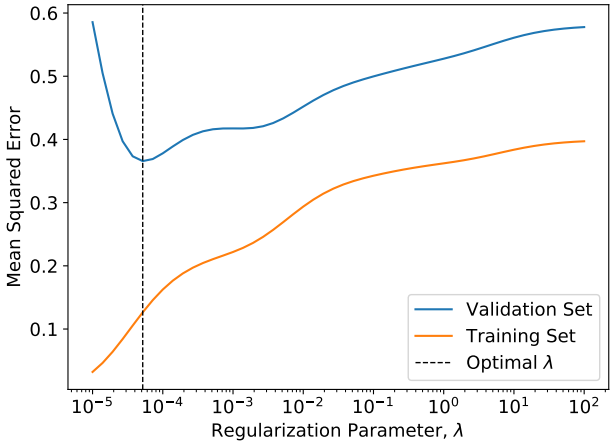


Figure 1.2: Here I show how the regularization parameter, λ, in ridge regression can decrease the validation error. When λ is small we have overfitting as characterized by a low training error and high validation error. While when λ is large we underfit. The optimal value of λ is between the two regimes as shown by the dashed line.

all zeros and hence it would under-fit the data. Figure 1.2 shows how the regularization parameter effects the validation and training error. On the left hand side in the Figure we have overfitting, with low training error and high validation error. While on the right hand side we have underfitting, with both errors large. Minimizing Equation (1.7) we find the same value as before for c and

$$\mathbf{b} = (X^T X + \lambda \mathbb{1})^{-1} X \bar{\mathbf{y}}. \tag{1.8}$$

There is no principal reason to use a squared regularizer, one could also use the absolute value, $\sum_{j=1}^{p} |b_j|$. Now when we take a derivative with respect to b_i we get $\text{sgn}(b_i) = b_i/|b_i|$. This has the effect of penalizing non-zero values and hence leads to many of the weights being zero. This is known as the least absolute shrinkage and selection operator (LASSO) and unlike ridge cannot be solved in closed form and instead must be solved by some other means, see section 1.4.2

1.2.2 Loss Functions

In general we would like a systematic way of finding the best parameters for the data. If we let θ be our model parameters and $d^{(i)} = (y^{(i)}, \mathbf{x}^{(i)})$ be the data-points, then one strategy is to find the parameters that the model thinks is the

most likely given the data. Using Bayes' Theorem[6] we can write the probability of a set of parameters given some data as

$$P(\theta|d^{(1)}, d^{(2)}, ..., d^{(N)}) = \frac{P(d^{(1)}, d^{(2)}, ..., d^{(N)}|\theta)P(\theta)}{P(d^{(1)}, d^{(2)}, ..., d^{(N)})} \tag{1.9}$$

where $P(A)$ is the probability of A and $P(A|B)$ is the probability of A given B. Since all the samples in the dataset are collected independent of one another we can write

$$P(d^{(1)}, d^{(2)}, ..., d^{(N)}|\theta) = \prod_{i=1}^{N} P(d^{(i)}|\theta) \tag{1.10}$$

and similarly $P(d^{(1)}, d^{(2)}, ..., d^{(N)}) = \prod_{i=1}^{N} P(d^{(i)})$. The best set of parameters, θ^*, that maximise Equation (1.9), is the so-called maximum likelihood estimate. For computational reasons it is easier to maximise the logarithm of this [7], we also divide by the number of samples for convenience. Putting everything together we have the mean log-likelihood

$$\begin{aligned} L &\equiv \frac{1}{N} \log P(\theta|d^{(1)}, d^{(2)}, ..., d^{(N)}) \\ &= \frac{1}{N} \sum_{i=1}^{N} \log P(d^{(i)}|\theta) + \frac{1}{N} \log P(\theta) - \frac{1}{N} \sum_{i=1}^{N} \log P(d^{(i)}). \end{aligned} \tag{1.11}$$

The first term here is called the posterior, how likely is the data given θ. The next term is the prior, how likely do we think θ is before we have seen the data. Finally we have a term which describes how likely the data is, however since it does not depend on θ we can ignore it. By convention instead of maximizing the likelihood, we seek to minimise the negative likelihood which leads to the following cost function

$$J = -\frac{1}{N} \sum_{i=1}^{N} \log P(d^{(i)}|\theta) - \frac{1}{N} \log P(\theta). \tag{1.12}$$

For the moment let us assume we do not have any idea what values of θ are likely and assign an infinitely wide prior $P(\theta) = U[-\infty, \infty]$ which produces a constant term in Equation (1.12) that we can simply ignore. This reduces the cost function to

$$J = -\frac{1}{N} \sum_{i=1}^{N} \log P(d^{(i)}|\theta) \equiv -\mathrm{E}_{d \sim p_{\mathrm{data}}} \log P(d|\theta) \tag{1.13}$$

[6] Bayes' Theorem can be derived by simply noting $P(A\&B) = P(A|B)P(B) = P(B|A)P(A)$.

[7] This will give the same maximum since the logarithm is a monotonically increasing function.

where $\mathrm{E}_{d\sim p_{\mathrm{data}}}$ is the expectation value over the dataset. Minimizing this function is equivalent to minimizing the Kullback-Leibler divergence between the real and predicted distributions

$$\begin{aligned} D_{\mathrm{KL}}[p_{\mathrm{data}}|P] &= \mathrm{E}_{d\sim p_{\mathrm{data}}} \log\left(\frac{p_{\mathrm{data}}(d)}{P(d|\theta)}\right) \\ &= -H[p_{\mathrm{data}}] - \mathrm{E}_{d\sim p_{\mathrm{data}}} \log P(d|\theta), \end{aligned} \tag{1.14}$$

where $H[p_{\mathrm{data}}]$ is the entropy of the data distribution defined by

$$H[p_{\mathrm{data}}] = -\mathrm{E}_{d\sim p_{\mathrm{data}}} \log p_{\mathrm{data}}(d). \tag{1.15}$$

As the Kullback-Leibler divergence is always non-negative and only zero when the two distributions are equal, the global minimum of the maximum likelihood estimation occurs at $P(d|\theta^*) = p_{\mathrm{data}}(d)$. Hence the maximum likelihood estimate reproduces the data distribution.

The relationship between $P(d|\theta)$ and the output of our model depends on the task. For binary classification as we have two labels 0 and 1 we can interpret our model $f(\mathbf{x})$ as outputting the probability that $\mathbf{x}$ is in class 1, $f(\mathbf{x}) = P(1|\mathbf{x})$ with $P(0|\mathbf{x}) = 1 - f(\mathbf{x})$. Thus for binary classification we can write

$$P(y|\mathbf{x}) = f(\mathbf{x})^y (1 - f(\mathbf{x}))^{1-y}, \tag{1.16}$$

and plugging this into Equation (1.13) we get

$$J = -\frac{1}{N}\sum_{i=1}^{N} \left(y^{(i)} \log f(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - f(\mathbf{x}^{(i)})) \right), \tag{1.17}$$

which is called the binary cross entropy. For multi-classification problems with k classes and one-hot encoding we can write

$$P(y|\mathbf{x}) = \prod_{j=1}^{k} f(\mathbf{x})_j^{y_j}, \tag{1.18}$$

where here $f(\mathbf{x})$ is a vector in which the j^{th} component is the probability of $\mathbf{x}$ belonging to class j. Thus the cost function is

$$J = -\frac{1}{N}\sum_{i=1}^{N} \left(\sum_{j=1}^{k} y_j^{(i)} \log f(\mathbf{x}^{(i)})_j \right) \tag{1.19}$$

which is known as the cross-entropy.

For regression we need to specify a probability distribution for the posterior, a common choice is a Gaussian distribution

$$P(y|\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(y - f(\mathbf{x}))^2/2\sigma^2}. \tag{1.20}$$

Inserting this into Equation (1.13) we get $J = -\frac{\log(\pi\sigma^2)}{2} + \frac{1}{2\sigma^2 N}\sum_{i=1}^{N}(y^{(i)} - f(\mathbf{x}^{(i)}))^2$, which leads to the mean squared error

$$J_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - f(\mathbf{x}^{(i)}))^2. \tag{1.21}$$

Another choice is the exponential distribution

$$P(y|\mathbf{x}) = \frac{\gamma}{2}e^{-\gamma|y-f(\mathbf{x})|}. \tag{1.22}$$

This gives $J = -\log(\gamma/2) + \frac{\gamma}{N}\sum_{i=1}^{N}|y^{(i)} - f(\mathbf{x}^{(i)})|$, which leads to the mean absolute error [8]

$$J_{\text{MAE}} = \frac{1}{N}\sum_{i=1}^{N}|y^{(i)} - f(\mathbf{x}^{(i)})|. \tag{1.23}$$

In order to illustrate the difference between the two, suppose we have a dataset where all the feature vectors are the same but the target values are different [9]. Since the feature vectors are all the same, the predicted value $\hat{y}$ will be the same. The mean squared error is minimized with the mean

$$\hat{y} = \frac{1}{N}\sum_{i=1}^{N}y^{(i)}, \tag{1.24}$$

while the mean absolute error is minimized with the $\hat{y}$ that satisfies

$$\frac{1}{N}\sum_{i=1}^{N}\text{sgn}(\hat{y} - y^{(i)}) = 0, \tag{1.25}$$

where $\text{sgn}(x) = x/|x|$ is the sign function, corresponding to the median.

Finally what about including a prior distribution? Suppose we have a linear model with p weights and think that all the weights should be small, then we might use a Gaussian prior centred around zero for each weight

$$P(\theta) = \prod_{i=1}^{p}\frac{1}{\sqrt{2\pi\lambda^2}}e^{-b_i^2/2\lambda^2}. \tag{1.26}$$

Taking the negative mean log likelihood we get the following term in our cost function

$$J_{\text{reg}} = \text{const} + \sum_{i=1}^{p}b_i^2/2\lambda^2 \tag{1.27}$$

which corresponds to weight regularization, like in ridge regression.

[8] The two are related by

$$\sqrt{\frac{J_{\text{MSE}}}{N}} \leq J_{\text{MAE}} \leq \sqrt{J_{\text{MSE}}}.$$

[9] This could come about because the feature vector does not contain enough information to predict the target or if the problem has inherent randomness.

1.3 Shallow Learning Models

Machine learning algorithms can be split into two groups, shallow models and deep models. A deep model is a neural network with at least two layers and shallow models are everything else. Ridge regression presented above is a shallow model, I now introduce two more. While all of the algorithms can also be used for classification, here I only discuss regression.

1.3.1 Random Forest Regression

Random forest (RF) regression uses a collection of regression trees to make predictions. A regression tree is a model that splits the feature space into M partitions $R_1, R_2, .., R_M$ and then for an input $\mathbf{x}$ returns

$$T(\mathbf{x}) = \sum_{m=1}^{M} c_m I(\mathbf{x} \in R_m), \tag{1.28}$$

where $I(x)$ is the indicator function and c_m is some constant [7]. Using a squared error loss, $J = \sum_{i=1}^{N}(y^{(i)} - T(\mathbf{x}^{(i)}))^2$, and minimizing with respect to c_m we get

$$c_m = \frac{1}{|R_m|} \sum_{\mathbf{x}^{(i)} \in R_m} y^{(i)} \tag{1.29}$$

where $|R_m|$ is the number of examples in R_m, i.e. c_m is the average target of region R_m. How are these partitions discovered? The first one is found by the optimal (j, s) pair that minimizes

$$J = \sum_{\mathbf{x}^{(i)} \in R_1(j,s)} (y^{(i)} - c_1)^2 + \sum_{\mathbf{x}^{(i)} \in R_2(j,s)} (y^{(i)} - c_2)^2 \tag{1.30}$$

where $R_1(j, s) = \{\mathbf{x}|x_j \leq s\}$, $R_2(j, s) = \{\mathbf{x}|x_j > s\}$ and c_i is again the average of the regions. Thus, given a p-dimensional feature vector we go through all p features and find the best split to make, if we have N data-points that means we must check pN different potential splits. More and more splits are created until some maximum tree depth has been reached.

An issue with regression trees is that they are prone to over-fitting the data. Random forests tries to rectify this by averaging over an ensemble of different trees - hence a forest. The trees differ in two ways, firstly when finding the best split, a random set of features is chosen. Secondly we use a method called bootstrapping to create a different training set for each tree. In bootstrapping we sample with replacement from the training set to create a new set the same size. This new set will contain duplicates and some samples in the original training set will not be in the bootstrapped set. The probability of a given

sample being in the bootstrapped set is roughly 63% [10]. Thus if we have B trees to make a new prediction we take an average

$$f(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} T_b(\mathbf{x}). \tag{1.31}$$

Averaging over a set of bootstrapped samples is called bagging - bootstrapped aggregation.

Suppose each tree has an expected squared error, $\mathrm{E}[(y - T_b(\mathbf{x}))^2] = \sigma^2$ and together they have a pairwise correlation of $\mathrm{E}[((y-T_b(\mathbf{x}))(y-T_c(\mathbf{x}))] = \rho$. Then the expected squared error for a prediction for the ensemble is

$$\mathrm{E}[((y - f(\mathbf{x}))^2] = \frac{\sigma^2 - \rho}{B} + \rho. \tag{1.32}$$

If the trees are all the same, then $\rho = \sigma^2$ and the ensemble error is obviously the same as the individual error, σ^2. If trees are completely uncorrelated, $\rho = 0$, then the error is σ^2/B which tends to zero as B becomes large. Thus by having the trees even slightly decorrelated the expected error of the forest will be lower than the expected error of the trees.

An advantage of using random forests is that they are interpretable. Each tree is essentially a look up table, and we can easily tell what they are doing. A disadvantage is that the pN scaling becomes intractable as we move to the big data limit.

1.3.2 Kernel Methods

Kernel methods are based around learning a similarity metric on the dataset, and then returning a weighted sum based on this metric for a prediction. Concretely, given a feature vector, $\mathbf{x}$, they predict

$$f(\mathbf{x}) = \sum_{m=1}^{M} b_m h_m(\mathbf{x}) + c, \tag{1.33}$$

with weights b_m, bias c and a set of M basis functions, $h_m(\mathbf{x})$. A popular type of kernel method for regression is kernel ridge regression (KRR), here we use a squared error loss with a regularization penalty

$$J = \sum_{i=1}^{N}\left(y^{(i)} - c - \sum_{m=1}^{M} b_m h_m(\mathbf{x}^{(i)})\right)^2 + \lambda\sum_{m=1}^{M} b_m^2. \tag{1.34}$$

[10] If the training set has N samples in it, then the probability for a given sample to not be in the bootstrapped set is $(1 - 1/N)^N$, which in the large N limit is $1/e$. Hence we get the probability of $1 - 1/e \approx 0.63$ of a given sample being in the bootstrapped set.

Similar to ridge $c = \frac{1}{N}\sum_{i=1}^{N} y^{(i)}$. Defining $\bar{y}^{(i)} = y^{(i)} - c$ and the matrix $H_{ij} = h_j(\mathbf{x}^{(i)})$ this becomes

$$J = (\bar{\mathbf{y}} - H\mathbf{b})^T(\bar{\mathbf{y}} - H\mathbf{b}) + \lambda \mathbf{b}^T\mathbf{b} \tag{1.35}$$

with

$$\nabla_{\mathbf{b}} J = 0 \quad \rightarrow \quad \mathbf{b} = (H^T H + \lambda \mathbb{1})^{-1} H^T \bar{\mathbf{y}}. \tag{1.36}$$

So far this is identical to ridge regression except for the basis functions, but the next step differentiates the two methods. We start by using the identity $(H^T H + \lambda\mathbb{1})^{-1}H^T = H^T(HH^T + \lambda\mathbb{1})^{-1}$, which allows us to write

$$\mathbf{b} = H^T(K + \lambda\mathbb{1})^{-1}\bar{\mathbf{y}}, \tag{1.37}$$

where we have defined the kernel matrix $K = HH^T$. The kernel matrix is an $N \times N$ positive definite matrix with $K_{ij} = (HH^T)_{ij} = \sum_{m=1}^{M} h_m(\mathbf{x}^{(i)})h_m(\mathbf{x}^{(j)})$, i.e. the elements are the inner products of the basis functions computed over the training set. The idea behind kernel methods is that we do not actually have to evaluate the basis functions, instead we evaluate their inner products, which we are free to choose. A popular choice is $K_{ij} = D(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ where

$$D(\mathbf{x}, \mathbf{y}) = \exp(-\gamma|\mathbf{x} - \mathbf{y}|^2). \tag{1.38}$$

As such in order to make a new prediction for the feature vector $\mathbf{x}$ we compute

$$f(\mathbf{x}) = c + \sum_{i=1}^{N} a_i D(\mathbf{x}, \mathbf{x}^{(i)}), \tag{1.39}$$

where $a = (K + \lambda\mathbb{1})^{-1}\bar{\mathbf{y}}$. Intuitively this makes sense, $D(\mathbf{x}, \mathbf{y})$ measures how close samples are and a contribution to the target is scaled accordingly. $D(\mathbf{x}, \mathbf{y})$ must satisfy two constraint namely symmetry $D(\mathbf{x}, \mathbf{y}) = D(\mathbf{y}, \mathbf{x})$ and positivity $\mathbf{v}^T HH^T\mathbf{v} = (H^T\mathbf{v})^T(H^T\mathbf{v}) \geq 0 \rightarrow \sum_{ij} v_i D(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})v_j \geq 0$.

An advantage of KRR is that it is exactly solvable. In contrast the fact that the kernel matrix needs N^2 evaluations to be computed and that to make a prediction it needs N evaluations, means that for large datasets it is inefficient.

1.4 Neural Networks and Deep Learning

Neural networks (NNs) are a type of model inspired by the brain. The philosophy that inspired them is called connectionism which posits

> "units which are not intelligent themselves, may exhibit intelligent behaviour operating as a whole" [10].

Neural networks have become the state of the art of ML - especially in the big data regime - for several reasons, which are expanded on in the coming sections. Firstly, neural networks are universal approximators in the sense that they can learn any function up to an arbitrary error. Next, the time to train and make a prediction scales linearly with the number of data-points making large scale training feasible. Finally, neural networks are very flexible and can have prior knowledge relevant to the problem encoded in them. While all of these properties are not unique to neural networks, it is the combination of them that makes them so powerful.

1.4.1 Neural Networks

The simplest type of NN is a fully connected neural network an example of which is shown in Figure 1.3. They are comprised of three different types of layers: input, hidden and output, with each layer containing a certain number of units [11]. By convention we label the first layer, the input layer, as 0, then the first hidden layer is 1 and so on. The number of units in each layer is $p^{(i)}$, with $p^{(0)} = p$ the dimension of the feature vector. The units are connected together via weight matrices, the weight matrix connecting layer i and $i+1$ is labelled $W^{(i)}$ with dimension $p^{(i+1)} \times p^{(i)}$. As well as weight matrices there are also bias vectors $\mathbf{b}^{(i)}$ of dimension $p^{(i+1)}$. The number of parameters in the network is given by

$$N_{\text{par}} = \sum_{i=0}^{L} (p^{(i)} + 1) p^{(i+1)}, \tag{1.40}$$

where L is the number of hidden layers and the plus 1 contribution comes from the bias vectors.

The output of the L hidden layers is given by

$$\mathbf{h}^{(i)} = g(\mathbf{b}^{(i-1)} + W^{(i-1)}\mathbf{h}^{(i-1)}) \tag{1.41}$$

where $\mathbf{h}^{(0)} = \mathbf{x}$ and g is a non-linear function called the activation function. After propagating through the layers the output is given by

$$\hat{y} = q(\mathbf{b}^{(L)} + W^{(L)}\mathbf{h}^{(L)}) \tag{1.42}$$

where the choice of function q depends on the task.

For regression problems we do not want to limit the range of the output and hence we use a linear output $q(z) = z$. For binary classification we want an output between 0 and 1, the sigmoid function is the standard choice

$$q(z) = \frac{1}{1 + e^{-z}}. \tag{1.43}$$

[11] Units, nodes and neurons are used interchangeably.

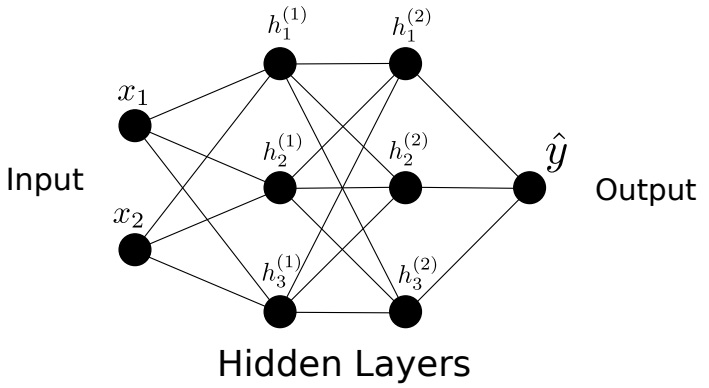


Figure 1.3: A neural network with a two hidden layers and one output unit. The connections between the nodes correspond to the weight matrix parameters. The total number of trainable parameters in the network is $(2+1)3+(3+1)3+(3+1)1 = 24$.

When this is used with the cross entropy of Equation (1.17) the log and exponential effectively cancel out and the cost function never saturates, which is an important property for gradient based optimization [12]. Finally for multi-classification typically the softmax function is used

$$q(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}. \tag{1.44}$$

Here we have an output for each class and using the softmax function guaranties the outputs will be non-negative and sum to one. This is just a generalization of the sigmoid, as can be seen when considering two classes $q(z)_1 = e^{z_1}/(e^{z_1} + e^{z_2}) = 1/(1 + e^{z_2 - z_1})$.

Returning to the activation function, the reason we use a non-linear function is that if the activation function was linear then the whole resulting network would only be capable of only learning linear functions [13]. There is an important theorem [11], which tells us that if g is non-linear and our neural network has a sufficient number of units, then it can approximate any reasonable function to arbitrary error. Thus we say that neural networks are universal approximators. Note that this does not guarantee that the neural net will always perform well, it could over-fit the training set or fail to converge during training.

Common choices of activation function are shown in Table 1.1. In general the ReLU activation function seems to be the best choice as it is fast to compute

[12] The mean squared error loss together with the sigmoid output does saturate leading to poor performance.

[13] A sequence of linear transformations is itself a linear transformation.

Name	Formula
Hyperbolic Tangent	$\tanh(z)$
Sigmoid	$1/(1+e^{-z})$
Rectified Linear Unit (ReLU)	$\max(0, z)$
Leaky ReLU	$\max(0, z) + \alpha \min(0, z)$
ELU	$\max(0, z) + \min(0, \alpha(e^z - 1))$

Table 1.1: A list of common activation functions used in neural networks, with their corresponding formula [8]. Note that the first two functions are bounded, while the final three are unbounded.

and is unbounded, however if possible, it is best practice to choose the activation function using the validation set.

One way of thinking about a neural network is that the hidden layers learn a representation for the decision layer. For example, suppose we have a regression problem, where we are mapping $\mathbf{x}$ to y. One approach could be to hand-design features based on the input, $\theta_i(x)$, and then predict y using a linear model $y = c + \sum_i w_i \theta_i(\mathbf{x})$. If we apply neural nets to the problem, then we could think of the hidden layers as trying to learn good features $\theta_i(\mathbf{x})$ with the final layer performing linear regression.

Choosing the architecture (number of layers and number of nodes per layer) of the neural network is more art than science in practice. However, there is evidence that using a deeper network leads to a lower test error [8]. The reasoning behind this is that the deeper network learns higher order features than a shallow net, with these higher order features less prone to over-fitting and hence leading to better generalization error.

1.4.2 Optimization

There is no exact solution to find the optimal set of parameters for a neural network so instead an optimizer is necessary. Furthermore the loss function for neural networks must have many local minima as we can create equivalent nets by permuting the weights. Thus after optimization it is extremely unlikely we will be in the global minimum. However, this is not a problem if the difference in value between the loss function at the local minima and global minimum is small, as in this case even if the weights are in a local minima the network will still perform well, and in practice this seems to be the case. For training neural networks the state of the art is gradient based optimization.

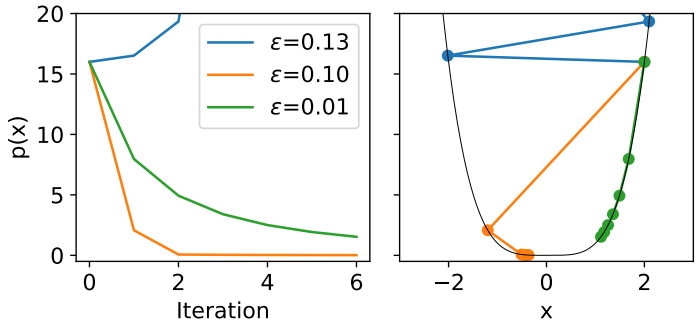


Figure 1.4: Here we use gradient descent to minimize the function $p(x) = x^4$ starting from the initial solution of $x = 2$. The left panel shows the value of the function over the iterations and in the right panel we show the course of the descent. As is shown a learning rate that is too large can quickly explode in value (blue line) while a learning rate too small is inefficient (green line).

Suppose we have some function, $p(\mathbf{x})$, that we want to minimize and we let $\mathbf{g} = \nabla p(\mathbf{x})$ be the gradient at position $\mathbf{x}$. Taking a step in the opposite direction to the gradient, $\mathbf{x}' = \mathbf{x} - \epsilon\mathbf{g}$, results in

$$p(\mathbf{x}') = p(\mathbf{x} - \epsilon\mathbf{g}) = p(\mathbf{x}) - \epsilon|\mathbf{g}|^2 + \frac{\epsilon^2}{2}\mathbf{g}^T H\mathbf{g} + ... \tag{1.45}$$

where H is the Hessian matrix and ϵ is the learning rate. We see for a sufficiently small ϵ, $p(\mathbf{x}') \leq p(\mathbf{x})$. This forms the basis of the gradient descent algorithm. We start from an initial guess $\mathbf{x}_0$ and update by

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \epsilon\nabla p(\mathbf{x}_n) \tag{1.46}$$

until convergence is reached. An example using gradient descent to minimize the function $p(x) = x^4$ is shown in Figure 1.4. The number of iterations needed to minimize the function can be reduced by fine tuning the value of the learning rate.

In order to train neural networks using gradient descent we first compute the cost function on the training set, then update the weights of the model using a single gradient descent step. This is repeated until we reach some convergence criterion. Thus updating the network involves having to compute the derivative of the cost function with respect to every parameter in the network. An effective algorithm to do this is called back-propagation [8], which uses the derivative chain rule to lessen the computation and memory requirements.

A slight modification to the gradient descent results in one of the most popular algorithms for optimizing neural networks - stochastic gradient descent.

Recall that the cost function is simply an expectation of some function over the training set [Equation (1.13)]. If our training set is very large then to make a single update to the parameters is expensive. Instead it is quicker to divide our dataset into batches and then use each batch to make one update. If we have N samples in our training set and use a batch size of b, then we will have ceil(N/b) batches [14]. For example, if we have $N_{\text{train}} = 200$ and $b = 16$, then we will have 13 batches, with 8 samples in the final batch. Or we could choose a random set of 16 data-points each time. We call an epoch one cycle through all the batches, where ceil(N/b) updates have been made. The larger the batch size the more accurate the estimate of the cost function with diminishing returns as the batch size gets larger. By choosing a batch size the same size as the training set, we get back to the original gradient descent algorithm. Thus we have the update rule

$$\begin{aligned} \mathbf{g}_n &= \nabla_\theta \left(\frac{1}{b} \sum_{i=1}^{b} L(f(\mathbf{x}^{(i)}), y^{(i)}) \right), \\ \theta_{n+1} &= \theta_n - \epsilon \mathbf{g}_n. \end{aligned} \tag{1.47}$$

The learning rate, ϵ, is a free parameter that must be determined from trial and error. The rest of the optimizers in this section are modifications of stochastic gradient descent.

A simple extension is to include so-called momentum. Here the update rule is

$$\begin{aligned} \mathbf{v}_{n+1} &= \alpha \mathbf{v}_n - \epsilon \mathbf{g}_n, \\ \theta_{n+1} &= \theta_n + \mathbf{v}_{n+1}, \end{aligned} \tag{1.48}$$

where $\mathbf{v}_n$ is the velocity, which is initialized at zero. The reasoning behind this is that the velocity term keeps track of the history of the gradient, which leads to a speed up in convergence.

Another optimizer that is used in Chapter 6 for the Wasserstein GAN is RMSProp [12] [15]. Here the update rule is

$$\begin{aligned} \mathbf{r}_{n+1} &= \rho \mathbf{r}_n + (1-\rho) \mathbf{g}_n \odot \mathbf{g}_n \\ \theta_{n+1} &= \theta_n - \frac{\epsilon}{\sqrt{\delta + \mathbf{r}_{n+1}}} \odot \mathbf{g}_n \end{aligned} \tag{1.49}$$

where δ is a very small number, which avoids numerical issues and $\odot$ is an element wise product [16]. Similar to before $\mathbf{r}_n$ is initially at zero.

[14] This is the ceiling function, it rounds up to the nearest greater integer, i.e. ceil$(0.1) = 1$

[15] Famously introduced by Geoffrey Hinton during a lecture and never published.

[16] For two vectors $\mathbf{a}$ and $\mathbf{b}$ the element wise product is defined by $(\mathbf{a} \odot \mathbf{b})_i = a_i b_i$.

Probably the most popular algorithm for training neural networks is Adam [13] and it is the default optimizer used throughout this thesis. The name comes from 'adaptive moments'. Here the update rule is

$$\begin{aligned}\mathbf{s}_{n+1} &= \frac{\rho_1}{1-\rho_1^{n+1}}\mathbf{s}_n + \frac{1-\rho_1}{1-\rho_1^{n+1}}\mathbf{g}_n,\\ \mathbf{r}_{n+1} &= \frac{\rho_2}{1-\rho_2^{n+1}}\mathbf{r}_n + \frac{1-\rho_2}{1-\rho_2^{n+1}}\mathbf{g}_n \odot \mathbf{g}_n,\\ \theta_{n+1} &= \theta_n - \epsilon\frac{\mathbf{s}_{n+1}}{\sqrt{r_{n+1}}+\delta}.\end{aligned} \tag{1.50}$$

Again $\mathbf{s}_n$ and $\mathbf{r}_n$ are initialized at zero and δ is to prevent numerical errors.

A final consideration for the optimization is how to initialize the parameters in the network. Often the biases are set to zero and the weights are drawn from a distribution centred at zero with some chosen variance. The reason the weights are randomly assigned is if they were all initialized at the same value then the symmetry would never be broken during training and they would always remain the same leading to the network being unable to learn. This is due to the fact that the update to each weight would be the same for every weight in a given layer.

Finally for gradient based optimization it is important to make sure that all the features are the same scale, as this is empirically found to produce better results. One way to do this is standardizing the data. This is done by computing the mean and standard deviation of each feature from the training set and subtracting each feature of each sample by the mean and dividing by the standard deviation. It is important to compute the statistics from the training set so each transformation is the same whether its the training, validation or test set.

1.4.3 Regularization Methods

There are several methods available to prevent over-fitting in neural networks.

Firstly, similar to ridge regression we can implement a weight penalty in the cost function penalizing large values of the weights. This would be implemented as

$$J_{\text{reg}} = \lambda\sum_{i=0}^{L}\sum_{j=1}^{p^{(i+1)}}\sum_{k=1}^{p^{(i)}}(W_{jk}^{(i)})^2 = \lambda\sum_{i=0}^{L}|W^{(i)}|_F^2. \tag{1.51}$$

Of course we could have alternatives to the square of the weights, like the absolute value, $|W_{jk}^{(i)}|$ which as mentioned before leads to a sparse set of weights. These are also known as L2 and L1 regularisation respectively. Note that similar

to the case of ridge regression we do not penalize the bias vectors. This method tends not to be used often, due to the success of the next two methods.

The next method is called early stopping, the reasoning behind it is incredibly simple. The longer the neural network trains the most accustomed it becomes to the training set. Thus by monitoring the validation error over time we can halt training, when the validation error begins to rise - which implies over-fitting.

Finally we have dropout [14]. Here, during training the output of each of the nodes is randomly set to zero with some probability p. There are two different ways to interpret dropout. Firstly we can think of it as forcing the network to learn several ways of mapping the inputs to outputs, making it more robust. Another way to think about it as follows, given a network with α hidden units in total there are 2^α networks that can be formed by removing hidden units. Dropout consists of averaging over this ensemble of sub networks. When it comes to making a prediction we scale the weights of the network by p, there is a formal justification for this for some linear models but for deep networks the only justification is that it performs well [8]. In practice to implement dropout we have to make the network deeper then it would have been without dropout, since dropout diminishes the capacity of the network. An advantage of dropout is how simple it is to implement, however since we have to make the network deeper, it may not be feasible for already very large models.

1.4.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [15] are an architecture of neural networks specially designed for inputs with a grid structure such as images or time series data. Data with this topology arises often in physics due to the locality of many laws of nature.

To see why CNNs are useful consider a problem involving images, where the image dimensions are 1000×1000 pixels. In a fully connected network this means the feature vector will have $1,000,000$ components. Even if we have a small number of units in the first hidden layer, say 64, there will be over 64 million trainable parameters, which is very computationally expensive to train and is likely to overfit. Instead we may try to avoid this by hand designing features, i.e. creating high-level features relevant to the problem, however this is non-trivial and would not be a general solution to the problem.

The idea behind CNNs is that the network learns filters, called kernels, which are used to find relevant features. As their name implies, they achieve this through a convolution operation. To understand convolution we start with

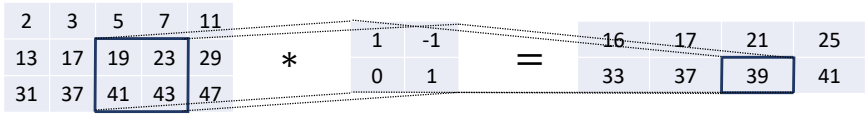


Figure 1.5: An example of a convolution operation in 2 dimensions. Here the input is a 3×5 matrix, the kernel a 2×2 matrix and the output a 2×4 matrix. When the highlighted segment is acted upon by the kernel the output is 39 since $19 - 23 + 43 = 39$.

an example in 1-dimension, this could be a time-signal, audio recording or a lattice model (as used in Chapter 3). Suppose we have a p-dimensional feature vector, $\mathbf{x}$, and a k-dimensional kernel vector $\mathbf{a}$. Then the convolutional operation results in a new vector, $\mathbf{s}$, with

$$s_i = (\mathbf{x} * \mathbf{a})_i = \sum_{j=1}^{k} x_{(i-1)s+j} a_j \tag{1.52}$$

and a dimension of $\frac{p-k}{s} + 1$ where s is the so-called stride parameter. For an explicit example suppose $\mathbf{a} = (1, -1)$, $s = 1$ and $p = 4$ then for a given $\mathbf{x}$ the operation results in

$$(x_1, x_2, x_3, x_4) * (1, -1) \to (x_1 - x_2, x_2 - x_3, x_3 - x_4). \tag{1.53}$$

Thus the convolution operation learns features that are relevant throughout the input, although for them to work there be must this local structure present. CNNs can be viewed in terms of fully connected networks, where the weight matrices are sparse and have many weights constrained to be the same value. Figure 1.5 shows how a 2 dimensional convolution is calculated.

Generally many kernels and several layers of convolution are used, with an activation function after each convolution layer. If we have k_1 kernels in the first convolution layer then the output will be a matrix with dimension $(k_1, \frac{p-k}{s} + 1)$. The convolution of the next layer is done on top of the previous layers. So for example if we have a kernel tensor A_{ijk} and the input to the layer C_{ij} then the outputted convolution is

$$(C * A)_{ij} = \sum_{lm} C_{l,(j-1)s+m} A_{ilm}. \tag{1.54}$$

Note that the stride, number of kernels and size of the kernels are hyper parameters, while the elements of the kernel are learnt via the optimizer. We

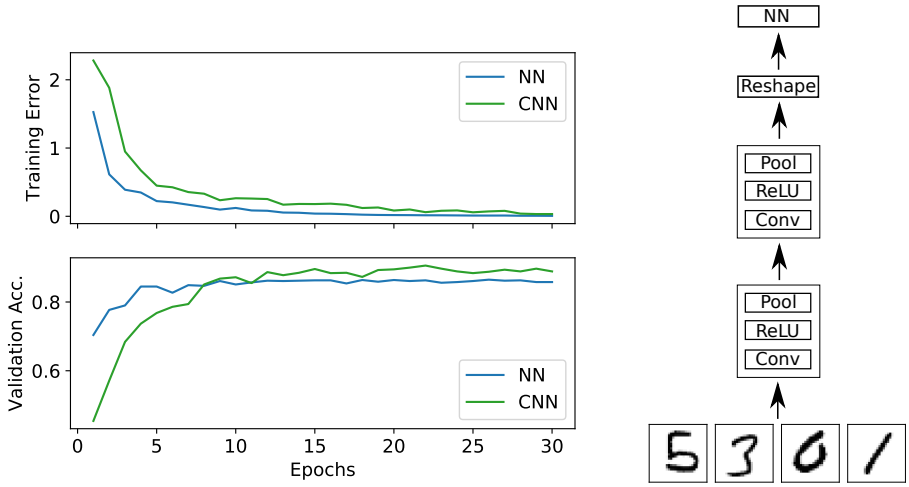


Figure 1.6: Left: A comparison of a CNN and a fully connected NN. The upper panel shows the training error and the lower the validation accuracy. Here the task is classifying digits from the MNIST dataset [16]. Not only does the CNN have a better validation accuracy, but it had a much smaller number of parameters using only 29,056 compared to 203,530 for the fully connected net. Right: The architecture of the CNN used. The inputs are the 2D pixels of the images. This then passes twice through a convolution layer (Conv), a ReLU activation function and then pooling (Pool). Finally it is reshaped into a vector and passed through a fully connected neural network.

can easily generalize this to 2 and higher dimensions. Often convolution is used with the pooling operation, where a region is summarized by its mean or maximum [8]. Pooling is used to reduce the size of the inputs, so that the feature vector passed to the fully connected network is not too large. The performance of a CNN versus a NN and an example architecture of a CNN are shown in Figure 1.6.

1.4.5 Recurrent Neural Networks

Another modified version of neural networks are recurrent neural networks (RNNs). These networks are designed for sequential data where we often have inputs of differing lengths. The idea behind RNNs is to use parameter sharing to allow the net to deal with different sized inputs [8]. RNNs allow neural networks to be very powerful, in fact they are equivalent to Turing machines and thus can be applied to any computational task. However they are notorious for

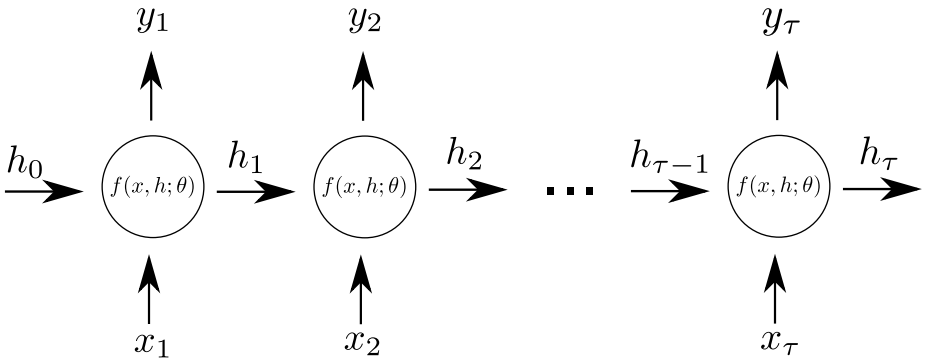


Figure 1.7: A simple recurrent neural network architecture with inputs x_i, outputs y_i and hidden states h_i. Here, given an input and a previous hidden state, the network, $f(x, h; \theta)$, predicts an output and the next hidden state.

being very hard to train, with gradients often diverging or vanishing to zero [17].

Figure 1.7 depicts a simple architecture of a RNN. Here we have an input sequence of length τ, $(x_1, x_2, ..., x_\tau)$. This could be time series data, a sentence, etc. For each input x_i the network predicts an output y_i and the hidden vector h_i. This hidden vector acts as a compressed representation of the previous sequence with h_0 often set to a vector of zeros. By using the same weights at each time step we can generalize to new sizes of input.

Suppose our task was sentiment analysis, here the RNN has to predict whether a sentence expresses positive or negative sentiment. Then we would ignore all of the y_i except y_τ which we would insert into the cost function, training it to be 1 for positive and 0 for negative sentiment. Or perhaps we want to train our net to learn the conditional probability distribution

$$P(x_1, x_2, ..., x_\tau) = P(x_1) \prod_{i=2}^{\tau} P(x_i | x_1 ... x_{i-1}) \tag{1.55}$$

then y_1 would be $P(x_1)$, y_2 would be $P(x_2|x_1)$ etc. Another modification we could do is to use the network to predict the next character, where now $y_1 = P(x_2|x_1)$ etc.

Many other variants of RNN are possible. For example to preform sequence to sequence prediction an encode-decoder architecture is used. Or to model long-term dependencies Long-Short-Term-Memory (LSTMs) models are used [8]. The fundamental idea they all share is that by reusing weights we can

[17] This happens because of the sequential parameter sharing. For a simple linear RNN without any bias vectors or inputs, $f(\mathbf{h}) = W\mathbf{h}$, the hidden-layer output after n recursions is $W^n\mathbf{h}$. Assuming for simplicity that W is diagonalizable and thus writing the eigendecomposition as $W = Q\Lambda Q^{-1}$ we see that if any eigenvalues are over 1 the output will exponentially diverge.

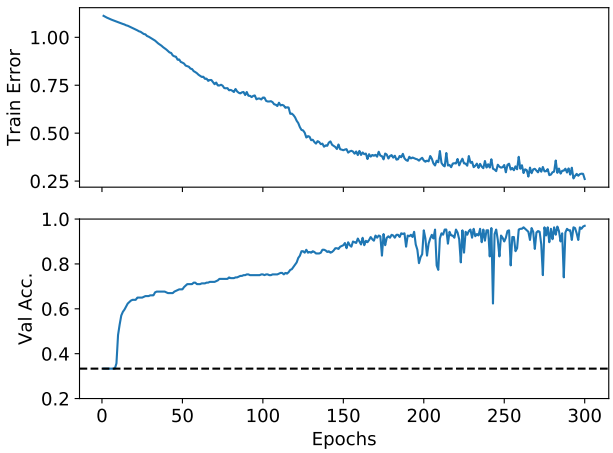


Figure 1.8: Here I show the RNN results on a toy sequence classification problem. The task here is to classify sequences of length τ into three categories: ascending, descending and neither. The top panel shows the training error on sequences with $\tau = 4, 5, 6$ and the lower panel shows the classification accuracy for $\tau = 7$. The dashed line corresponds to the accuracy random guessing would result in which is $1/3$ as there are three classes.

interact with data of differing lengths.

To illustrate the power of RNNs lets see a toy problem, the task of classifying sequences $(x_1, ..., x_\tau)$ into three categories: ascending $x_{i+1} > x_i$, descending $x_{i+1} < x_i$ and neither. For simplicity I set $x_i \in [0, 1]$. The RNN is trained on sequences with lengths between 4 and 6 and then validated on sequences of length 7. Figure 1.8 shows the results, the final RNN is able to correctly extrapolate to $\tau = 7$ with a classification error of only 5%.

1.5 Uncertainty

For practical usage it is often very important to have an estimate of how accurate the model's prediction is. This can let us know when to stop trusting the model and might be the difference between a model working in theory and practice.

A naive way to estimate the model error would be to, have two models, one to make predictions and another to estimate the first models error. This approach will potentially work well for data both models have seen before, however, given a new very different data-point, the second models estimation of the uncertainty will be unreliable. Here I detail how ensembling can be used as a robust method

for estimating the uncertainty [18].

We have already seen ensembling when describing random forests. The idea is to have many models, which are uncorrelated to each other. Suppose each model is parametrized by $\theta^{(i)}$, then when we want to make a prediction we take the average

$$\mu(\mathbf{x}) = \frac{1}{B}\sum_{i=1}^{B} f(\mathbf{x}|\theta^{(i)}). \tag{1.56}$$

As mentioned before the expected squared error for B ensembles is

$$E[(y - \mu(\mathbf{x}))^2] = \frac{\sigma^2 - \rho}{B} + \rho \tag{1.57}$$

where σ^2 is the expected individual squared error and ρ is the covariance between each model. If the models are uncorrelated we get an expected error lower than using a single model.

This method can be used to estimate the uncertainty as there is no reason for uncorrelated models to agree on data they have never seen before and thus will give different predictions, which leads to a high variance. We can measure this variance with

$$\sigma^2 = \frac{1}{B}\sum_{i=1}^{B}(\mu(\mathbf{x}) - f(\mathbf{x}|\theta^{(i)}))^2, \tag{1.58}$$

although this uncertainty estimation will be more qualitative than quantitative. However, if we just need it to tell us when our predictions are reliable this is fine.

There are many ways to create different models, one can use different algorithms or use bagging to create many datasets for example. In neural networks since the parameters are randomly initialized [19] several nets can be trained on the same dataset and given different predictions. Another method exclusive to neural nets is to use dropout. Recall that in dropout we randomly switch off activations in the neural network. Usually when we want to make predictions we rescale the weights by the probability they were to be turned off. However instead of rescaling, we can keep the dropout in the net and make many predictions and then ensemble them together. An example of using ensembling with neural networks to estimate the uncertainty is shown in Figure 1.9.

[18] Another very popular approach, which I do not discuss is Gaussian Process Regression [17]

[19] Selecting the batches randomly can add another source of randomness.

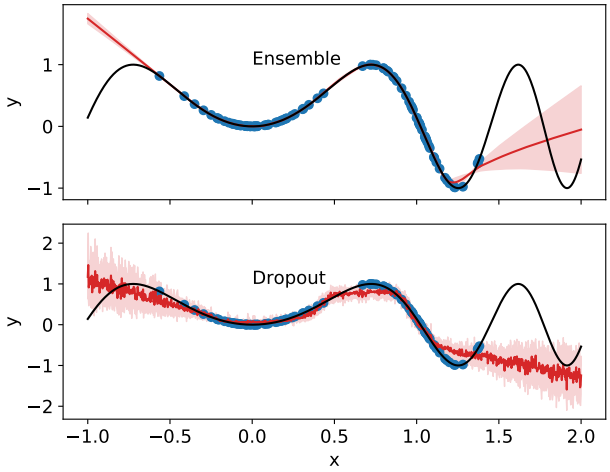


Figure 1.9: Confidence intervals for both an ensemble of neural nets and one neural net with dropout. The black solid line is the true function, the blue dots are the training set, the red solid line is the prediction and red shaded region the standard deviation of the predictions. The dataset was created by unevenly sampling the function $f(x) = \sin(3x^2)$.

1.6 Unsupervised Learning

The task of unsupervised learning is to find some structure in un-labelled (no targets) data. This is used in many applications for example: reducing the dimension of a dataset for visualization, finding anomalous data or generating new samples similar to ones in a dataset. If we have a very large un-labelled dataset then one could even use it to learn useful features with neural networks. Here I introduce three types of unsupervised learning: principle component analysis, autoencoders and generative adversarial networks.

1.6.1 Principle Component Analysis

Suppose we have a dataset of p-vectors $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$ and we want to compress them to k dimensions, $k < p$. The idea behind principle component analysis (PCA) is to find a linear mapping to a lower dimension that maximises the variance of the data.

For simplicity we assume that our data is centred with mean 0 along each of the p feature dimensions, $\frac{1}{N}\sum_{i=1}^{N} x_k^{(i)} = 0$. Thus the total variance along each feature is

$$\sigma^2 = \sum_{k=1}^{p} \frac{1}{N} \sum_{i=1}^{N} (x_k^{(i)})^2. \tag{1.59}$$

Starting with a reduced dimension of $k = 1$, our linear mapping is $y_i = \mathbf{w}^T\mathbf{x}^{(i)}$. The variance in this reduced space is $\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}^{(i)} - \bar{y})^2$ where $\bar{y} = \frac{1}{N}\mathbf{w}^T\sum_{i=1}^{N}\mathbf{x}^{(i)}$=0. Since this could be maximized by making $\mathbf{w}$ very large, we constrain it to be a unit vector. Letting $X = \sum_{i=1}^{N}\mathbf{x}^{(i)}\mathbf{x}^{(i)T}$, this results in the Lagrangian

$$L = \mathbf{w}^T X\mathbf{w} - \lambda(|\mathbf{w}|^2 - 1), \tag{1.60}$$

which we wish to maximise. Taking a derivative with respect to $\mathbf{w}$ results in

$$\nabla_{\mathbf{w}} L = 0 \quad \rightarrow \quad X\mathbf{w} = \lambda\mathbf{w}, \tag{1.61}$$

and thus we see $\mathbf{w}$ is an eigenvector of X. Normalizing it and subbing into the Lagrangian we see that it is the largest eigenvalue we want and the variance in the reduced dimension is the corresponding eigenvalue.

If we want to project to k dimensions we choose the k eigenvectors with the largest eigenvalues, as these are all orthogonal. Thus we use the matrix U_k where the columns of U_k are the k eigenvectors and if H is the matrix whose i^{th} row is $\mathbf{x}^{(i)T}$ then our transformation is

$$\underbrace{H}_{N\times p} \rightarrow \underbrace{HU_k}_{N\times k}. \tag{1.62}$$

Note that the total variance in the raw data, Equation (1.59), is given by $\frac{1}{N}\mathrm{Tr}(H^TH) = \frac{1}{N}\mathrm{Tr}(X)$. When $k = p$ the reduced space has the same variance as the original data as U_p is an orthogonal matrix. This leads to using the explained variance, the total variance in the reduced space over the total variance as a metric of how easy the data is to compress.

Figure 1.10 shows the results of using PCA to reduce the dimension of a dataset.

1.6.2 Autoencoders

An autoencoder consists of two neural networks, one an encoder $f(\mathbf{x}) = \mathbf{h}$ and the other a decoder $\mathbf{r} = g(\mathbf{h})$. Autoencoders are generally used for data denoising (here noise is added to the input and the autoencoder must learn to produce the original sample) and dimensionality reduction.

To perform dimensionality reduction, the autoencoder is trained to reproduce the input, with the standard loss function being

$$J = \sum_i |\mathbf{x}^{(i)} - g(f(\mathbf{x}^{(i)}))|^2. \tag{1.63}$$

However, the dimension of the representation $\mathbf{h}$ is chosen to be smaller than the input, thus creating an information bottleneck. The benefit of autoencoders is

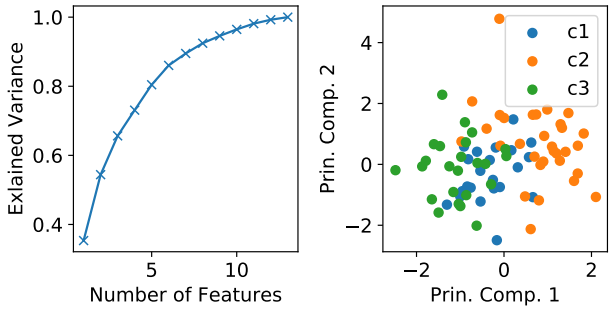


Figure 1.10: Here we use data from wine dataset from Ref. [18] which has 3 classes and 14 features. We split the dataset into training (size 100) and validation (size 78). Left Panel: Here we show the explained variance against the number of reduced features. We can see that over 60% of the variance can be kept with just 3 features. Right Panel: Here we visualize the dataset using the two principle components with the most variance on the validation set.

that unlike PCA they can learn non-linear mappings [20] and hence outperform them, however we must be careful not to overfit and learn a reduced representation which does not contain any useful information.

Figure 1.11 shows the autoencoder trained on the same dataset as the PCA was. As we can clearly see the learned reduced dimension is much more informative for the autoencoder than PCA, as it distinguishes more effectively the class labels.

1.6.3 Generative Adversarial Networks

GANs [19] are a class of methods, able to implicitly learn a distribution given a dataset. This means that while GANs can generate new samples, which are similar to the samples in the dataset, they do not represent the data distribution, p_{data}, explicitly.

They are comprised of two neural networks, which play a zero-sum game against each other. One network, the generator, tries to produce samples that look like those coming from the dataset. The other network, the discriminator, predicts whether a given sample is real or generated. The game terminates when the discriminator can no longer tell the difference between the generated and true data, with the resulting generator having learned the underlying distribution.

[20] A linear autoencoder can learn the same mapping as PCA.

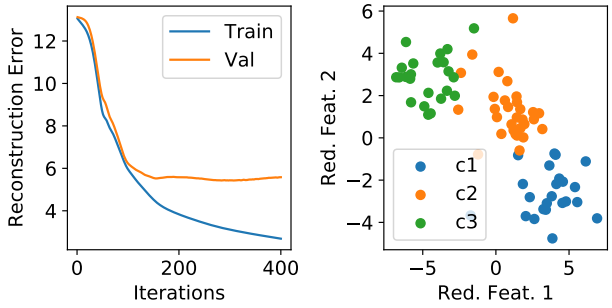


Figure 1.11: Here we again use data from the wine dataset from Ref. [18], splitting into a training set (size 100) and validation set (size 78). The size of the reduced dimension in the autoencoder was 2. Left Panel: Here we show the reconstruction error (Equation 1.63) over training iterations, note that we do start seeing overfitting after roughly 100 iterations. Right Panel: Here we visualize the validation dataset using the reduced representation.

The discriminator is a binary classifier, given a sample X its output is $D(X)$, which is 1 if it thinks the sample is real and 0 if it thinks the sample is generated. Letting p_{gen} be the distribution of the generated samples, the cost function for the discriminator - which recall it wants to minimise - is simply the binary cross entropy [Equation (1.17)]

$$J_D = -\frac{1}{2}\mathrm{E}_{X\sim p_{\text{data}}}[\log D(X)] - \frac{1}{2}\mathrm{E}_{X\sim p_{\text{gen}}}[\log(1-D(X))], \tag{1.64}$$

with the overall factor $1/2$ added so that at equilibrium the cost functions for the discriminator and generator are equal. Writing the discriminator cost function in integral form

$$J_D = -\frac{1}{2}\int dx\,[p_{\text{data}}(x)\log D(x) + p_{\text{gen}}(x)\log(1-D(x))]\,, \tag{1.65}$$

and by using the Euler-Lagrange equations we get the optimal discriminator policy

$$D^{\text{op}}(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_{\text{gen}}(x)}. \tag{1.66}$$

Thus by ensuring the discriminator is being optimally trained, it estimates the ratio between $p_{\text{data}}(x)$ and $p_{\text{data}}(x)+p_{\text{gen}}(x)$. It is easy to show by inserting Equation (1.66) into (1.64) that the cost function of the optimal discriminator is the Jensen-Shannon divergence between the generated and real data.

The generator takes in a latent vector [21], $\mathbf{z}$, which is randomly sampled from some distribution $p_z(\mathbf{z})$ (generally either normal or uniform) and returns

[21] A latent variable is simply a variable, which we cannot observe directly.

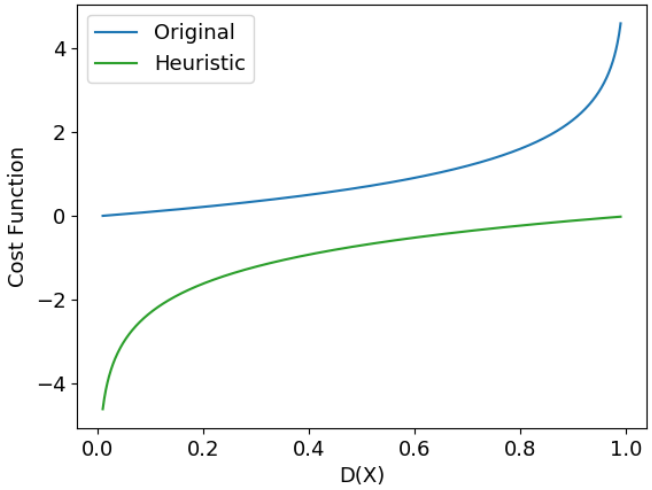


Figure 1.12: The original and heuristic cost functions for the generator against the output of the discriminator. The original cost function has a small gradient for small values of $D(X)$ and thus is generally replaced by the heuristic cost function, which has a much larger gradient for small values of $D(X)$.

an attempt at a realistic sample $G(\mathbf{z}) = X$. Thus, one can view the generator as transforming the latent distribution to the distribution of the dataset. As the generator wants to 'fool' the discriminator it has cost function

$$J_G = \mathrm{E}_{X\sim p_{\mathrm{gen}}}[\log(1 - D(X))]. \tag{1.67}$$

In practice, however, this cost function is not used, the reason for this is that the function $\log(1 - x)$ has a small derivative for values of x near zero. Thus at the start of the training, when the discriminator can easily tell the difference between the generated and real samples, there is not a strong enough gradient to update the weights of the generator to produce good samples. This motivates the heuristic cost function

$$J_G = -\mathrm{E}_{X\sim p_{\mathrm{gen}}}[\log D(X)], \tag{1.68}$$

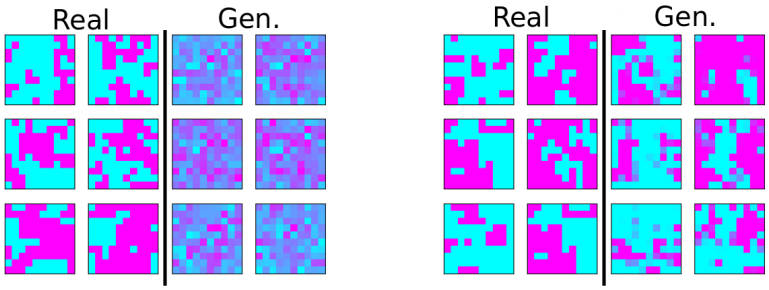
which, as Figure 1.12 shows, has a large gradient for small values of $D(X)$.

Nash equilibrium is reached when the discriminator cannot tell the difference between the real and generated data, and hence returns $1/2$ for all samples. Thus at equilibrium we get

$$J_D = J_G = \log 2 \approx 0.69, \tag{1.69}$$

which holds for both the original and heuristic generator cost function.

In order to demonstrate the effectiveness of GANs we apply them to a dataset of Ising model samples generated near the critical temperature. The probability of a state with energy E_i occurring at inverse temperature β is $p_i = e^{-\beta E_i}/Z$. Thus, in order for the GAN to model this, it must implicitly estimate the partition function, Z, which is intractable to compute for large lattices. Figure 1.13 shows the results, demonstrating that the GAN after a short amount of training is capable of producing nearly realistic samples.



Initially After 400 Epochs

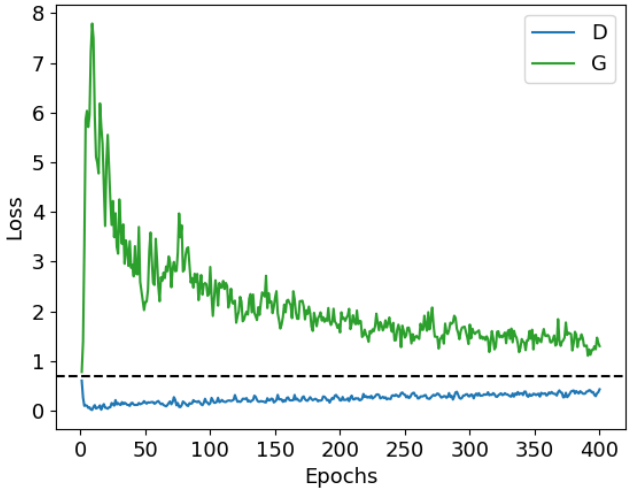


Figure 1.13: Here we show results relating to applying a GAN to a dataset of Ising model samples generated near the critical temperature. The top plots compare the real and generated samples initially and after 400 training epochs. The bottom plot shows the generator and discriminator cost functions over training epochs, with the dashed line corresponding to the Nash equilibrium value of log(2).

1.7 Summary

In this Chapter I have given an introduction to Machine Learning. Specifically, I looked at supervised learning, where the task is to predict some quantity given a dataset, and unsupervised learning, where the task is to find some structure in a given dataset. In the coming chapters, I will show how these models can be applied in physics, where they can be used to speed up existing methods and make discoveries.

Chapter 2

The Physics of Many Particles

> "The history of science, like the history of all human ideas, is a history of irresponsible dreams, of obstinacy, and of error. But science is one of the very few human activities - perhaps the only one - in which errors are systematically criticized and fairly often, in time, corrected. This is why we can say that, in science, we often learn from our mistakes, and why we can speak clearly and sensibly about making progress there."
>
> - Karl Popper, Conjectures and Refutations: The Growth of Scientific Knowledge

2.1 Introduction

This chapter serves as a brief introduction to many-body quantum mechanics and models of interacting electrons. It begins with outlining the relationship between the spin of electrons and magnetism. Then after introducing second quantization, we derive the Hubbard model, which forms the basis of our investigations in Chapter 3. The following section details the Heisenberg model, which is studied using machine learning in Chapter 4. Finally I end the chapter with a discussion of the difficulty of solving these models for large systems, which motivates the work of chapters 3 and 4.

For more detail on: the basics of quantum mechanics see chapter 2 of Nielsen et al. [20] or Merzbacher [21], the second quantization formalism see chapter 1 of Stefanucci et al. [22], the Hubbard model see Essler et al. [23] and the

Heisenberg model see Fazekas [24].

Throughout this chapter, I use natural units, setting the reduced Planck's constant and Boltzmann's constant to one, $\hbar = k_B = 1$.

2.2 The Magnetism of Electrons

From classical mechanics we know that the angular momentum of a particle with position $\mathbf{r}$ and momentum $\mathbf{p}$ is given by $\mathbf{L} = \mathbf{r} \times \mathbf{p}$. In quantum mechanics all measurable quantities are associated with Hermitian operators and since the position and momentum operators do not commute with

$$[\hat{r}_a, \hat{p}_a] = \hat{r}_a\hat{p}_a - \hat{p}_a\hat{r}_a = i \tag{2.1}$$

for $a \in [x, y, z]$, we have the relations

$$[\hat{L}_a, \hat{L}_b] = i\sum_c \epsilon_{abc}\hat{L}_c \tag{2.2}$$

for $a, b, c \in [x, y, z]$.

Another difference, unique to quantum mechanics, is the fact that electrons have an intrinsic angular momentum called spin. Letting an electron with spin-up be represented by $|\uparrow\rangle = \begin{pmatrix}1\\0\end{pmatrix}$ and with spin-down by $|\downarrow\rangle = \begin{pmatrix}0\\1\end{pmatrix}$, the general spin state - called a spinor - is the superposition of such states, namely

$$|\psi\rangle = a|\uparrow\rangle + b|\downarrow\rangle = \begin{pmatrix}a\\b\end{pmatrix} \tag{2.3}$$

with $|a|^2 + |b|^2 = 1$. The spin angular momentum operator is given by

$$\hat{\mathbf{S}} = \frac{1}{2}\hat{\boldsymbol{\sigma}} \tag{2.4}$$

where $\hat{\boldsymbol{\sigma}}$ is a vector of matrices $\hat{\boldsymbol{\sigma}} = (\hat{\sigma}_x, \hat{\sigma}_y, \hat{\sigma}_z)$ with the Pauli matrices defined as

$$\hat{\sigma}_x = \begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}, \quad \hat{\sigma}_y = \begin{pmatrix}0 & -i\\i & 0\end{pmatrix}, \quad \hat{\sigma}_z = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}. \tag{2.5}$$

All of the Pauli Matrices are all Hermitian and when squared give the identity matrix, $\hat{\sigma}_i^2 = \mathbb{1}$. We can compute the action of each matrix on the spin basis states with results:

$$\begin{aligned}
\hat{\sigma}_x|\uparrow\rangle &= |\downarrow\rangle \quad & \hat{\sigma}_x|\downarrow\rangle &= |\uparrow\rangle \\
\hat{\sigma}_y|\uparrow\rangle &= i|\downarrow\rangle \quad & \hat{\sigma}_y|\downarrow\rangle &= -i|\uparrow\rangle \\
\hat{\sigma}_z|\uparrow\rangle &= |\uparrow\rangle \quad & \hat{\sigma}_z|\downarrow\rangle &= -|\uparrow\rangle
\end{aligned}$$

Similarly to Equation (2.2), the spin operators obey the following commutation relations

$$[\hat{S}_a, \hat{S}_b] = i \sum_c \epsilon_{abc} \hat{S}_c \tag{2.6}$$

for $a, b, c \in [x, y, z]$. Often it is useful to define the raising and lowering operators, respectively

$$\hat{S}_+ = \hat{S}_x + i\hat{S}_y \quad , \quad \hat{S}_- = \hat{S}_x - i\hat{S}_y \tag{2.7}$$

which have the following actions on the spin states:

$$\begin{aligned} \hat{S}_+|\uparrow\rangle &= 0 \qquad \hat{S}_+|\downarrow\rangle = |\uparrow\rangle \\ \hat{S}_-|\uparrow\rangle &= |\downarrow\rangle \quad \hat{S}_-|\downarrow\rangle = 0 \end{aligned}$$

To end this section, I proceed to derive the Hamiltonian for an electron in a magnetic field, $\mathbf{B}$. Consider first the classical problem, ignoring quantum mechanics. As there are no magnetic monopoles, the net magnetic flux of $\mathbf{B}$ through any surface S must be zero, $\int_S \mathbf{B} \cdot d\mathbf{A} = 0$, as a result that follows directly from the divergence theorem $\nabla \cdot \mathbf{B} = 0$. This in turn implies that we can write $\mathbf{B} = \nabla \times \mathbf{A}$ where $\mathbf{A}$ is the magnetic vector potential. An electron in a magnetic field experiences the Lorentz force, $\mathbf{F} = -e\mathbf{v} \times \mathbf{B}$, which can be rewritten as $\frac{d}{dt}(m\mathbf{v} - e\mathbf{A}) = -\nabla(e\mathbf{v} \cdot \mathbf{A})$. Thus the momentum of an electron in a magnetic field is $\mathbf{p} = m\mathbf{v} - e\mathbf{A}$ with kinetic energy

$$H = \frac{1}{2}m|\mathbf{v}|^2 = \frac{1}{2m}|e\mathbf{A} + \mathbf{p}|^2. \tag{2.8}$$

Returning to the quantum problem, we choose the Coulomb Gauge [1] so that $\hat{\mathbf{p}}$ and $\hat{\mathbf{A}}$ commute. Expanding and using $\hat{\mathbf{A}} = \frac{1}{2}\mathbf{B} \times \hat{\mathbf{r}}$ the Hamiltonian becomes

$$\hat{H} = \frac{1}{2m}|\hat{\mathbf{p}}|^2 + \frac{e}{2m}\hat{\mathbf{p}} \cdot (\mathbf{B} \times \hat{\mathbf{r}}) + \frac{e^2}{8m}|\mathbf{B} \times \hat{\mathbf{r}}|^2. \tag{2.9}$$

The second term is the dominant correction to the hamiltonian due to the magnetic field. We can rewrite it by noticing that $\hat{\mathbf{p}} \cdot (\mathbf{B} \times \hat{\mathbf{r}}) = \mathbf{B} \cdot (\hat{\mathbf{r}} \times \hat{\mathbf{p}}) = \mathbf{B} \cdot \hat{\mathbf{L}}$. If we insert the spin angular component we get the Zeeman hamiltonian

$$\hat{H}_{\text{Zeeman}} = \frac{2}{2m}\mathbf{B} \cdot (\hat{\mathbf{L}} + 2\hat{\mathbf{S}}), \tag{2.10}$$

where the factor of 2 in the spin term, has its origins in relativistic quantum mechanics [25].

[1] $\mathbf{A}$ is unique up to a gradient since $\nabla \times (\mathbf{A} + \nabla f) = \nabla \times \mathbf{A}$. In Coulomb Gauge, $\nabla \cdot \mathbf{A} = 0$, this implies $\nabla^2 f = -\nabla \cdot \mathbf{A}$. A convenient expression for $\mathbf{A}$ in Coulomb Gauge is $\mathbf{A} = \frac{1}{2}\mathbf{B} \times \mathbf{r}$.

2.3 The Hubbard Model

2.3.1 Second quantizing the Hamiltonian

We now consider a solid comprised of N electrons and L nuclei with the structure defined by lattice vectors $\{\mathbf{a}_1, ..., \mathbf{a}_d\}$ where d is the dimension of the solid. To render this problem easier to solve we make the Born-Oppenheimer approximation and assume the nuclei are static. This can be justified on the grounds that electrons move much faster than the nuclei. Then then resulting classical hamiltonian for the electrons is

$$H = \sum_{i=1}^{N} \left(\frac{|\mathbf{p}_i|^2}{2m} - \sum_{j=1}^{L} \frac{Z_j e^2}{|\mathbf{r}_i - \mathbf{R}_j|} \right) + \frac{1}{2} \sum_{i,j=1, i\neq j}^{N} \frac{e^2}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{2.11}$$

where $\mathbf{p}_i$ is the momentum of the i^{th} electron, $\mathbf{x}_i$ is the position of the i^{th} electron, m is the electron mass and Z_i is the atomic number of the nuclei at position $\mathbf{R}_i$. The first term on the right-hand side is a sum of single particle terms

$$h_1(\mathbf{r}, \mathbf{p}) = \frac{|\mathbf{p}|^2}{2m} - \sum_{j=1}^{L} \frac{Z_j e^2}{|\mathbf{r} - \mathbf{R}_j|} \tag{2.12}$$

and the second term contains the electron-electron interactions

$$U(\mathbf{r}, \mathbf{s}) = \frac{e^2}{|\mathbf{r} - \mathbf{s}|}. \tag{2.13}$$

We now proceed to solve the non-interacting part of the hamiltonian, $\hat{H}_1 = \sum_{i=1}^{N} \hat{h}_1(\hat{\mathbf{r}}_i, \hat{\mathbf{p}}_i)$. Suppose we have 2 particles with one in eigenstate n with energy ϵ_n and another in eigenstate m with energy ϵ_m. Then the total energy will be $\epsilon_n + \epsilon_m$ with wavefunction

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|n\rangle|m\rangle \pm |m\rangle|n\rangle) \tag{2.14}$$

with the plus sign for bosons and the minus sign for fermions (see Appendix A.1). In general, for N such particles the energy will be $\sum_{i=1}^{N} \epsilon_i$ with wavefunction

$$|\psi\rangle = \frac{1}{\sqrt{N!}} \sum_{P} (\pm 1)^P |n_{P(1)}\rangle ... |n_{P(N)}\rangle \tag{2.15}$$

where $P(i)$ is the permutation operator and the sign depends on the parity of the number of permutations. Thus we can write the non-interacting hamiltonian in the form

$$\begin{aligned} \hat{H}_1 &= \sum_n \epsilon_n \hat{c}_n^\dagger \hat{c}_n = \sum_n \int d\mathbf{x} d\mathbf{y} \hat{\psi}^\dagger(\mathbf{x}) \langle \mathbf{x}|\hat{h}_1|n\rangle \langle n|\mathbf{y}\rangle \hat{\psi}(\mathbf{y}) \\ &= \int d\mathbf{x} d\mathbf{y} \hat{\psi}^\dagger(\mathbf{x}) \langle \mathbf{x}|\hat{h}_1|\mathbf{y}\rangle \hat{\psi}(\mathbf{y}) = \int d\mathbf{x} \hat{\psi}^\dagger(\mathbf{x}) \hat{h}_1(\mathbf{r}, -i\nabla) \hat{\psi}(\mathbf{x}) \end{aligned} \tag{2.16}$$

where $\hat{c}_n^\dagger$ is the creation operator for state n, $\hat{\psi}^\dagger(\mathbf{x})$ is the field operator for a particle at position $\mathbf{r}$ and spin σ, and $\int d\mathbf{x} = \sum_\sigma \int d\mathbf{r}$, see Appendix A.2 for more detail. Finally we have also used the fact that the momentum operator in quantum mechanics is $\hat{\mathbf{p}} = -i\nabla$.

It is then straightforward to write the two body operator in second quantization

$$\begin{aligned}\frac{1}{2}\sum_{i,j=1,i\neq j}^{N}\hat{U}(\hat{\mathbf{r}}_i,\hat{\mathbf{r}}_j) &= \frac{1}{2}\int d\mathbf{x}d\mathbf{y}\hat{U}(\mathbf{x},\mathbf{y})\hat{n}(\mathbf{x})\hat{n}(\mathbf{y}) - \frac{1}{2}\int d\mathbf{x}\hat{U}(\mathbf{x},\mathbf{x})\hat{n}(\mathbf{x}) \\ &= \frac{1}{2}\int d\mathbf{x}d\mathbf{y}\hat{U}(\mathbf{x},\mathbf{y})\hat{\psi}(\mathbf{x})^\dagger\hat{\psi}(\mathbf{y})^\dagger\hat{\psi}(\mathbf{y})\hat{\psi}(\mathbf{x}),\end{aligned} \tag{2.17}$$

which can be verified by using Equation A.14. Here $\hat{n}(\mathbf{x})$ is the density operator $\hat{n}(\mathbf{x}) = \hat{\psi}^\dagger(\mathbf{x})\hat{\psi}(\mathbf{x})$.

In the case we are treating a periodic system, the periodicity of the solid has important consequences for the wavefunction. Inferred by translational invariance, the hamiltonian commutes with the translation operator $\hat{T}_\mathbf{n}$ where $\hat{T}_\mathbf{n}\psi(\mathbf{r}) = \psi(\mathbf{r}+T_\mathbf{n})$ and $T_\mathbf{n} = n_1\mathbf{a}_1 + ... + n_d\mathbf{a}_d$. Note here that $\psi(\mathbf{r}) = \langle\mathbf{r}|\psi\rangle$. Thus, the eigenstates of $\hat{H}$ can be chosen to be eigenstates of $\hat{T}_\mathbf{n}$. The eigenvalues of $\hat{T}_\mathbf{n}$, $t_\mathbf{n}$, must satisfy $t_{\mathbf{n}_1+\mathbf{n}_2} = t_{\mathbf{n}_1}t_{\mathbf{n}_2}$. Thus, we can write $t_\mathbf{n} = t_{\mathbf{a}_1}^{n_1}...t_{\mathbf{a}_d}^{n_d}$ and requiring the operator to be bounded we get $t_{\mathbf{a}_j} = e^{i2\pi y_j}$. By applying periodic boundary conditions we require $t_{\mathbf{a}_j}^{N_j} = 1$, where N_j is the number of fundamental translations of $t_{\mathbf{a}_j}$ in one period. Thus, $y_j = 1/N_j$ and writing $\mathbf{k} = (n_1/N_1)\mathbf{b}_1 + ... + (n_d/N_d)\mathbf{b}_d$ where $\mathbf{b}_i \cdot \mathbf{a}_j = 2\pi\delta_{ij}$ we arrive at

$$\hat{T}_\mathbf{n}\psi(\mathbf{r}) = t_\mathbf{n}\psi(\mathbf{r}) = e^{i\mathbf{k}\cdot T_\mathbf{n}}\psi(\mathbf{r}) \tag{2.18}$$

and from this it follows that we can write

$$\psi(\mathbf{r}) = e^{i\mathbf{k}\cdot\mathbf{r}}u(\mathbf{r}), \tag{2.19}$$

where $u(\mathbf{r})$ is periodic $u(\mathbf{r}+T_\mathbf{n}) = u(\mathbf{r})$. This result is known as Bloch's Theorem and we now use this to construct a basis, where to expand the wavefunction and the operators.

We let $|\alpha,\mathbf{k}\rangle$ be the eigenstates of the single particle hamiltonian, i.e. $\hat{h}_1|\alpha,\mathbf{k}\rangle = e_{\alpha,\mathbf{k}}|\alpha,\mathbf{k}\rangle$. We know from Bloch's Theorem that $\langle\mathbf{r}|\alpha,\mathbf{k}\rangle = e^{i\mathbf{k}\cdot\mathbf{r}}u_{\alpha\mathbf{k}}(\mathbf{r})$. We introduce Wannier functions [26], $\phi_\alpha(\mathbf{r}-\mathbf{R}_i)$, centred about atom $\mathbf{R}_i$ with

$$\phi_\alpha(\mathbf{r}) = \frac{1}{\sqrt{L}}\sum_\mathbf{k}\langle\mathbf{r}|\alpha,\mathbf{k}\rangle \tag{2.20}$$

and $\langle\mathbf{r}|\alpha,\mathbf{k}\rangle = \frac{1}{\sqrt{L}}\sum_j e^{i\mathbf{k}\cdot\mathbf{R}_j}\phi_\alpha(\mathbf{r}-\mathbf{R}_j)$. Finally we introduce operators, which create Bloch states with spin σ, $\hat{c}^\dagger_{\alpha,\mathbf{k},\sigma}|0\rangle = |\alpha,\mathbf{k},\sigma\rangle$ and their Fourier Transforms $\hat{c}^\dagger_{\alpha,j,\sigma} = \frac{1}{\sqrt{L}}\sum_\mathbf{k} e^{-i\mathbf{k}\cdot\mathbf{R}_j}\hat{c}^\dagger_{\alpha,\mathbf{k},\sigma}$. Since $|\alpha,\mathbf{k},\sigma\rangle$ form a basis,

$|\mathbf{x}\rangle = \sum_{\alpha,\mathbf{k}} |\alpha, \mathbf{k}, \sigma\rangle\langle\alpha, \mathbf{k}, \sigma|\mathbf{x}\rangle$ implies

$$\hat{\psi}^\dagger(\mathbf{x}) = \sum_{\alpha,\mathbf{k}} \langle\alpha, \mathbf{k}, \sigma|\mathbf{x}\rangle \hat{c}^\dagger_{\alpha,\mathbf{k},\sigma} = \sum_{j,\alpha} \phi^*_\alpha(\mathbf{r} - \mathbf{R}_j)\hat{c}^\dagger_{\alpha,j,\sigma}. \tag{2.21}$$

The end result of this is that we can write the field operators in terms of Wannier functions, which we insert into the second quantized forms of the hamiltonian getting

$$\hat{H} = \sum_\alpha \sum_{i,j} \sum_s t^\alpha_{ij} \hat{c}^\dagger_{\alpha i s} \hat{c}_{\alpha j s} + \frac{1}{2} \sum_{\alpha\beta\gamma\zeta} \sum_{ijkl} \sum_{ss'} U_{\alpha\beta\gamma\zeta ijkl} \hat{c}^\dagger_{\alpha i s} \hat{c}^\dagger_{\beta j s'} \hat{c}_{\gamma k s'} \hat{c}_{\zeta l s}. \tag{2.22}$$

This defines: the onsite-energies

$$v^\alpha_j = t^\alpha_{jj} = \int d\mathbf{r} \phi^*_\alpha(\mathbf{r} - \mathbf{R}_j)\hat{h}_1(\mathbf{r}, -i\nabla)\phi_\alpha(\mathbf{r} - \mathbf{R}_j), \tag{2.23}$$

the hopping terms

$$t^\alpha_{ij} = \int d\mathbf{r} \phi^*_\alpha(\mathbf{r} - \mathbf{R}_i)\hat{h}_1(\mathbf{r}, -i\nabla)\phi_\alpha(\mathbf{r} - \mathbf{R}_j) \tag{2.24}$$

and finally the interaction terms

$$U_{\alpha\beta\gamma\zeta ijkl} = \int d\mathbf{r} d\mathbf{s} \phi^*_\alpha(\mathbf{r} - \mathbf{R}_i)\phi^*_\beta(\mathbf{s} - \mathbf{R}_j)\hat{U}(\mathbf{r}, \mathbf{s})\phi_\gamma(\mathbf{s} - \mathbf{R}_k)\phi_\zeta(\mathbf{r} - \mathbf{R}_l). \tag{2.25}$$

We now make several assumptions to simplify this model. Firstly, we only consider a single band and hence lose the α label. Secondly, we only consider onsite interactions for the U term and hence we have only a single term in the spin sum. For convenience we absorb the factor of 2 into the definition of U. Finally, we consider only nearest neighbour hopping, all of which results in

$$\hat{H} = \sum_{i=1}^{L} \sum_\sigma v_i \hat{n}_{i\sigma} - t \sum_{\langle i,j\rangle} \sum_\sigma \hat{c}^\dagger_{i\sigma} \hat{c}_{j\sigma} + U \sum_{j=1}^{L} \hat{n}_{j\uparrow} \hat{n}_{j\downarrow}. \tag{2.26}$$

This is the so-called Hubbard model and is shown graphically in Figure 2.1.

To solve the model we must introduce a suitable basis set. Here I follow Essler et al. [23] by defining

$$|\mathbf{x}, \mathbf{a}\rangle = \hat{c}^\dagger_{x_N, a_N} ... \hat{c}^\dagger_{x_1, a_1} |0\rangle \tag{2.27}$$

where $x_j \in \{1, ..., L\}$ and $a_j \in \{\uparrow, \downarrow\}$. The set $\{|\mathbf{x}, \mathbf{a}\rangle\}$ can be made a basis by enforcing the convention $x_{j+1} \geq x_j$ and in the case $x_{j+1} = x_j$ having the up-spin creation operator left most in Equation (2.27). Thus for example the state $|\downarrow, 0, \uparrow\downarrow\rangle$ corresponds to $\mathbf{x} = (1, 3, 3)$ and $\mathbf{a} = (\downarrow, \downarrow, \uparrow)$ with $|\downarrow, 0, \uparrow\downarrow\rangle = \hat{c}^\dagger_{3,\uparrow} \hat{c}^\dagger_{3,\downarrow} \hat{c}^\dagger_{1,\downarrow} |0\rangle$.

If we define the total spin operator as

$$\hat{N}_\sigma = \sum_{i=1}^{L} \hat{n}_{i\sigma}, \tag{2.28}$$

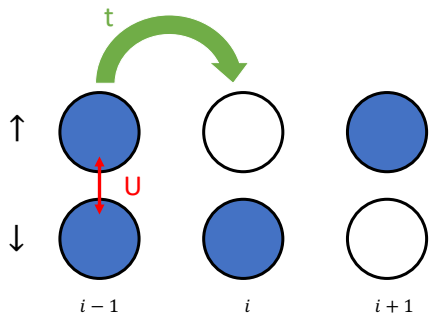


Figure 2.1: Here is a representation of the Hubbard model, for simplicity disorder is not shown (all the v_i are set to zero). Each site can be occupied by a maximum of two electrons, one with spin-up and another with spin-down. In the doubly occupied case there is an energy penalty of U. An electron can jump to an unoccupied neighbouring site, with this process being mediated by the hopping parameter t.

then we see that this commutes with the Hubbard hamiltonian, $[\hat{H}, \hat{N}_\sigma]$. As a consequence of this the total spin and number of particles will always be conserved. Thus if we have $N_\uparrow$ up-electrons and $N_\downarrow$ down-electrons we only have to consider the Hilbert space with that same number of particles and magnetization.

To gain some intuition about the Hubbard model I will now solve it in two limits.

2.3.2 The Non-interacting Limit

When U is zero the particles are non-interacting and we can simply solve the one-particle hamiltonian and use Pauli's exclusion principle to ascertain the configuration of the many particle system. Since we are in the one-particle sector, we drop the spin subscript and define $|j\rangle$ as the particle at site j. With periodic boundary conditions, such that $|L+1\rangle = |1\rangle$, and restricting our attention to one dimension the hamiltonian in first quantization reads

$$\hat{H} = -t\sum_{j=1}^{L}(|j\rangle\langle j+1| + |j+1\rangle\langle j|) + \sum_{j=1}^{L} v_j |j\rangle\langle j|. \tag{2.29}$$

For the homogeneous case with $v_j = v_0$ the hamiltonian is easily solved with the wavefunction

$$|\psi_k\rangle = \frac{1}{\sqrt{L}}\sum_{j=1}^{L} e^{ikj}|j\rangle \tag{2.30}$$

leading to the energy

$$E = v_0 - 2t\cos k. \tag{2.31}$$

Since we have periodic boundary conditions $\langle L+1|\psi_k\rangle = \langle 1|\psi_k\rangle$ and hence $k = 2\pi n/L$ with $n \in N$. Further since $|\psi_k\rangle = |\psi_{k+2\pi}\rangle$ values of n differing by a factor of L are equivalent. Hence the ground-state of the N particle system is found by populating the N lowest states with no two particles having the same value of n and spin.

For the inhomogeneous case in general we must diagonalize the full hamiltonian. However, if we have a periodic unit, then we just need to deal with that unit. For example consider a chain of L atoms (L even) with two different species of atoms interlaced with onsite energies of $-\Delta$ and Δ [$v_j = \Delta(-1)^j$], then using the anstaz

$$|\psi_k\rangle = a\sum_{j=1}^{L/2} e^{ik(2j-1)}|j\rangle + b\sum_{j=1}^{L/2} e^{ik2j}|j\rangle \tag{2.32}$$

we get

$$E\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} -\Delta & -2t\cos k \\ -2t\cos k & \Delta \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} \tag{2.33}$$

and thus

$$E = \pm\sqrt{\Delta^2 + 4t^2\cos^2 k}. \tag{2.34}$$

Hence, by adding inhomogeneities we get a gap in the allowed energies. For the N particle configuration we proceed as before except now $|\psi_k\rangle = |\psi_{k+\pi}\rangle$ and hence, values of n differing by a factor of $L/2$ are now equivalent.

For the homogeneous and inhomogeneous case the energies are plotted against the k-value in Figure 2.2. Also shown is the ground-state configuration for $L = 10$ and $N = 12$.

2.3.3 Mean-Field Theory for the Hubbard dimer

The mean-field theory for the Hubbard model consists of approximating the interaction terms as

$$\begin{aligned} \hat{n_\uparrow}\hat{n_\downarrow} &= \left(\langle\hat{n}_\uparrow\rangle + [\hat{n}_\uparrow - \langle\hat{n}_\uparrow\rangle]\right)\left(\langle\hat{n}_\downarrow\rangle + [\hat{n}_\downarrow - \langle\hat{n}_\downarrow\rangle]\right) \\ &\approx \hat{n}_\uparrow\langle\hat{n}_\downarrow\rangle + \hat{n}_\downarrow\langle\hat{n}_\uparrow\rangle - \langle\hat{n}_\uparrow\rangle\langle\hat{n}_\downarrow\rangle \end{aligned} \tag{2.35}$$

namely we have assumed that the fluctuation-fluctuation term, $(\hat{n}_\uparrow - \langle\hat{n}_\uparrow\rangle)(\hat{n}_\downarrow - \langle\hat{n}_\downarrow\rangle)$, is small. If we make the assumption of uniformity $\langle\hat{n}_{j\sigma}\rangle = N_\sigma/L$ then the interaction hamiltonian becomes

$$\hat{H}_U = U\sum_{i=1}^{L}\hat{n}_{i\uparrow}\hat{n}_{i\downarrow} \approx U\frac{N_\uparrow N_\downarrow}{L}. \tag{2.36}$$

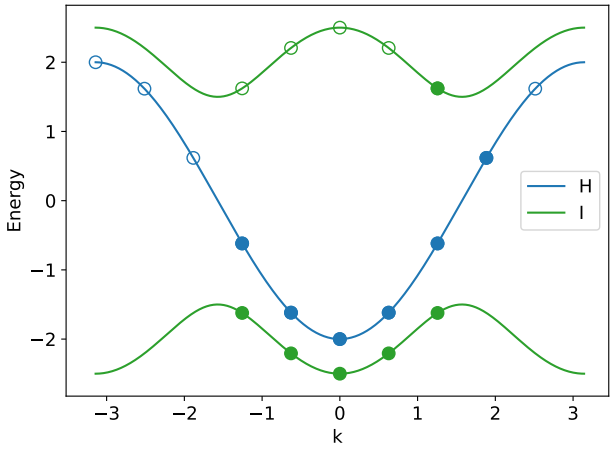


Figure 2.2: Here the energy is plotted against the k value for both the homogeneous (H) and inhomogeneous (I) case, where the later case corresponds to $v_j = (-1)^j \Delta$. The continuous line is for the infinite L limit and the circles correspond to the allowed energies for the $L = 10$ case. The filled in circles are the occupied states for the ground-state for $L = 10$ and $N = 12$. Here each filled in circle is doubly occupied with one electron with spin up and one with spin down.

To see how accurate mean field theory is we compare it to the exact solution for the two-site case (dimer) at half filling, $N = L = 2$. Using the previously introduced basis convention [Equation (2.27)] we write:

Number	
1	$\vert\uparrow\downarrow\rangle = \hat{c}^\dagger_{2\downarrow}\hat{c}^\dagger_{1\uparrow}\vert 0\rangle$
2	$\vert\downarrow\uparrow\rangle = \hat{c}^\dagger_{2\uparrow}\hat{c}^\dagger_{1\downarrow}\vert 0\rangle$
3	$\vert d0\rangle = \hat{c}^\dagger_{1\uparrow}\hat{c}^\dagger_{1\downarrow}\vert 0\rangle$
4	$\vert 0d\rangle = \hat{c}^\dagger_{2\uparrow}\hat{c}^\dagger_{2\downarrow}\vert 0\rangle$
5	$\vert\uparrow\uparrow\rangle = \hat{c}^\dagger_{2\uparrow}\hat{c}^\dagger_{1\uparrow}\vert 0\rangle$
6	$\vert\downarrow\downarrow\rangle = \hat{c}^\dagger_{2\downarrow}\hat{c}^\dagger_{1\downarrow}\vert 0\rangle$

Here "d" means that the site is doubly occupied and we have again used the convention of ordering the second site first and the up-spin before the down-spin. It is vitally important that once a convention is chosen it is stuck to. Assuming $v_1 = v_2 = 0$ the hamiltonian matrix writes

$$\hat{H} = \left(\begin{array}{cccc|cc} 0 & 0 & -t & -t & 0 & 0 \\ 0 & 0 & t & t & 0 & 0 \\ -t & t & U & 0 & 0 & 0 \\ -t & t & 0 & U & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right), \tag{2.37}$$

where we have divide the matrix into sectors that do not mix with each other. A comparison between the mean field theory approximated eigenvalues and the exact eigenvalues is shown in Table 2.1. From this we see that the mean field approximation becomes increasing inaccurate as U gets larger, with the it underestimating the two lowest eigenvalues and overestimating the largest two eigenvalues.

2.4 The Heisenberg Model

2.4.1 From Hubbard to Heisenberg

When the U term in the Hubbard model is large the low energy solutions will avoid having any double occupancy, thus it is possible to derive effective models in this limit. At half filling with $N = L$ the effective model is the Heisenberg model, which writes

$$\hat{H} = -J \sum_{\langle i,j \rangle} \hat{\mathbf{S}}^{(i)} \cdot \hat{\mathbf{S}}^{(j)} \tag{2.38}$$

No.	Exact eigenvalue	MFT eigenvalue	Difference
1	$U/2 - \sqrt{(U/2)^2 + 4t^2}$	$U/2 - 2t$	$2t - \sqrt{(U/2)^2 + 4t^2}$
2	0	$U/2$	$-U/2$
3	0	0	0
4	0	0	0
5	U	$U/2$	$U/2$
6	$U/2 + \sqrt{(U/2)^2 + 4t^2}$	$U/2 + 2t$	$\sqrt{(U/2)^2 + 4t^2} - 2t$

Table 2.1: For the Hubbard dimer we list the exact eigenvalues in increasing order. Also shown is the mean field theory (MFT) approximated eigenvalues and the difference between the exact and MFT values.

No.	**Exact eigenvector** (Unnormalized)
1	$\frac{2t}{-U/2+\sqrt{(U/2)^2+4t^2}}(\lvert\uparrow\downarrow\rangle - \lvert\downarrow\uparrow\rangle) + \lvert d0\rangle + \lvert 0d\rangle$
2	$\lvert\uparrow\downarrow\rangle + \lvert\downarrow\uparrow\rangle$
3	$\lvert\uparrow\uparrow\rangle$
4	$\lvert\downarrow\downarrow\rangle$
5	$\lvert d0\rangle - \lvert 0d\rangle$
6	$\frac{2t}{-U/2+\sqrt{(U/2)^2+4t^2}}(-\lvert\uparrow\downarrow\rangle + \lvert\downarrow\uparrow\rangle) + \lvert d0\rangle + \lvert 0d\rangle$

Table 2.2: For the Hubbard dimer we list the exact unnormalized eigenvectors in increasing order of eigenvalue. The corresponding eigenvalues are listed in Table 2.1.

where $J = -4t^2/U$ [2]. The derivation for an arbitrary number of sites is given in Appendix B, here we demonstrate it for the two site case.

To do this, let us take the large U limit of the exact results listed in Table 2.1. The 5^{th} and 6^{th} eigen-energies go off to infinity, so we ignore them as we are considering the low energy limit. Next we use the approximation

$$\sqrt{(U/2)^2 + 4t^2} = \frac{U}{2}\sqrt{1 + \left(\frac{4t}{U}\right)^2} \approx \frac{U}{2} + \frac{4t^2}{U} \tag{2.39}$$

which allows us to write the lowest energy term as $-4t^2/U$. The exact eigenstates are listen in Table 2.2 and we see that using the above approximation the lowest eigenstates converges to $1/\sqrt{2}(\lvert\uparrow\downarrow\rangle - \lvert\downarrow\uparrow\rangle)$ in the large-U limit. We

[2] In one dimension with L sites this simplifies to $\hat{H} = -2J\sum_{i=1}^{L} \hat{\mathbf{S}}^{(i)} \cdot \hat{\mathbf{S}}^{(i+1)}$ where the factor of 2 comes from double counting.

note that each of these states is also an eigenstate of $\hat{\mathbf{S}}^{(1)} \cdot \hat{\mathbf{S}}^{(2)}$. To summarize, we have:

State	Energy	S_z	$\hat{\mathbf{S}}^{(1)} \cdot \hat{\mathbf{S}}^{(2)}$
$\vert\uparrow\uparrow\rangle$	0	1	1/4
$\vert\downarrow\downarrow\rangle$	0	−1	1/4
$1/\sqrt{2}(\vert\uparrow\downarrow\rangle + \vert\downarrow\uparrow\rangle)$	0	0	1/4
$1/\sqrt{2}(\vert\uparrow\downarrow\rangle - \vert\downarrow\uparrow\rangle)$	$-4t^2/U$	0	−3/4

Thus we can reproduce the energy spectrum using the hamiltonian

$$H = -J(\hat{\mathbf{S}}^{(1)} \cdot \hat{\mathbf{S}}^{(2)} - 1/4) \tag{2.40}$$

where $J = -4t^2/U$. Removing the constant term leads to the Heisenberg model for two sites.

We could make the Heisenberg model more complex by adding a magnetic field through the Zeeman term

$$\hat{H}_{\text{Zeeman}} = -\hat{\mathbf{S}} \cdot \mathbf{B}, \tag{2.41}$$

where we have defined the total spin

$$\hat{\mathbf{S}} = \sum_{j=1}^{N} \hat{\mathbf{S}}^{(j)}. \tag{2.42}$$

Or we could have J depend on direction

$$\hat{H} = -\sum_{\langle i,j\rangle} \sum_{a=x,y,z} J_a \hat{S}_a^{(i)} \hat{S}_a^{(j)}. \tag{2.43}$$

2.4.2 Properties of the Heisenberg Model

Now we ask what symmetries and conserved quantities there are in the Heisenberg model. If we define the permutation operator $\hat{P}\hat{S}^{(j)} = \hat{S}^{(j+1)}$ we find $[\hat{H}, \hat{P}] = 0$, i.e. the model has site symmetry. Similarly defining the reflection operator $\hat{R}\hat{S}^{(j)} = \hat{S}^{(N+1-j)}$ we again find $[\hat{H}, \hat{R}] = 0$, and hence the model has mirror symmetry.

If J depends on direction, using Equation (2.43) we find

$$[\hat{H}, \hat{\mathbf{S}}] = -i \sum_{\langle i,j\rangle} (\hat{\mathbf{R}}^{(i)} \times \hat{\mathbf{S}}^{(j)} - \hat{\mathbf{S}}^{(i)} \times \hat{\mathbf{R}}^{(j)}), \tag{2.44}$$

where we have defined $\hat{R}_a^{(k)} = J_a \hat{S}_a^{(k)}$. Thus, for general values of $\mathbf{J} = (J_x, J_y, J_z)$ the total spin is not conserved. Similarly for a magnetic field we find

$$[\hat{H}_{\text{Zeeman}}, \hat{\mathbf{S}}] = i\mathbf{B} \times \hat{\mathbf{S}}. \tag{2.45}$$

Thus, for the z-component of the total spin to be conserved, for example, the necessary conditions are $B_x = B_y = 0$ and $J_x = J_y$. However, for uniform $\mathbf{J}$ and no magnetic field, as in Equation (2.38), the total spin along each axis is conserved.

2.4.3 Spin Waves

To gain further intuition on the Heisenberg model I will solve it in one dimension in the spin-wave regime. Here we have N sites, a uniform value of J [Equation (2.38)] and periodic boundary conditions. To solve the model, it is easier to write it in terms of raising and lowering operators [Equation (2.7)]

$$\hat{H} = -2J\sum_{i=1}^{N}\hat{S}_z^{(i)}\hat{S}_z^{(j)} - J\sum_{i=1}^{N}(\hat{S}_+^{(i)}\hat{S}_-^{(i+1)} + \hat{S}_-^{(i)}\hat{S}_+^{(i+1)}). \tag{2.46}$$

The ground-state of this hamiltonian occurs when all the spins are spin-up or spin-down, with ground-state energy, $E^{\mathrm{GS}} = -NJ/2$. Spin-waves are low energy excitations of the system, with a periodic oscillation in the spin orientation [25]. Consider the subset of states with spin $-(N-2)/2$, which consists of states which are identical to the ground state, except one spin has been flipped. A basis on this subspace is the set $\{|j\rangle\}$, where the state $|j\rangle$ consists of the j^{th} spin, pointing up and all the others down

$$|j\rangle \equiv |\downarrow ... \underbrace{\uparrow}_{j} .. \downarrow\rangle. \tag{2.47}$$

Diagonalizing the hamiltonian in this sector, we find the eigen-kets spanning the subspace are

$$|n\rangle = \frac{1}{\sqrt{N}}\sum_{j=1}^{N} e^{ik_n j}|j\rangle, \tag{2.48}$$

with energies

$$E_n = -2J(N/4 - 1) - 2J\cos k_n, \tag{2.49}$$

where $k_n = 2\pi n/N$ with $n \in 0, 1, ..., N-1$ due to the boundary conditions. The eigenstates are spin-waves, corresponding to a Fourier Transform of the basis states.

If we initially have a system in state $|j\rangle$, with the up-spin localized to one site, then, since this isn't an eigenstate, the state will become a superposition of the basis states. In the process the up-spin disperses throughout the system. At time t, the ket will be

$$|\psi(t)\rangle = \frac{1}{N}\sum_{j=1}^{N}\sum_{n=0}^{N-1}\exp(i\{2Jt\cos k_n + k_n(j-k)\})|j\rangle, \tag{2.50}$$

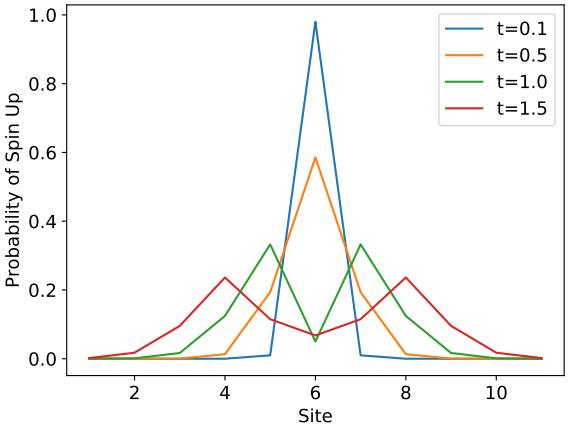


Figure 2.3: The probability of each site being spin up at different times for a 11 site Heisenberg model. Note how this shows that the waves propagate at a finite speed. Here the times are in units of J.

where we have removed an unmeasurable global phase. In Figure 2.3 we show the probability of site l at time t being spin up, $P_l = |\langle l|\psi(t)\rangle|^2$, for several times. Here we can see the wave nature, initially the up-spin is localized at one site, with it spreading out to other sites over time.

2.5 Diagonalization and Scaling

Solving either the Hubbard model or Heisenberg model in general is conceptually simple. One simply constructs the hamiltonian and diagonalizes it [3]. However, for both models the size of the basis scales exponentially with the number of sites making diagonalization impossible for large lattices.

For L sites the Hubbard hamiltonian is a $4^L \times 4^L$ matrix. This can be seen by realizing that each site can either be empty, filled with an up spin, filled with a down spin or doubly occupied. The hamiltonian of the Heisenberg model for N particles is $2^N \times 2^N$ since each site can only be spin up or down.

An important property of both models, is the fact that we can split the hamiltonian into sectors which do not interact as we have already seen (namely, the hamiltonian is block diagonal). In the Hubbard model, if we have L sites with $N_\uparrow$ spin up particles and $N_\downarrow$ spin down particles, then the size of the Hilbert

[3]For certain specialized cases the models can be solved analytically.

space of that sector will be

$$\Omega = \binom{L}{N_\uparrow}\binom{L}{N_\downarrow}. \tag{2.51}$$

For example the 2-site Hubbard model has a 16×16 hamiltonian matrix, but by splitting it into sectors the most we have to diagonalize is a 4×4 matrix. However, this really is not much of an improvement in the long run as Equation (2.51) still scales exponentially [4].

Several numerical methods have been designed to solve many-body problems for systems, whose size makes them not accessible by exact diagonalisation. If the interest lies with ground-state properties, the density matrix renormalisation group (DMRG) [27] and variational Monte Carlo [28] are valuable options, while at finite temperature continuous-time Monte Carlo [29] can be considered. This latter method usually suffer from the sign problem, [30] which affects its accuracy at low temperature [5]. In any case, although these schemes allow one the investigation of relatively large systems, their numerical overheads are still significant. This means that information concerning the thermodynamic limits of the various models and about the interplay between interaction and disorder remain difficult to access.

Any method that attempts to overcome this exponential scaling barrier will necessarily be an approximation. In the next two chapters we construct different approximations to overcome this scaling difficulty by using machine learning.

[4] For example, if the electrons with spin-up are half filled, $N_\uparrow = L/2$, and in the absence of any spin-down electrons, $N_\downarrow = 0$, then using Stirling's approximation the Hilbert space scales as

$$\Omega \approx \sqrt{\frac{2}{\pi L}} 2^L.$$

[5] The sign problem is the fact that since the fermionic wavefunction is highly oscillatory, it is very difficult to estimate reliably expectation values using Monte Carlo methods [30].

Chapter 3

Learning the Hubbard Model

> "The underlying physical laws necessary for the mathematical theory of a large part of physics and the whole of chemistry are thus completely known, and the difficulty is only that the exact application of these laws leads to equations much too complicated to be soluble. It therefore becomes desirable that approximate practical methods of applying quantum mechanics should be developed, which can lead to an explanation of the main features of complex atomic systems without too much computation. "
>
> - Paul Dirac [31]

Part of this chapter was published in Physical Review B, titled 'Machine learning density functional theory for the Hubbard model' [32].

3.1 Introduction

The Hubbard model, as introduced in Chapter 2, describes electrons interacting with each other on a lattice. It is useful for investigating systems of strongly interacting electrons, such as materials with f-electrons and Mott insulators [23]. As mentioned in the last section, the central problem with solving the Hubbard model is the fact that the Hilbert space scales exponentially with the number of sites and electrons, making the hamiltonian unfeasible to diagonalize for large systems. However, it turns out that all the information about the hamiltonian of the system is contained in the ground-state electron density, which scales linearly with the system size. The use of the density as the primary variable

instead of the wavefunction is known as Density Functional Theory (DFT) [26]. Thus, using DFT we can analyse systems of interacting electrons, while avoiding the exponential scaling of the wavefunction. While, normally DFT is combined with the Kohn-Sham scheme [33, 34] and several approximations, with the use of ML we can avoid these.

Applying ML to lattice models has generated much interest over the last few years, with ML being used to: identify phase transitions [35, 36] and solve models efficiently [37, 38]. While for learning the DFT mappings, ML has been used extensively for both continuous systems and lattice models [39–42].

In this chapter we consider two different versions of the one-dimensional Hubbard model, writing it in general as

$$\hat{H} = \hat{T} + \hat{U} + \sum_{i\sigma} \hat{n}_{i\sigma} v_i \tag{3.1}$$

where $\hat{T} = -t \sum_{i\sigma} (\hat{c}^\dagger_{i+1,\sigma} \hat{c}_{i\sigma} + \hat{c}^\dagger_{i\sigma} \hat{c}_{i+1,\sigma})$ is the kinetic energy with hopping parameter t and $\{v_i\}$ are the onsite energies. The first version is the standard Hubbard model seen in the last section with the Coulomb interaction

$$\hat{U} = U \sum_{i=1} \hat{n}_{i\uparrow} \hat{n}_{i\downarrow}, \tag{3.2}$$

where the repulsion strength is $U > 0$. The second version is the spinless extended Hubbard model. Here there is no spin - or only one spin - and thus we drop the σ subscript [1]. The interaction term is now

$$\hat{U} = V \sum_{i=1} \hat{n}_i \hat{n}_{i+1}, \tag{3.3}$$

where $V > 0$. We introduce this second model as its Hilbert space scales better, allowing us to probe larger lattices. For both models we use periodic boundary conditions.

This chapter begins with the fundamentals of lattice DFT, which are the Hohenberg-Kohn theorems. It then proceeds to our first set of results showing that ML can learn exact DFT maps for the Hubbard model, which satisfy the Hohenberg-Kohn theorems. Next we make a local approximation, which allows us to measure the locality of the maps and to use maps learned on small lattices to predict larger systems. Finally we show that finite temperature quantities can also be incorporated into the DFT scheme and, hence, we can compute thermal quantities at the thermodynamic limit.

[1] The spinless Hubbard model can also be viewed as the Hubbard model with spin in an infinite magnetic field.

3.2 Lattice Density Functional Theory

When applying DFT to lattice systems, our primary variable becomes the expected number of particles per site per spin

$$n_{i\sigma} = \langle\psi|\hat{n}_{i\sigma}|\psi\rangle, \tag{3.4}$$

which we refer to as the density or occupation.

Before delving into DFT, we note an important property of the Hubbard model. Assuming we have a fixed number of particles, N, if we shift the onsite energies by a constant, $v_i \to v_i + \alpha$, then the eigenstates of the hamiltonian stay the same and the eigenvalues shift by $N\alpha$.

The basis of DFT are the Hohenberg-Kohn (HK) theorems [34], which relevant to our case has been extended to lattice models [43]. We now proceed to state and prove them [2]:

1. The onsite energies $\{v_i\}$ are determined up to a constant by the ground-state density $\{n_{i\sigma}^{\mathrm{GS}}\}$. It follows from this that the hamiltonian is determined by the ground-state density and, hence, all the properties of the system are also determined.

2. The energy can be written in the form

$$E = F(\{n_{i\sigma}\}) + \sum_{i\sigma} n_{i\sigma} v_i, \tag{3.5}$$

   where $F(\{n_{i\sigma}\})$ is called the universal function and for a given set of onsite energies $\{v_i\}$ the exact ground-state density $\{n_{i\sigma}^{\mathrm{GS}}\}$ minimises Equation (3.5).

In order to prove the first HK theorem we show via contradiction that two different sets of onsites energies (that do not simply differ by a constant) must result in different ground-state densities. Consider two different sets of onsite energies $\{v_i\}$ and $\{\tilde{v}_i\}$ with ground-state wavefunctions $|\psi^{\mathrm{GS}}\rangle$ and $|\tilde{\psi}^{\mathrm{GS}}\rangle$ and ground-state energies E^{GS} and $\tilde{E}^{\mathrm{GS}}$ that both have the same ground-state density $\{n_{i\sigma}^{\mathrm{GS}}\}$. We assume the sets of onsite energies do not simply differ by a constant. Now we have

$$\begin{aligned}\tilde{E}^{\mathrm{GS}} &\equiv \langle\tilde{\psi}^{\mathrm{GS}}|\hat{T} + \hat{U} + \sum_{i\sigma}\hat{n}_{i\sigma}\tilde{v}_i|\tilde{\psi}^{\mathrm{GS}}\rangle < \langle\psi^{\mathrm{GS}}|\hat{T} + \hat{U} + \sum_{i\sigma}\hat{n}_{i\sigma}\tilde{v}_i|\psi^{\mathrm{GS}}\rangle \\ &= E^{\mathrm{GS}} + \sum_{i\sigma} n_{i\sigma}^{\mathrm{GS}}(\tilde{v}_i - v_i)\end{aligned} \tag{3.6}$$

[2] Note that I have tailored the theorems specifically for lattice DFT.

and similarly $E^{\rm GS} < \tilde{E}^{\rm GS} - \sum_{i\sigma} n_{i\sigma}^{\rm GS}(\tilde{v}_i - v_i)$. By adding the two together we arrive at a contradiction $E + \tilde{E} < E + \tilde{E}$. Hence we arrive at the conclusion that two different onsite energies cannot have the same ground-state density. It follows from this that the ground-state density determines the onsite energies up to an overall constant, and hence, also the hamiltonian and, hence, all properties of the system. Note that the theorem holds even if the ground-state is degenerate, as it just guarantees a ground-state density determined the properties of the system.

For the second theorem, we define an extremely important quantity, the universal function

$$F(\{n_{i\sigma}\}) = \langle\psi|\hat{T} + \hat{U}|\psi\rangle = \langle\psi|\hat{H}|\psi\rangle - \sum_{i\sigma} v_i n_{i\sigma}. \tag{3.7}$$

To prove the second theorem consider the the ground-state wavefunction $|\psi^{\rm GS}\rangle$ with energy $E^{\rm GS} = F(\{n_{i\sigma}^{\rm GS}\}) + \sum_{i\sigma} n_{i\sigma}^{\rm GS} v_i$. Since a different density will correspond to a different wavefunction - by the first theorem - its expected energy will be higher and hence theorem 2 follows.

Note that so far we have assumed that the ground-state is not degenerate. When this assumption does not hold, the HK theorems break-down, however one can still define a universal function with the properties given above, and map the density to the onsite energies, which for our purposes is all we require [44].

As an illustration of these results we consider the inhomogeneous two site Hubbard model with a single electron. The hamiltonian is

$$\hat{H} = v_1|1\rangle\langle 1| + v_2|2\rangle\langle 2| - t(|2\rangle\langle 1| + |1\rangle\langle 2|), \tag{3.8}$$

with energy eigenvalues

$$E = \bar{v} \pm \sqrt{\delta^2 + t^2}, \tag{3.9}$$

where $\delta = (v_1 - v_2)/2$ and $\bar{v} = (v_1 + v_2)/2$. By solving for the ground-state densities we find

$$n_1^{\rm GS} = \frac{1}{2}\left(1 - \frac{\delta}{\sqrt{t^2 + \delta^2}}\right), \tag{3.10}$$

with $n_2^{\rm GS} = 1 - n_1^{\rm GS}$. By solving for δ we find

$$v_1 - v_2 = t\frac{n_2^{\rm GS} - n_1^{\rm GS}}{\sqrt{n_1^{\rm GS} n_2^{\rm GS}}}, \tag{3.11}$$

thus we can determine the difference between the onsite energies from the ground-state density, which is the first HK theorem. This allows us to write the universal function as

$$F = E^{\rm GS} - \sum_i n_i^{\rm GS} v_i = -2t\sqrt{n_1^{\rm GS} n_2^{\rm GS}}. \tag{3.12}$$

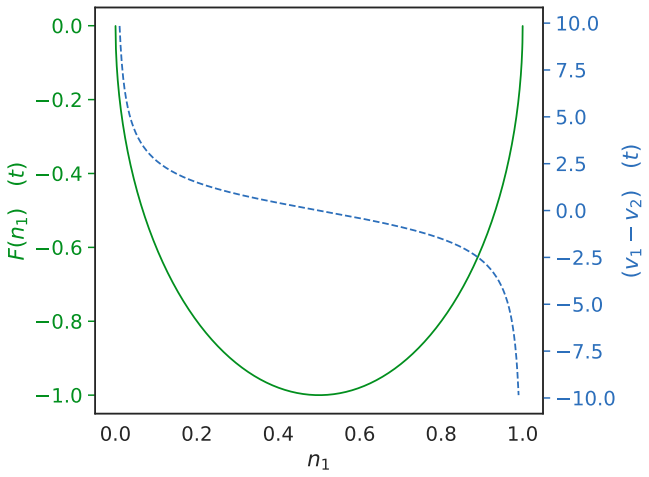


Figure 3.1: Here we show how the universal function (green) and the difference between the onsite energies (blue) vary as a function of the ground-state density of site 1, n_1, for the two site single electron Hubbard model. Both functions are in units of t.

Thus we can write the total ground-state energy as

$$E^{\mathrm{GS}} = 2t\sqrt{n_1^{\mathrm{GS}}(1-n_1^{\mathrm{GS}})} + v_1 n_1^{\mathrm{GS}} + v_2(1-n_1^{\mathrm{GS}}) \tag{3.13}$$

and if we differentiate with respect to n_1^{GS} and set to zero, we get back Equation (3.10), thus demonstrating the second HK theorem. Equations (3.11) and (3.12) are shown in Figure 3.1.

Finally, I make two further comments on DFT and the HK theorems. Firstly there is no HK theorem for excited states [45]. This is to say, one cannot, in general, map the excited state density to properties of the system. Secondly, as previously mentioned, usually DFT is used with the Kohn-Sham scheme [26, 33]. Here a non-interacting system, which reproduces the correct density is solved instead of the full interacting system. However, we can bypass the Kohn-Sham equations by using machine learning to directly learn the density functionals.

3.3 Machine Learning the Hohenberg-Kohn Theorems

We now turn to verifying the HK theorems with ML using the standard Hubbard model with spin, Equation (3.2). We create a dataset by choosing a system of

$L = 8$ sites arranged in a ring, with $N_\uparrow = N_\downarrow = 2$. Here the size of the Hilbert space is $\Omega = \binom{8}{2}^2 = 784$. Following the guidelines from Ref. [46] for creating the hamiltonian matrix, we randomly generate the onsite energies, build the hamiltonian and diagonalize it. We store only the ground-state density, $\{n_{i\sigma}^{\mathrm{GS}}\}$, and the corresponding ground-state energy, E^{GS}. For each sample in our dataset we compute the universal function, Equation (3.5). We repeat this procedure to build up a sufficiently large dataset for machine learning. Throughout the energy is measured in units of t.

Since the occupation sector we are investigating has an equal number of up electrons and down electrons, by symmetry of the hamiltonian, the ground-state expected occupations on each site must be the same for both spins $\hat{n}_{j\uparrow}^{\mathrm{GS}} = \hat{n}_{j\downarrow}^{\mathrm{GS}} = n_j^{\mathrm{GS}}/2$. Thus, we can take the expected number of particles on each site, n_j^{GS}, as our primary variable.

The random external potential is drawn according to a uniform distribution with $v_i \in [-W, W]$. In particular we have constructed several external potential distributions with W varying between 0.005t and 2.5t. Furthermore, in order to prevent the dataset from having large fluctuations in the universal function we neglect potentials yielding to universal functions $0.15t$ larger than that of the homogeneous case, which corresponds to the minimum universal function. This threshold is enforced to improve the performance of the model, since it ensures that at every energy scale there are a sufficient number of samples to avoid the model overfitting. The sizes of the datasets - training, test and validation - are given below.

Figure 3.2 shows the characteristics of the dataset. On the y-axis is the universal function and on the x-axis is the standard deviation of the density, given by

$$\sigma(\{n_i\}) = \sqrt{\frac{1}{L}\sum_{i=1}^{L}(n_i - N/L)^2}. \tag{3.14}$$

Note that for the homogeneous case, with all onsite energies equal, $n_i^{\mathrm{GS}} = N/L$, and the standard deviation of the density is zero.

In general in machine learning, the more data one has, the better the results are. When learning images, often the dataset is increased by adding copies of samples with added noise or rotated samples. Here we can increase the dataset size by including symmetries. In particular for any potential v_i the mirror-symmetric potential $v_i \to v_{L+1-i}$ yields a mirror-symmetric charge density with identical total energy. A similar situation applies to potentials obtained by translation, namely $v_i \to v_{i+1}$. Applying both symmetries means we can increase the dataset size by a factor of $2L$ without any further exact diago-

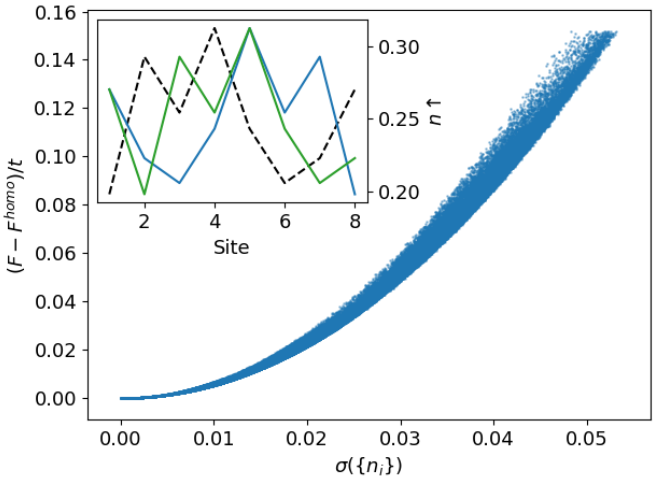


Figure 3.2: We plot the standard deviation of the occupations [Equation (3.14)] on the x axis and the universal function (minus the homogeneous universal function) on the y axis. Note that, as expected, the minimum universal function occurs for the homogeneous case. The insert shows an example of the symmetries added to the dataset. The original data-point is in black, the green line is generated from translational symmetry and the blue line from mirror symmetry.

nalization steps. Examples of the symmetries are shown in Figure 3.2.

In order to construct the ML models, the dataset is split into four mutually exclusive subsets. The training set, containing 52,500 samples, is used to train the model. The validation set, containing 26,250 samples, is employed to select the best ML model. The test set, also containing 26,250 samples, serves to estimate the generalization error of the model. Finally we set aside 100 configurations to test the minimization scheme for the demonstration of the second Hohenberg-Kohn theorem. These latter configurations are chosen uniformly over energy such that the entire range is explored. Throughout this section, we use convolutional neural networks with 8 kernel filters, followed by 2 fully connected layers with 128 units in each, follwed by an output layer. The convolutional neural networks were found to perform better than fully connected ones. The activation function was ReLU with the network implemented using Keras [47].

We begin with the first HK theorem and map the ground-state density to the onsite energies. As discussed before, we can only map up to a constant, thus we need to introduce a convention to make the mapping unique. We do this by subtracting the first site off the rest, $\tilde{v}_i = v_i - v_1$, and predicting the $L - 1$

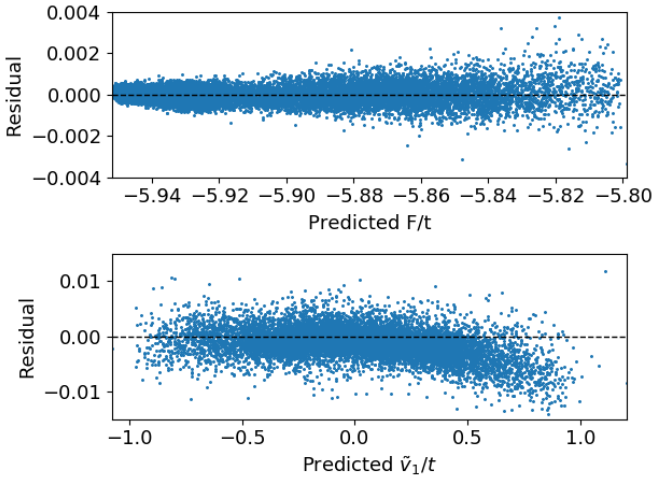


Figure 3.3: Here we show the results for the prediction of the universal function (top) and onsite energies (bottom). The residual is defined as the predicted quantity minus the exact value. For the universal function the mean absolute error is $0.0002t$ and for the onsite energies (subtracted by the original first site) it is $0.001t$ for the first site, with similar results for the other sites.

vector $\tilde{\mathbf{v}}$, where we have left out the first site as it is simply zero. The results are shown in Figure 3.3 with a mean absolute error of $0.001t$, thus demonstrating the mapping.

Now we move to the second theorem, showing that the universal function exists and has the variational property. Taking $\{n_i^{\mathrm{GS}}\}$ as an input again, we use a convolutional neural network and map to the universal function. The results are shown again in Figure 3.3 with a mean absolute error of $0.0002t$ demonstrating that the function does exist.

Finally we wish to show that by minimizing the total energy, using the machine-learned universal function we can get the correct ground-state density. To minimize the function we use gradient descent. We recall that in gradient descent we minimise a function, $f(\mathbf{x})$, starting from some initial choice $\mathbf{x}_0$, by applying the update $\mathbf{x}_{i+1} = \mathbf{x}_i - \epsilon f'(\mathbf{x}_i)$, where ϵ is the learning rate and must be specified. If we do not have an analytic equation for the first derivative, then we can estimate it using second order finite differences $\frac{\partial}{\partial x_i} f(\mathbf{x}) = \frac{f(\mathbf{x}+\alpha\mathbf{e}_i)-f(\mathbf{x}-\alpha\mathbf{e}_i)}{2\alpha} + O(\alpha^2)$, where e_i is the i^{th} unit vector. We keep updating $\mathbf{x}_i$ until we reach some convergence criterion, i.e. $|f(\mathbf{x}_{i+k}) - f(\mathbf{x}_i)| < \xi$ where ξ is some chosen tolerance. The gradient of the energy with respect to the total

particle density is simply

$$\frac{\partial}{\partial n_i} E = \frac{\partial}{\partial n_i} F(\{n_i\}) + 2v_i. \tag{3.15}$$

While minimizing we have two constraints that we must obey, namely the expected number of particles on a given site must be between 0 and 2, $0 \leq n_i \leq 2$, and we must have particle number conservation, $\sum_{i=1}^{L} n_i = N$. Recall, that we are using the total particle density here, since the up-spin and down-spin ground-state occupations are equal. To impose the first constraint, we will just halt the gradient descent algorithm, if this is violated. To impose the second, after every gradient descent step we normalize the density. We try two different minimization tests with the machine-learned universal function.

First, we start from some random occupations, consistent with the constraints, and use gradient descent to minimise the universal function. As this is equivalent to minimizing the total energy of the homogeneous case, all v_i the same, the minimum will correspond to the homogeneous density, with all n_i being the same. The results of this exercise are shown in Figure 3.4, demonstrating conclusively that we can get back the homogeneous density by minimizing the universal function.

Next we minimize the total energy given some set of onsite energies starting from the homogeneous density. The results are again shown in Figure 3.4, further verifying the variational property of the universal function. This concludes our numerical demonstration of both parts of the HK theorem.

3.4 The Semi-Local Density Approximation

While it is interesting that we can use machine learning to learn the HK maps, the work in the last section does not allows us to investigate large systems, that cannot be solved exactly due to the scaling of Hilbert space. In this section we confront this directly, by introducing a local approximation independent of the number of sites. Using this, we can transfer information from smaller lattices to larger lattices. For this section, since we want to make predictions on larger lattices we switch to the Spinless Hubbard model with a filling $\rho = N/L$ of 0.5. Note that now the total particle number per site, n_i, is between 0 and 1, in contrast to the last section. As we are working with lattices of different lengths, we divide all extensive quantities by the total number of sites, throughout, and represent them by a lowercase letter.

In order to predict larger lattices using information from smaller lattices we use a local density approximation (LDA) where we assume the universal

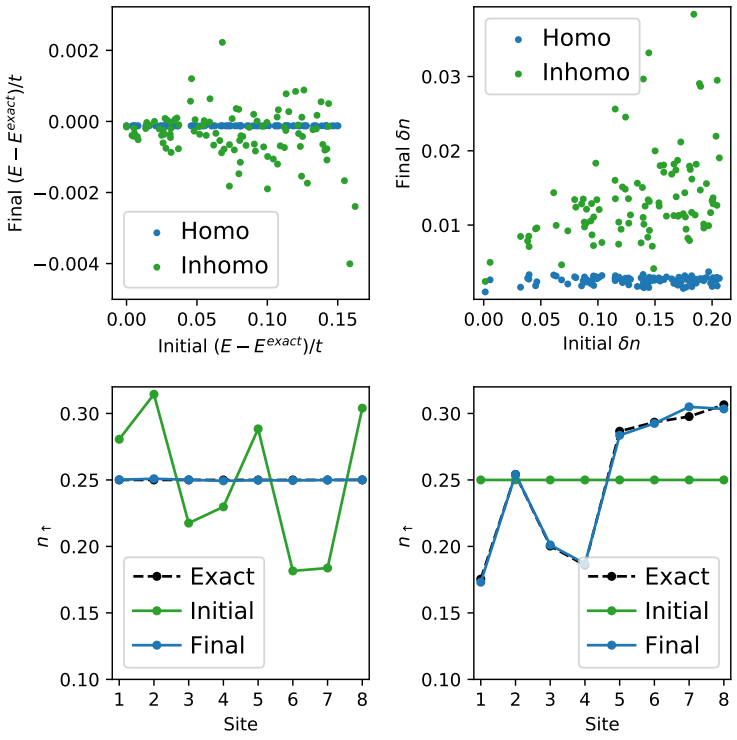


Figure 3.4: Here we shown the minimization results where we start from: some random occupation and minimize the universal function (Homo.) and the homogeneous occupation and minimize the energy (Inhomo). Top Left: We show how the accuracy of the predicted energy improves by plotting the initial predicted error against the converged predicted error. Top Right: Here we define the distance $\delta n = |\mathbf{n} - \mathbf{n}^{\text{eaxct}}|$ and we plot the value for the initial density and converged density. Bottom: Here we show the evolution of a random density with the initial density (green), converged density (blue) and exact density (black). The bottom left is for the Homo. case and the bottom right for the Inhomo. case.

function (divided by the total number of sites), can be approximated as

$$f_V^{\mathrm{LDA}}(\{n_i\}) = \frac{1}{L}\sum_{i=1}^{L} W_V(\bar{n}_{i,a}) \tag{3.16}$$

where $W_V(\bar{n}_{i,a})$ is associated with the i^{th} site. $W_V(\bar{n}_{i,a})$ in turn depends on the local site occupation,

$$\bar{n}_{i,a} = \{n_{i-a}, n_{i-a+1}, ..., n_i, ..., n_{i+a-1}, n_{i+a}\}\,, \tag{3.17}$$

meaning that the universal function associated to site i depends on the occupation at site i and on that at the first a sites around it (it depends on the site occupation at $2a+1$ sites). Thus a defines the locality of the universal function. This approximation allows us to write the universal function in a form that does not depend on the total number of sites, and hence it can be used to make predictions on lattices of different lengths. As well as being a local approximation, Equation 3.16 also encodes translational symmetry directly into the ML model. The $W_V(\bar{n}_{i,a})$ functions are modelled using neural networks implemented with PyTorch [48]. Throughout we use ReLU activation functions. Fig. 3.5 illustrates an example of how to construct the functional for a ring of 4 sites and a local density of range $a = 1$.

Recently, a conceptually similar way to construct ML models for extensive quantities has been brought forward by Mills and co-workers [49]. Our approach is similar in spirit, with the main difference being, that while we use the density as the primary variable, they opted for an image representation composed from the positions of the atomic nuclei.

We created a dataset for the Spinless Hubbard model, in a similar fashion as before, sampling the onsite energies from uniform distributions $[-\Delta, \Delta]$ with $\Delta = 2, 4, 6, 8$. For training we used a mixture of all of these values of Δ and for testing we used only $\Delta = 4$. We created nine datasets in total, for $L = 6, 10, 14$ and $V = 1, 2, 4$. In total for a given L and V there were $24,000$ points in the training set, $8,000$ in the validation set and 500 in the test set. Figure 3.6 shows the characteristics of this data.

We can use this approximation to measure how local the universal function is, by measuring the error between the predicted and exact universal function, against a. For $L = 6$ and $V = 1$ this is shown in Figure 3.7 with similar results obtained for larger lattices and other values of V. Here we see that the universal function is local, with there not being a substantial difference in error between $a = 1$ and $a = 2$.

Since the LDA representation is lattice size independent and the universal function is local, we can use small cheaply generated systems to train the ML

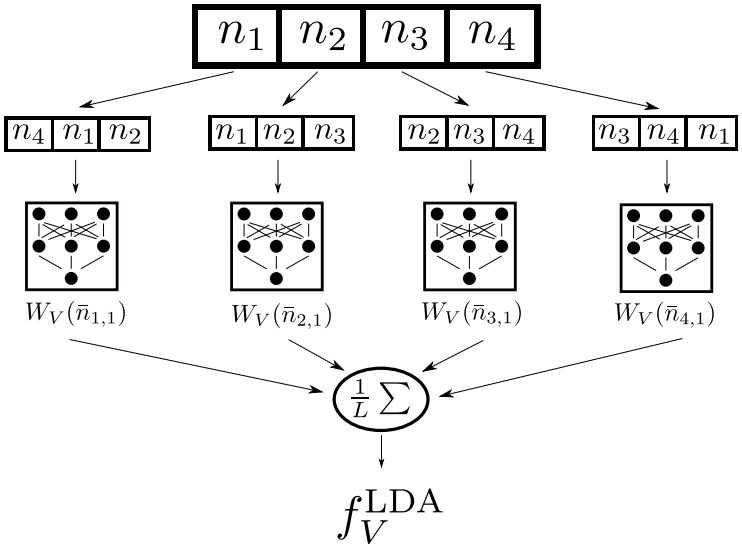


Figure 3.5: An illustration of how the local functional, $f_V^{\mathrm{LDA}}(\{n_i\})$, is constructed for a 4-site ring with $a = 1$. In this case four occupations, $\{n_1, n_2, n_3, n_4\}$, define uniquely the universal function. The LDA is written as the site average of four $W_V(\bar{n}_{i,a})$ contributions, constructed via a ML neural network, each one of them depending only on three site occupations.

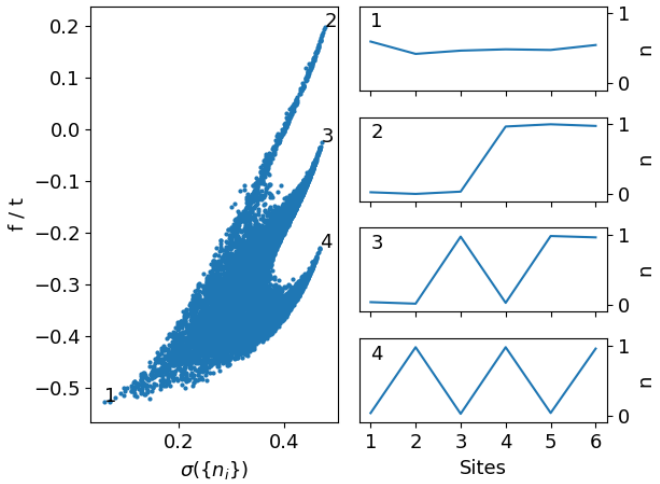


Figure 3.6: Here we show the structure of our dataset for $V = 1$ and $L = 6$. The left panel shows the exact per-site universal function, f_V, against the standard deviation of the occupations, $\sigma(\{n_i\}) = \sqrt{1/L\sum_{i=1}^{L}(n_i - N/L)^2}$, which is a measure of both the distance from the homogeneous case and disorder. The right panels shows the occupations which correspond to the numbers in the first plot, which are extrema of $\sigma(\{n_i\})$.

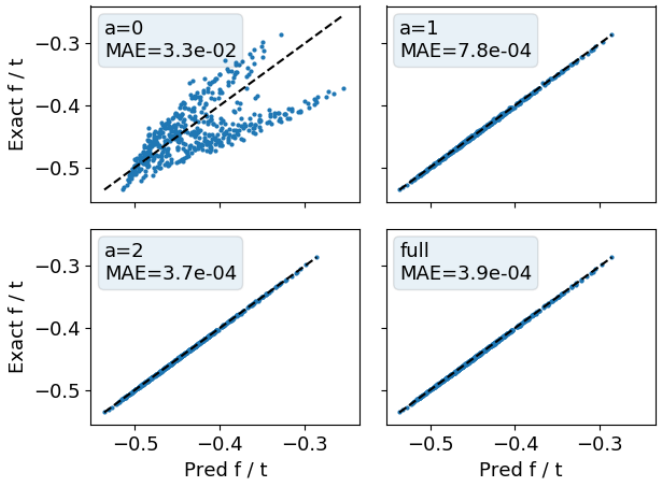


Figure 3.7: Here we plot the exact universal function against the predicted for $L = 6$ and $V = 1$. The box inside each plot details the a value and the mean absolute error, 'full' corresponds to all the sites being used in the input.

function and then predict quantities for larger systems. Before even testing the representation on larger lattices we can see that there is a problem with our representation. Consider the homogeneous case with $\{v_j = \text{const}\}$ and thus by site symmetry $\{n_j = N/L\}$. In this case all of the local site occupations, $\bar{n}_{i,a}$, will be the same, $\bar{n}_a = (N/L, ..., N/L)$ and thus, for any L,

$$f_V^{\text{LDA}}(\{n_i\}) = \frac{1}{L}\sum_{i=1}^{L} W_V(\bar{n}_{i,a}) = W_V(\bar{n}_a). \tag{3.18}$$

Hence, the representation will always present the homogeneous cases for differing lattices sizes as the same. The main benefit of the representation, its lattice size independence, is paradoxically the problem. This issue is not necessarily catastrophic, however. Figure 3.8 shows the results of transferring to larger systems for various values of V. From this analysis we can propose two solutions. Firstly, for largely disordered systems the energy scale is large compared to this finite size effect (here $\Delta = 4$). Secondly training on larger systems moves us further away from the effect and gives more accurate results, i.e. training on a lattice of ten sites gives much more accurate results than six sites for predicting the fourteen site case. The locality, a, shown in the figure, was chosen by minimizing the absolute error.

With this in mind, we can now proceed to rerun the minimization scheme seen in the last section, this time with the local functional trained on a smaller

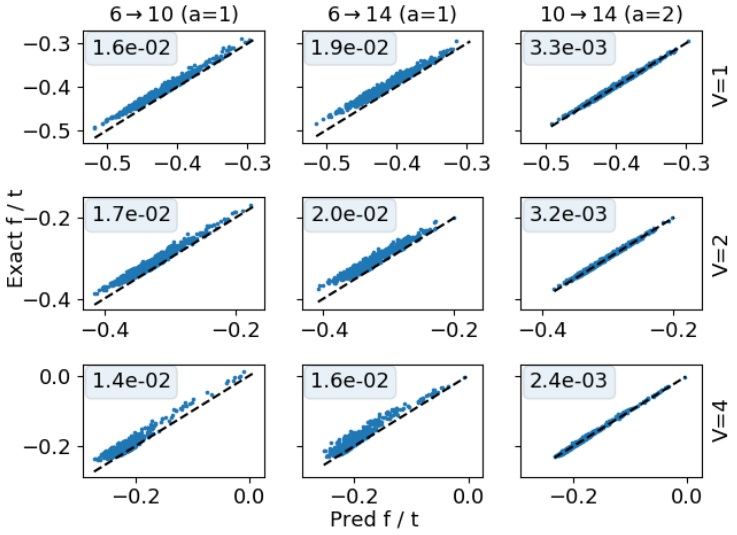


Figure 3.8: Here we plot the exact universal function against the predicted for $\Delta = 4$. In the first column the functions have been trained on $L = 6$ and are predicting $L = 10$. In the second column the functions have been trained on $L = 6$ and are predicting $L = 14$. Finally in the third column the functions have been trained on $L = 10$ and are predicting $L = 14$. The rows correspond to different values of V. The value of a is the same for each column. In a box in each plot is the mean absolute error.

lattice. Starting from the homogeneous site occupations, we update the density using gradient descent with momentum to minize the total energy. Here we used a learning rate of 0.002 and a momentum of 0.9. Using the trained function from $L = 10$ and $V = 1$, in Figure 3.9 we show the results of minimizing the energy for $L = 14$ given a set of onsite energies, for 100 samples. The figure shows that the converged site occupations and energies agree accurately with the exact value. This finding demonstrates, that transfer learning (training on small systems, to predict larger ones), can be used to approximate larger systems, which normally would be computationally intractable to solve.

3.5 Learning Thermodynamics

So far we have stayed in the original HK scheme, however, we are not just limited to ground-state properties, we can consider excited states as well. Here, an issue arises if we want to generalize from smaller systems to larger ones, since the size of the spectrum changes, and thus it cannot be learnt from smaller systems. Instead of excited states, we consider thermal properties, which are weighted averages of excited states. At the inverse temperature β, the equilibrium density matrix is $\hat{\rho} = e^{-\beta \hat{H}}/Z$ (see Appendix C) and the thermodynamical quantities of interest include, the expected energy

$$E = \mathrm{Tr}(\hat{H} e^{-\beta \hat{H}})/Z, \tag{3.19}$$

the entropy [3]

$$S = -\mathrm{Tr}(\hat{\rho} \log \hat{\rho}) = \log Z + E, \tag{3.20}$$

the free energy

$$H = E - TS = -T \log Z, \tag{3.21}$$

and finally the heat capacity

$$C = \frac{\partial}{\partial T} E. \tag{3.22}$$

Again we use the spinless extended Hubbard model and the same dataset as the previous section. To generated the thermal data we sampled the temperature, T, from the uniform distribution $[1, 2]$ with two temperatures used per data point.

In the DFT framework we can access thermal states via two different methods. Note that, however, since the ground-state density maps to the onsite energies up to an overall constant, we must again fix this constant by some convention, in order to be able to predict the thermodynamic energy from the

[3] Here we use the natural logarithm.

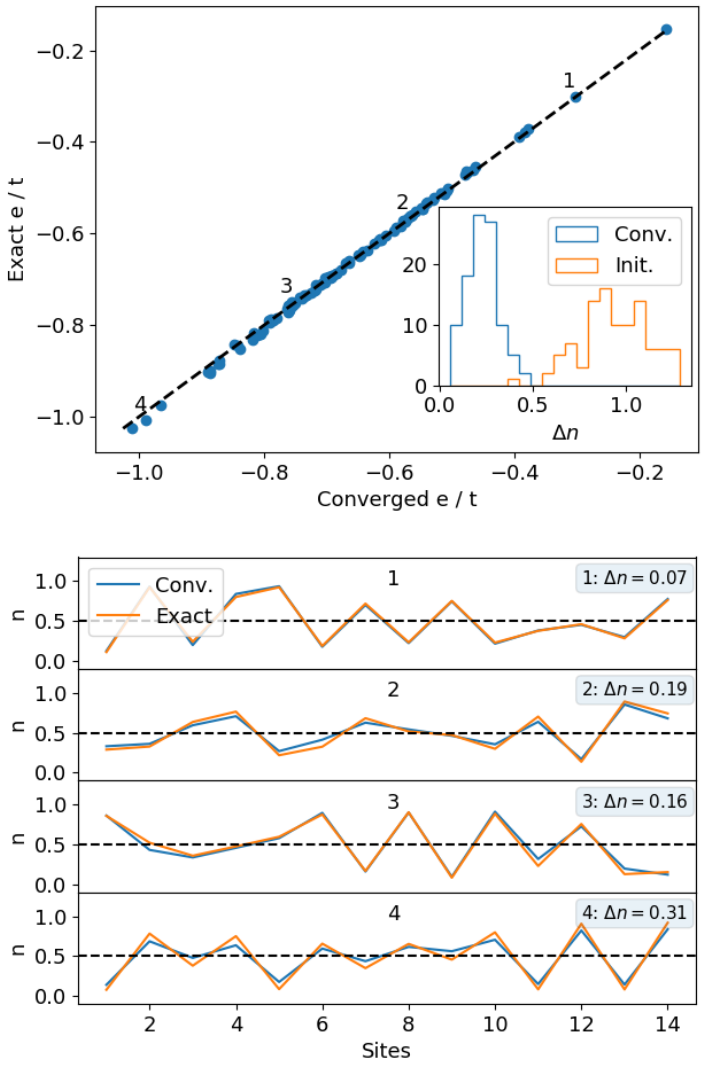


Figure 3.9: We show the results of the minimization of the total energy for $L = 14$, using the universal function trained on $L = 10$. Here we are using $V = 1$ and for the test set $\Delta = 4$. The top panel shows the converged energy against the exact energy. With the insert showing a histogram of the inital and converged euclidean distances from the exact occupations, Δn. On the lower panel is a selection of the converged occupations compared against the exact, with the numbers corresponding to their energy on the top panel.

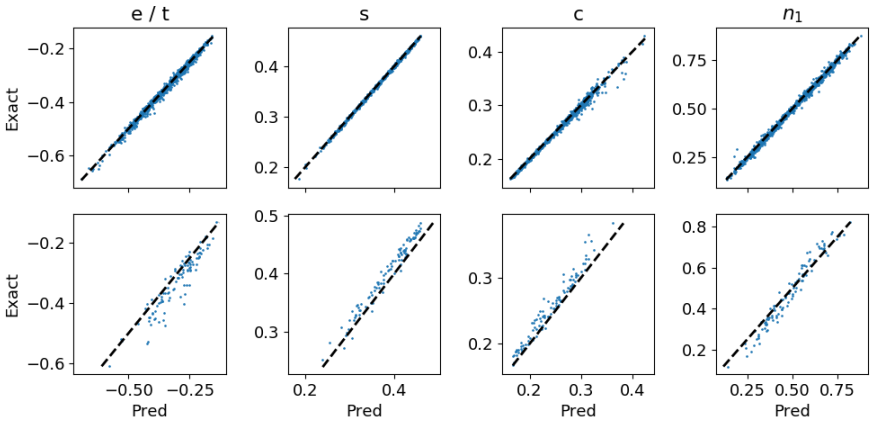


Figure 3.10: For the per site energy, entropy and heat capacity, and for the first occupation we plot the predicted and exact results, for a temperature range of $T \in [1, 2]$. The top and bottom panels correspond to different dataset, both use $V = 1$ and $\Delta = 4$. On the top panel the thermal functions were trained on $L = 10$ and use the full set of sites, and they are predicting $L = 10$. On the bottom panel the thermal functions were trained on $L = 10$ and use $a = 2$, and they are predicting $L = 14$ converged site densities as calculated by minimizing the total energy using the $L = 10$ local universal function.

density. Here we do this by forcing the onsite energies in a lattice to sum up to zero [4].

Firstly, recall that the HK theorems state that the ground-state density contains the same information as the hamiltonian. Thus, we can construct a map from the ground-state to the excited states and since thermal properties are just functions of the eigenstates we can predict them. Thus, here we map

$$\{n_{i,\sigma}^{GS}\} \to \langle o \rangle_T, \tag{3.23}$$

where $\langle o \rangle_T$ is some thermal quantity. Figure 3.10 shows the results of using this method, we see that by training the local thermodynamic functions of $L = 10$, we can accurately predict them for $L = 14$. Where, here we have used the converged densities (as calculated by minimizing the energy using the $L = 10$ local universal function), as the input for the local thermodynamic functions. Thus, here we have two sources of error, the error in the converged density and the error in extrapolating the thermal functions.

[4] This does not lose any generality for our method, since forcing the onsite energies to sum up to zero does not change the eigenstates, and only changes the energies by a constant.

A second method was introduced by Mermin [50]. It is an extension of DFT to finite temperatures, where instead of the ground-state density we take the expected density at a given temperature, $\langle n_{i,\sigma}\rangle$. Mermin showed[5] that there exists a functional

$$H = G[\{\langle n_{i,\sigma}\rangle\}] + \sum_i \sum_\sigma \langle n_{i,\sigma}\rangle v_i \tag{3.24}$$

where H is the free energy and is minimized for the equilibrium density $\langle n_{i,\sigma}\rangle_T$. Here $G[\{\langle n_{i,\sigma}\rangle\}]$ is the thermal universal function and we also define, $g[\{\langle n_{i,\sigma}\rangle\}] = G[\{\langle n_{i,\sigma}\rangle\}]/L$. Naturally, we approximate the thermal universal function with the local approximation.

Before minimizing the free energy, we investigate how temperature effects the locality of the thermal universal function. Figure 3.11 shows how the mean absolute error of the thermal universal function varies with temperature for $a = 0$ and 1. Here we are using $L = 6$, $V = 1$ and only data from $\Delta = 4$. Unsurprisingly, as the temperature increases, the function becomes more local. This is due to the fact that, in the infinite temperature limit, the equilibrium density matrix becomes

$$\hat{\rho} = \frac{1}{\Omega}\mathbb{1}, \tag{3.25}$$

where Ω is the dimension of the Hilbert space. Thus the infinite-temperature density is uniform, $n_i = N/L$, and hence $a = 0$ contains the same information as a non-zero. Note that $T = 0$ corresponds to the ground-state universal function.

Finally we minimize the free energy using the local thermal universal function. Here we again train on $L = 10$ and minimize $L = 14$ given a set of onsite energies. We use the same parameters for the gradient descent with momentum as before, and use $V = 1$ and $\Delta = 4$. Figure 3.12 shows the results, demonstrating that we can recover the thermal occupations and free energy with excellent accuracy. Recall here that the temperatures are randomly distributed between 1 and 2. Similarly to the original HK theorems, Mermin showed that the equilibrium density uniquely determines the hamiltonian (again up to constant). Thus we can in principle map the converged equilibrium density to any thermal property. Thus this method, for determining thermal quantities, is completely equivalent to the previous one.

[5]Here we modify the result for the canonical ensemble and lattice DFT.

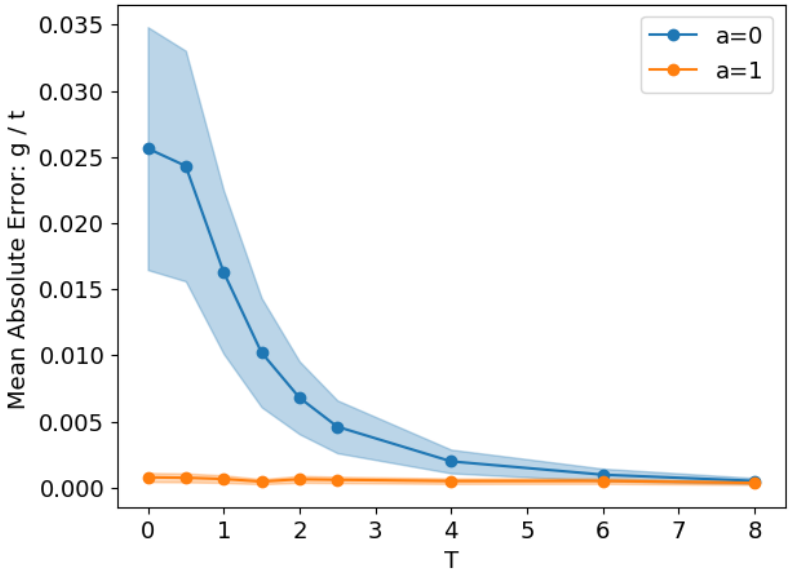


Figure 3.11: We show the mean absolute error of the thermal universal function against the temperature for $a = 0$ and 1. Here we use $L = 6$, $V = 1$ and $\Delta = 4$. The shaded regions are the standard deviation of the absolute error.

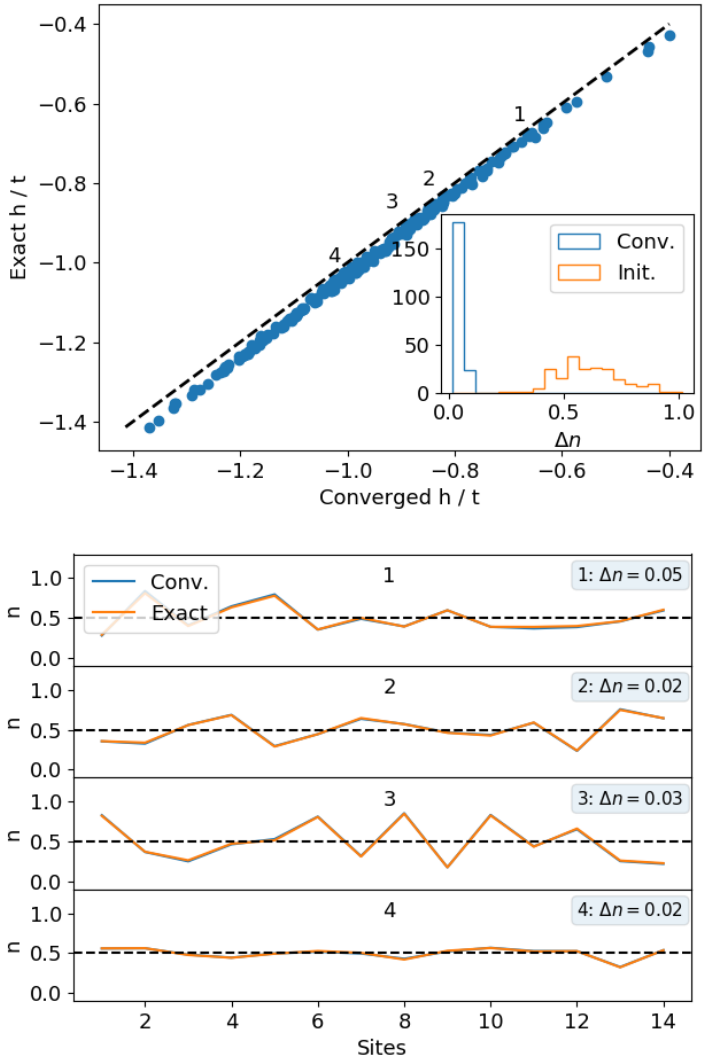


Figure 3.12: We show the results of the minimization of the free energy for $L = 14$, using the universal function trained on $L = 10$. Here we are using $V = 1$ and for the test set $\Delta = 4$. The top panel shows the converged free energy against the exact free energy. With the insert showing a histogram of the initial and converged euclidean distances from the exact occupations, Δn. On the lower panel is a selection of the converged occupations compared against the exact, with the numbers corresponding to their energy on the top panel.

3.6 Summary

In this chapter we have investigated the application of DFT to the Hubbard model using ML. While we have used specific models for our investigation our results are general and in theory applicable to any Hubbard like model with an external potential. We found that by using ML we could construct maps from the density to properties of the system, bypassing the need of calculating the wavefunction. Extending this, by using a local approximation, we showed that one could train the function on small lattices, which are accurate when transferred to larger systems. This allows one to avoid the exponential scaling of the Hilbert space and explore the Hubbard model in the large-L limit. As well, we can also predict thermal quantities, using the same scheme. However, one must bear in mind that our function is an approximation, and does fail in the homogeneous limit. But, it achieves impressive accuracy in the large disorder (onsite energy) regime.

Chapter 4

Learning Quantum Time Evolution

> "Nature can only shake so fast."
>
> - Signal Analysis: Time, Frequency, Scale and structure, Allen and Mills

4.1 Introduction

In the last chapter we saw that ML can be utilized to accurately predict the ground state and thermal properties of quantum systems. This leads naturally to the question of whether it can be used to predict the future evolution of such systems, which is what this chapter investigates.

There have been many applications of ML in quantum physics, beyond the DFT schemes discussed in the previous chapter. These include using ML to: efficiently measure entanglement [51], perform quantum state tomography [52], prepare quantum states [53] and infer quantum dynamics from measurements [54]. At the same time, impressive results have been achieved for the time evolution of classical systems [55, 56]. A recent approach to evolving quantum states is through representing the wavefunction with restricted Boltzmann machines and using the Dirac-Frenkel variational principle to evolve the state [57]. This method can be extend to dissipative systems as well [58].

Here we take a general approach: given quantum trajectories we use ML to learn the quantum mechanical time propagator, which given the current state can predict future states of the system. We investigate both Markovian and non-Markovian systems, with the non-Markovian systems requiring a memory

of previous states. Non-Markovian systems have inspired several investigations with ML, for example modelling with recurrent neural networks [59] and using ML to learn a Markovian embedding [60]. Distinguishing our work from these other approaches is: the universality of our approach, it can be used with any method to generate quantum evolutions and our investigations into the nature of the non-Markovian evolution.

This chapter begins with some background theory relating to qubits, entanglement and dynamics. Then we discuss the results in two sections. Firstly, we present the Markovian results and then a longer investigation into non-Markovian systems.

4.2 Background Theory

4.2.1 Subsystems and Entanglement

For a given quantum system, if we are able to define the wavefunction $|\psi\rangle$ then we call it a pure state. The density matrix for a pure state is given by $\hat{\rho} = |\psi\rangle\langle\psi|$. When we cannot define the wavefunction for a given system we have a mixed state and we must instead define an ensemble of states $\{|\psi_i\rangle\}$ each with probability p_i, giving the density matrix

$$\hat{\rho} = \sum_i p_i |\psi_i\rangle\langle\psi_i|. \tag{4.1}$$

The density matrix has three important properties: the probabilities add to one $\mathrm{Tr}[\hat{\rho}] = 1$, it is a positive operator with $\langle\theta|\hat{\rho}|\theta\rangle \geq 0$ for all $|\theta\rangle$ and it is Hermitian. Given a density matrix the expectation value of an operator $\hat{A}$ is $\mathrm{Tr}(\hat{A}\hat{\rho}) = \sum_i p_i \langle\psi_i|\hat{A}|\psi_i\rangle$.

A useful quantity is the Von-Neumann entropy, defined by [1]

$$S(\hat{\rho}) = -\mathrm{Tr}(\hat{\rho}\log_2\hat{\rho}) = -\sum_i p_i \log_2 p_i. \tag{4.3}$$

From the properties of the density matrix the entropy is non-negative and equal to zero only if the state is pure, $p_i = 1$. If the Hilbert space is of dimension 2^d then the state where every state in the ensemble is equally likely, $p_i = 1/2^d$, is the maximum entropy state [2] with entropy d. Thus the entropy measures

[1] If $\hat{A}$ is an operator with diagonalized form $\hat{A} = \sum_i a_i |i\rangle\langle i|$ then we define a function on $\hat{A}$, $f(\hat{A})$ by

$$f(\hat{A}) = \sum_i f(a_i)|i\rangle\langle i|. \tag{4.2}$$

[2] We can see this by maximizing with a constraint, we write the Lagrangian $L = -\sum_i \lambda_i \log_2 \lambda_i + \gamma(\sum_i \lambda_i - 1)$ and maximizing we see $\lambda_i^* = 1/2^d$ with the maximum entropy of d.

how mixed a state is [3]. Note that if the density matrix evolves under a unitary operation, $\hat{\rho} \to \hat{U}\hat{\rho}\hat{U}^\dagger$ then the entropy remains constant.

An important class of mixed states is given by the reduced density matrices. If two systems A and B together form a composite system AB, we can define the reduced density matrix for either (in this case A) as

$$\hat{\rho}_A = \mathrm{Tr}_B[\hat{\rho}] \tag{4.4}$$

where the partial trace is defined by

$$\mathrm{Tr}_B[|A_1, B_1\rangle\langle A_2 B_2|] = |A_1\rangle\langle A_2|\mathrm{Tr}_B[|B_1\rangle\langle B_2|] = |A_1\rangle\langle A_2|\langle B_2|B_1\rangle. \tag{4.5}$$

To see why the reduced density matrix is useful note that for the operator $\hat{O} = \hat{O}_A \otimes \mathbb{1}$ (where $\hat{O}_A$ acts only on subsystem A) we obtain $\mathrm{Tr}[\hat{O}_A\hat{\rho}_A] = \mathrm{Tr}[\hat{O}\hat{\rho}]$. Thus $\hat{\rho}_A$ contains all the information required to compute any operator acting only on the subsystem A.

One of the strangest aspects of quantum mechanics is the phenomena of quantum entanglement, which allows for non-local instantaneous interactions. Consider a system in a pure state composed of two subsystems A and B. If the wavefunction of the whole system $|\psi\rangle$ cannot be written in the form $|\psi\rangle = |A\rangle|B\rangle$, i.e. the state is not separable, then the state is entangled. Consider for example the Bell state $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, where the two subsystems are two-level systems, with the levels denoted by 0 and 1. This state is entangled, after measuring one of the subsystems and finding it in some state we immediately know the other subsystem is in the same state.

Suppose we have a wavefunction describing a system composed of two subsystems A and B with Hilbert space dimensions Ω_A and Ω_B, respectively. Then we can write a general state as

$$|\psi\rangle = \sum_{i=0}^{\Omega_A - 1} \sum_{j=0}^{\Omega_B - 1} \psi_{ij} |i\rangle_A |j\rangle_B \tag{4.6}$$

where the first ket refers to subsystem A and the second to B. If we treat ψ_{ij} as a $\Omega_A \times \Omega_B$ matrix $\psi_{i,j}$, then we can use singular value decomposition (SVD)

[3] Another way of measuring how mixed a state is is $\mathrm{Tr}(\hat{\rho}^2) = \sum_x \lambda_x^2 \leq 1$ with equality being found only if ρ describes a pure state.

[4] to write $\psi_{i,j} = \sum_{k=0}^{r-1} U_{i,k} D_{k,k} V^*_{j,k}$ where r is equal to $\min(\Omega_A, \Omega_B)$ and thus

$$\begin{aligned} |\psi\rangle &= \sum_{k=0}^{r-1} D_{k,k} \left(\sum_{i=0}^{\Omega_A - 1} U_{i,k} |i\rangle_A \right) \left(\sum_{j=0}^{\Omega_B - 1} V^*_{j,k} |j\rangle_B \right) \\ &= \sum_k \lambda_k |k\rangle_A |k\rangle_B \end{aligned} \tag{4.8}$$

where $\lambda_k = D_{k,k}$. The new basis is orthogonal since $\langle k'|_A |k\rangle_A = (U^\dagger U)_{k'k} = \delta_{kk'}$ with similar result for B. Writing the state in this form is known as Schimdt Decomposition with the number of non-zero values of λ_k known as the Schimdt number. From this it follows that $\hat{\rho}_A = \sum_k \lambda_k^2 |k\rangle_A \langle k|_A$ and $\hat{\rho}_B = \sum_k \lambda_k^2 |k\rangle_B \langle k|_B$ and thus the entropy of the two reduced matrices are equal, $S(\hat{\rho}_A) = S(\hat{\rho}_B) = -\sum_k \lambda_k^2 \log_2 \lambda_k^2$. Thus, if we have a pure state composed of two subsystems, the entropy of the reduced density matrix of either subsystem is a measure of the entanglement between the two subsystems. If the states are not entangled then the reduced matrices will be pure with a Schmidt number of 1 and the entropy will be zero. Schmidt decomposition gives us an interesting interpretation of entanglement. If we write the wavefunction of a composite system as $|\psi\rangle = \sum_A \sum_B \psi_{A,B} |A\rangle |B\rangle$ and try to compress the matrix $\psi_{A,B}$ with SVD then the more entanglement between A and B the less accurate the compression will be.

Suppose now we have a system composed of three subsystems A,B and C, if we let $\hat{\rho}^{\mathrm{AB}} = \mathrm{Tr}_C(\hat{\rho})$ and define $S(A,B) = -\mathrm{Tr}(\hat{\rho}^{\mathrm{AB}} \log_2 \hat{\rho}^{\mathrm{AB}})$ then we have the inequality [20]

$$|S(A) - S(B)| \leq S(A,B) \leq S(A) + S(B). \tag{4.9}$$

This can be generalized further, if $A_1, ..., A_N$ are the constituents of some system A then $S(A_1, ..., A_N) \leq \sum_{i=1}^N S(A_i)$, thus if all the subsystems are in pure states, $S(A_i) = 0$, then there can be no higher levels of entanglement in the system.

4.2.2 Qubits

In quantum mechanics a two-level system is called a qubit (quantum bit). It is convention to label one level 1 and the other 0. The physical interpretation of the levels depends on the background model. For the spin-1/2 Heisenberg model

[4] Given any $n \times m$ matrix A, SVD decomposes A as

$$A = USV^\dagger. \tag{4.7}$$

Here U and V are unitary matrices with dimensions $n \times r$ and $r \times m$ respectively where $r = \min(n, m)$ and S is a diagonal $r \times r$ matrix.

0 denotes an up-spin and 1 denotes a down-spin, say. Thus the wavefunction for a generic qubit is $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. The density matrix for either a mixed or pure state qubit can be written as

$$\hat{\rho} = \frac{1}{2}(\mathbb{1} + r_1\hat{\sigma}_x + r_2\hat{\sigma}_y + r_3\hat{\sigma}_z), \tag{4.10}$$

where $\mathbf{r}$ is called the Bloch vector [5] and $\hat{\sigma}_i$ are the Pauli matrices, see Equation (2.5). The Bloch vector fully specifies the state and has several convenient properties. Firstly the eigenvalues of the density matrix are $\lambda_\pm = \frac{1}{2}(1 \pm |\mathbf{r}|)$. Since they must be positive we conclude that $|\mathbf{r}| \leq 1$, i.e. the Bloch vector lies inside the unit sphere. The entropy of a qubit is a function of the Bloch vector magnitude

$$S(\hat{\rho}) = \left(\frac{1+|\mathbf{r}|}{2}\right)\log_2\left(\frac{1+|\mathbf{r}|}{2}\right) + \left(\frac{1-|\mathbf{r}|}{2}\right)\log_2\left(\frac{1-|\mathbf{r}|}{2}\right) \tag{4.11}$$

with the maximum entropy qubit state having a Bloch vector at the origin, $\mathbf{r} = (0,0,0)$, and pure states lying on the surface of the sphere. Finally we can write the expectation value of a Pauli matrix as

$$\langle\hat{\sigma}_i\rangle = \mathrm{Tr}(\hat{\sigma}_i\hat{\rho}) = r_i. \tag{4.12}$$

Often we want to be able to measure how different two density matrices, $\hat{\rho}$ and $\hat{\tau}$, are. The metric used in this work is the trace distance defined as

$$D(\hat{\rho},\hat{\tau}) = \frac{1}{2}\mathrm{Tr}|\hat{\rho}-\hat{\tau}|. \tag{4.13}$$

It is straightforward to see that $0 \leq D(\hat{\rho},\hat{\tau}) \leq 1$ with $D(\hat{\rho},\hat{\tau}) = 1$ when $\hat{\rho}$ and $\hat{\tau}$ have orthogonal support [6]. When both density matrices describe qubit systems with Bloch vectors $\mathbf{r}_\rho$ and $\mathbf{r}_\tau$ the trace distance reduces to

$$D(\hat{\rho},\hat{\tau}) = \frac{1}{2}|\mathbf{r}_\rho - \mathbf{r}_\tau| \tag{4.14}$$

since $\rho - \tau = \frac{1}{2}(\mathbf{r}_\rho - \mathbf{r}_\tau)\cdot\sigma$ has eigenvalues $\pm\frac{1}{2}|\mathbf{r}_\rho - \mathbf{r}_\tau|$. If two density matrices evolve via the same unitary operator the trace distance between them is preserved.

4.3 Markovian Maps

A system is Markovian if its future evolution only depends on the current state. Thus if we have a Markovian system in state x_n at time $t = n\Delta$, where Δ is the

[5] Given a qubit density matrix ρ the Bloch vector is $\mathbf{r} = (2\mathrm{Re}(\rho_{10}), 2\mathrm{Im}(\rho_{10}), 2\rho_{00} - 1)$.

[6] For a Hermitian operator the support is the vector space spanned by the eigenvectors of the operators with non-zero eigenvalues. If $\hat{A} = \sum_i a_i|a_i\rangle\langle a_i|$ and $\hat{B} = \sum_i b_i|b_i\rangle\langle b_i|$ then if they have orthogonal support $\hat{A}\hat{B} = 0$.

fundamental time, then there exists a map

$$x_{n+1} = f(x_n). \tag{4.15}$$

By repeatedly applying the propagator function we can access any future time step, $x_{n+m} = f^{(m)}(x_n)$, where $f^{(m)}$ denotes the function f being applied m times. This is known as autoregression. Note that given x_n we can only access past time states if the map is invertible, i.e. it never maps two different states to the same state.

As the wavefunction and density matrix capture all the information of a system they are both Markovian and we proceed to describe both.

4.3.1 Wavefunction

The time evolution of the wavefunction of a closed system is governed by the Schrödinger equation

$$i\frac{d}{dt}|\psi(t)\rangle = \hat{H}|\psi(t)\rangle, \tag{4.16}$$

where $\hat{H}$ is the hamiltonian operator. This has the generic solution

$$|\psi(t)\rangle = e^{-i\hat{H}t}|\psi(0)\rangle = \sum_{jk} U_{jk}(t)\,\langle k|\psi(0)\rangle\,|j\rangle \tag{4.17}$$

where $U_{jk}(t) = \langle j|e^{-i\hat{H}t}|k\rangle$ and $|j\rangle$ is some basis. If the hamiltonian has spectral decomposition $\hat{H} = \sum_n \epsilon_n|\epsilon_n\rangle\langle\epsilon_n|$ then $U_{jk}(t) = \sum_n e^{-i\epsilon_n t}\langle j|\epsilon_n\rangle\langle\epsilon_n|k\rangle$ and we see that energy eigenvalues define the frequencies of the evolution.

For use with ML, we write the wavefunction as a real 2Ω-vector

$$x_i = \begin{cases} \mathrm{Re}(\psi_i) & i \leq \Omega \\ \mathrm{Im}(\psi_{i-\Omega}) & i > \Omega \end{cases} \tag{4.18}$$

where $\psi_i = \langle i|\psi\rangle$ and Ω is the dimension of the Hilbert space. Here $\mathbf{x}$ is also a unit vector since $\langle\psi|\psi\rangle = |\mathbf{x}|^2 = 1$. If we define the $2\Omega \times 2\Omega$ matrix

$$M(t) = \begin{pmatrix} \mathrm{Re}(U(t)) & -\mathrm{Im}(U(t)) \\ \mathrm{Im}(U(t)) & \mathrm{Re}(U(t)) \end{pmatrix}, \tag{4.19}$$

where for example $\mathrm{Re}(U(t))$ is the real part of the U matrix at time t, then

$$\mathbf{x}(t) = M(t)\mathbf{x}(0). \tag{4.20}$$

Given a set of wavefunctions at an initial time and another set of the same wavefunction evolved after some time we can learn the mapping between the two by using linear regression. If we have p samples in the set, then defining X

as the $p \times 2\Omega$ matrix whose rows are the initial states and Y as the $p \times 2\Omega$ matrix whose rows are the evolved states, we can compute M exactly using multi-linear regression

$$M = (X^T X)^{-1} X^T Y. \tag{4.21}$$

This is a generalization of the linear regression model presented in Section 1.2.1, to vector targets. The only requirement here is that the matrix $X^T X$ is non-singular, which is the case if we have $p \geq 2\Omega$ different samples.

4.3.2 Density Matrix

Applying the Schrödinger equation to an ensemble of density matrices we obtain Von-Neumann's equation

$$\frac{d}{dt}\hat{\rho}(t) = i[\hat{\rho}(t), \hat{H}]. \tag{4.22}$$

In general, for the density matrix we must use a similar encoding to the one for the wavefunction and again we can solve for the propagator using linear regression. However, the dimension of the representation will be Ω^2 now and, hence, we would need $p > \Omega^2$ samples. For the qubit case we can use the Bloch vector, $\mathbf{r}$, as representation.

Under unitary evolution, as the entropy is constant the Bloch vector simply rotates about the origin with

$$\mathbf{r}(t) = M(t)\mathbf{r}(0), \tag{4.23}$$

where $M(t)$ is an orthonormal matrix. A non-unitary transformation can thus alter the entropy, although it must keep the properties of the density matrix, namely positivity and $\mathrm{Tr}(\hat{\rho}) = 1$. For a general operation on a density matrix, the mapping between the initial density matrix and the density matrix at time t takes the form [20]

$$\hat{\rho}(t) = \sum_k \hat{E}_k(t)\rho(0)\hat{E}_k^\dagger(t), \tag{4.24}$$

where the Krauss operators, $\hat{E}_k(t)$, satisfy the condition $\sum_k \hat{E}_k^\dagger(t)\hat{E}_k(t) = \mathbb{1}$. For Markovian dynamics [7] it is easy to see that this leads to the following Block vector evolution

$$\mathbf{r}(t) = \mathbf{a}(t) + M(t)\mathbf{r}(0) \tag{4.25}$$

for some vector $\mathbf{a}(t)$ and matrix $M(t)$. By defining $\tilde{\mathbf{r}} = (1, r_1, r_2, r_3)$ we can rewrite this equation in the form $\tilde{\mathbf{r}}(t) = P(t)\tilde{\mathbf{r}}(0)$, where $P(t)$ is a matrix and hence apply Equation (4.21) to learn the propagator. Figure 4.1 shows the

[7] For non-Markovian dynamics the Kraus operators, $\hat{E}_k$, could depend on the initial state $\hat{\rho}(0)$.

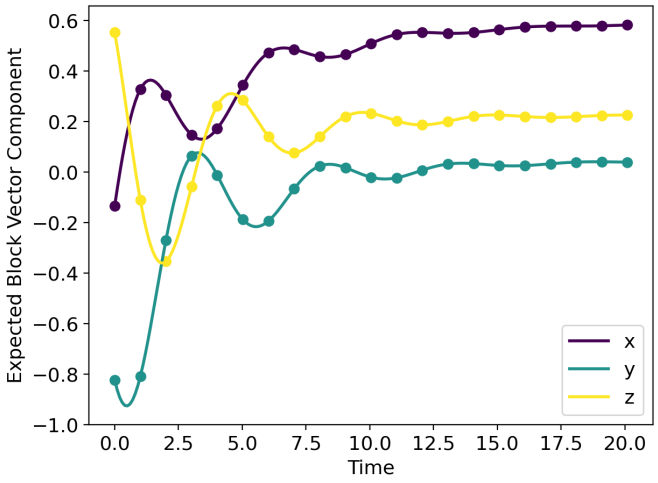


Figure 4.1: Here we show how the ML propagator (dots) compares to the exact propagator (solid lines) for the non-unitary Linblad data, generated by solving Equation (4.26). After a time of 20 the mean trace distance error between the ML propagator and the exact one was still within floating point error.

results of learning non-unitary qubit trajectories generated using the Lindblad master equation [61]

$$\frac{d\hat{\rho}}{dt} = -i[\hat{H}, \hat{\rho}] + 2\hat{L}\hat{\rho}\hat{L}^{\dagger} - \{\hat{L}^{\dagger}\hat{L}, \hat{\rho}\} \tag{4.26}$$

choosing $\hat{H} = 0.7\hat{\sigma}_x + 0.8\hat{\sigma}_y + 0.2\hat{\sigma}_z$ and $\hat{L} = \sqrt{0.1}\hat{\sigma}_x$, which although non-unitary is Markovian [8]. Here only four samples were needed to learn the propagator to floating point accuracy.

4.4 Non-Markovian Maps

Non-Markovian dynamics occurs when the state at a particular time is not sufficient to describe the future dynamics, but a collection of past states are. Thus now we have the map

$$x_{n+1} = f(x_n, x_{n-1}, ..., x_{n-h+1}), \tag{4.27}$$

where h is the Markov order or the history parameter. Markovian dynamics corresponds to the case when the history parameter is 1. Note that even with the history we can still perform autoregression, as before. For example, if $\tilde{x}_{n+1}$

[8] The form of the equation ensures $\frac{d}{dt}\mathrm{Tr}[\hat{\rho}] = \mathrm{Tr}[\frac{d}{dt}\hat{\rho}] = 0$ and thus $\mathrm{Tr}[\hat{\rho}] = 1$ throughout.

is the predicted state at time step $n+1$, we can compute the state at time state $n+2$ by $\tilde{x}_{n+2} = f(\tilde{x}_{n+1}, x_n, ..., x_{n-h+2})$. Continuing in this fashion, we can access any time step in the future.

In order to see how non-Markovian dynamics can come about, consider a system comprised of two parts: A and B. Defining the hamiltonian $\hat{H} = \hat{H}_A + \hat{H}_B + \hat{H}_{AB}$ and the reduced density matrix for sub-system A, $\hat{\rho}_A(t) = \mathrm{Tr}_B[\hat{\rho}(t)]$, we get

$$\frac{d}{dt}\hat{\rho}_A(t) = i\mathrm{Tr}_B([\hat{\rho}(t), \hat{H}]) = i[\hat{\rho}_A(t), \hat{H}_A] + i\mathrm{Tr}_B([\hat{\rho}(t), \hat{H}_{AB}]), \tag{4.28}$$

with similar result for the subsystem B. Thus, if the coupling between the subsystems, $\hat{H}_{AB}$, is non-zero then the reduced matrices will evolve non-unitarily.

Throughout this section for the non-Markovian data we consider a ring of qubits interacting via the Heisenberg model (see section 2.4) with hamiltonian

$$\hat{H} = -\frac{J}{2}\sum_{j=1}^{N} \hat{\boldsymbol{\sigma}}^{(j)} \cdot \hat{\boldsymbol{\sigma}}^{(j+1)}, \tag{4.29}$$

where N is the number of qubits and J, the strength of interaction, defines the time scale. Here we are using periodic boundary conditions with $\hat{\boldsymbol{\sigma}}^{(N+1)} = \hat{\boldsymbol{\sigma}}^{(1)}$. Trajectories of the wavefunction were generated by integrating the Schrödinger equation using the Quantum Toolbox in Python (QuTiP) package [62].

For $N = 2, 3, 4$ this hamiltonian gives rise to a periodic behaviour in time with periods $T_2 = \pi/2$, $T_3 = 2\pi/3$ and $T_4 = \pi$ with all times in units of $1/J$. This can be seen in Figure 4.2, where we plot the fidelity between the wavefunctions as they evolve over time and the initial state for a range of random initial states [9]. We also note that for all N, after a time of approximately $1/J$, the system is maximally far from the initial state, thus defining a decorrelation time [10].

A motivation for studying non-Markovian dynamics is the fact that the dimension of the wavefunction is 2^N, and hence a compressed representation would be highly desirable. However, using a compressed representation means that we lose much of the information about the state and hence the dynamics in terms of this representation are no longer Markovian. In order to introduce the representation we use, let $\mathbf{r}_i^n$ be the Bloch vector for qubit i at time $t = n\Delta$ where

[9] If $|\psi(t)\rangle$ is the state at time t then the fidelity between it and the initial state is given by

$$F(t) = |\langle\psi(0)|\psi(t)\rangle|^2. \tag{4.30}$$

[10] Although for the periodic cases ($N = 2, 3, 4$), it is perhaps better named as a quasi-decorrelation time, as the system always returns to the initial state.

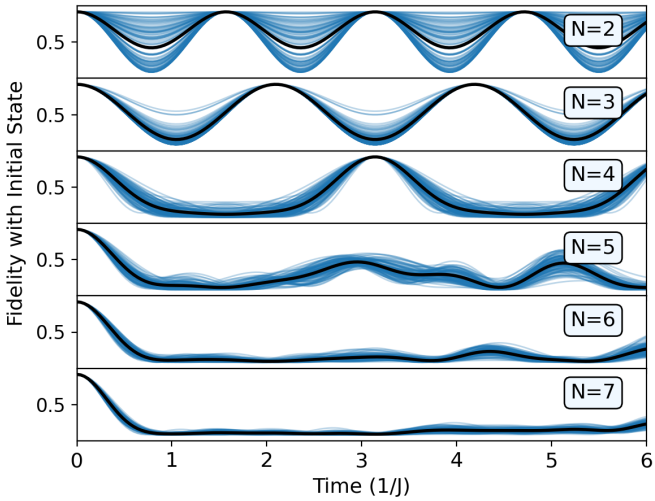


Figure 4.2: The fidelity of the wavefunction at a given time with the initial state for a range of states initialized randomly. Here by fidelity we mean the squared overlap, i.e. for two states ϕ and ψ we compute $|\langle\phi|\psi\rangle|^2$. The black lines are the means. N=2,3,4 are found to be periodic.

Δ is the fundamental time. We define

$$R^n = (\mathbf{r}_1^n, \mathbf{r}_2^n, ..., \mathbf{r}_N^n), \tag{4.31}$$

i.e. the collection of all Bloch vectors at a given time. We consider two types of mappings (in the underbraces we show the dimensions):
all (a)

$$\underbrace{(R^n, R^{n-1}, ..., R^{n-h+1})}_{3\times N\times h} \rightarrow \underbrace{R^{n+m}}_{3\times N} \tag{4.32}$$

and single (s)

$$\underbrace{(\mathbf{r}^n, \mathbf{r}^{n-1}, ..., \mathbf{r}^{n-h+1})}_{3\times h} \rightarrow \underbrace{\mathbf{r}^{n+m}}_{3} \tag{4.33}$$

where the history, h, determines how far back to go and m is the distance into the future to predict. Unlike the case for the wavefunction, both mappings grow linearly with the number of qubits. Note that increasing the history, h, or using all instead of single gives the network more information and thus should not increase the error assuming no training issues. Both mappings are shown in Figure 4.3.

While Recurrent Neural Networks have been used previously in the literature to model quantum evolution [59, 63] here we opt for a fully connected neural

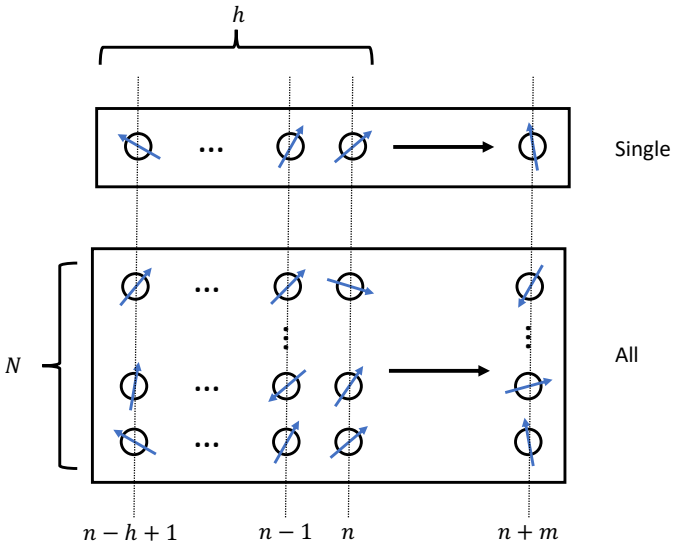


Figure 4.3: Here the two types of non-Markovian mappings are shown. The blue arrows symbolise the Block vectors of the individual qubits, while the dashed lines are the time steps. N is the number of qubits, h is the history and m is the future time step being mapped to. In the 'single' case, the history of a single qubit is used to predict its future dynamics, while in the 'all' case, the history of all the qubits in the ring, is used to predict the future dynamics of the entire ring.

network, since we only deal with inputs of a fixed size. In particular we use a two-layer network with 64 nodes in each hidden layer. We use the Exponential Linear Unit (ELU) activation function which we found to perform best. To learn the parameters in the network we minimize the trace distance between the predicted and true Bloch vectors. The networks were implemented using PyTorch [48].

We consider states initialized in two different ways: randomly and in a product state. For the random states the wavefunction is initialized as the superposition

$$|\psi(0)\rangle = \sum_{i_1=0,1} \sum_{i_2=0,1} ... \sum_{i_N=0,1} a_{i_1 i_2 ... i_N} |i_1 i_2 ... i_N\rangle, \tag{4.34}$$

where the terms $a_{i_1 i_2 ... i_N}$ are chosen randomly. For the product states the wavefunction is initialized as

$$|\psi(0)\rangle = \sum_{i_1=0,1} a_{i_1}|i_1\rangle \sum_{i_2=0,1} a_{i_2}|i_2\rangle ... \sum_{i_N=0,1} a_{i_N}|i_N\rangle, \tag{4.35}$$

where all of the a_i are chosen randomly. Note that for the product state data the single-particle reduced density matrices all have zero entropy and hence there is no entanglement present initially in the ring.

Figure 4.4 shows that the mean single site entropies do not change meaningfully with time for random states, while for states initialized in a product state it increases over time. Thus we make the hypothesis that states initialized randomly are initially at equilibrium with nothing special about the initial state. We note that the entropy of the product states increases up to a point, which is less than the corresponding random state entropy. The figure also shows that as the number of sites increases the qubits on average are more entangled with the entropy per site tending towards unity.

For all of our experiments - unless specified - we use a dataset of 11,000 samples for each lattice size with 8,000 in the training set, 2,000 in the validation set and 1,000 in the test set. The validation set was used to determined when to halt the training. Throughout all results shown are calculated on the test set.

4.4.1 Memory

We begin by seeing how far back we must go to be able to predict the future dynamics of the random state systems. We set the time step at $\Delta = 0.08\pi \approx 0.25$ and predict several time steps into the future. Figure 4.5 shows the results for all qubits and a single qubit respectively. As previously mentioned the single-qubit case never outperforms the all-qubit case in terms of memory usage.

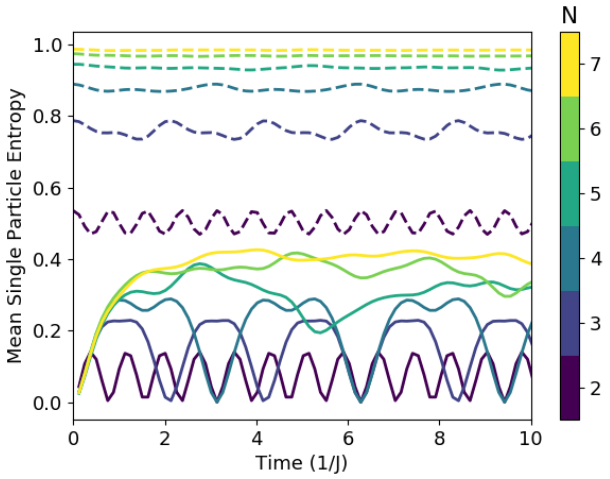


Figure 4.4: Here we plot the mean single site entropy (calculated on the exact data) against time. The dashed lines correspond to randomly generated initial states and continuous lines to product initial states.

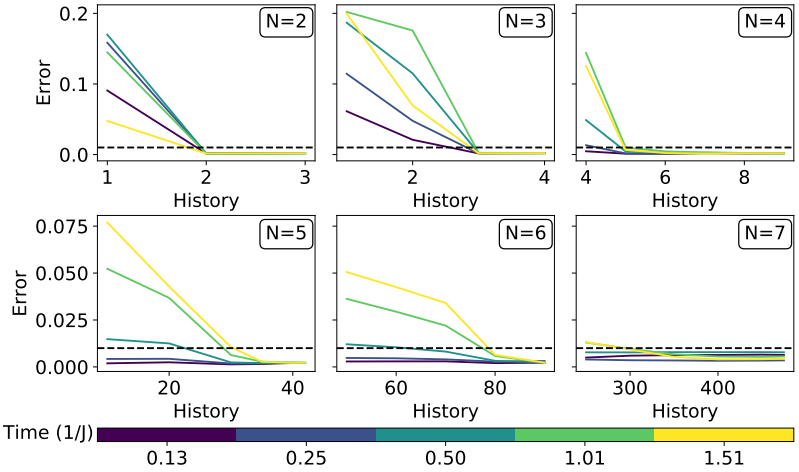
We define the necessary history as the memory needed to predict the state at time $0.48\pi \approx 1.5$ - which recalling, Figure 4.2, is much larger than the decorrelation time - to a mean accuracy of under 0.01. In order to determine what influences the memory we plot the necessary history against the number of qubits, the lowest frequency present in the system and the number of unique frequencies in the system again for $\Delta = 0.08\pi$, shown in Figure 4.5. To find the frequencies present in the dynamics we use Equation (4.17), which implies the density matrix at time t is

$$\hat{\rho}(t) = \sum_{n,m} e^{i(\epsilon_m - \epsilon_n)t} \langle \epsilon_n | \hat{\rho}(0) | \epsilon_m \rangle | \epsilon_n \rangle \langle \epsilon_m |. \tag{4.36}$$

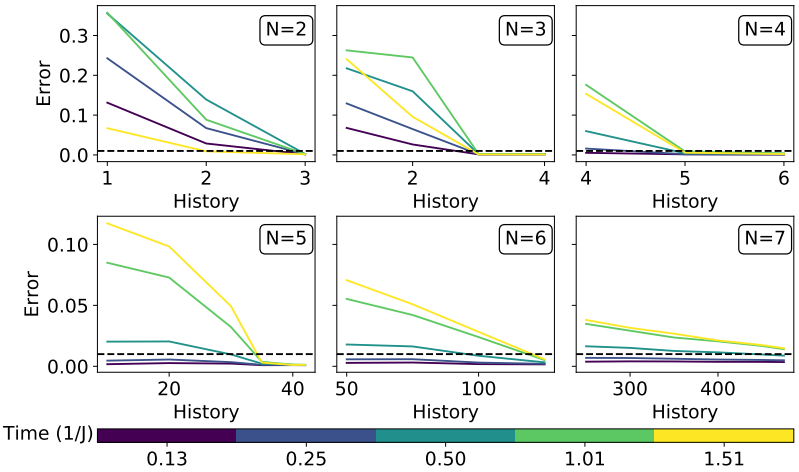
Thus we see that the frequencies are determined by the gaps between the energy levels with $f_{nm} = (\epsilon_m - \epsilon_n)/2\pi$. The number of unique frequencies in the system best correlates with the necessary history. This opens the possibility that by using a different hamiltonian, which has less unique frequencies less history will be required.

4.4.2 Propagation

To show that being able to predict 0.48π ahead into the future, demonstrates that the network has learnt the full dynamics of the system, we perform autoregression, accessing predictions further into the future. In Figure 4.7 we show the



All (a)



Single (s)

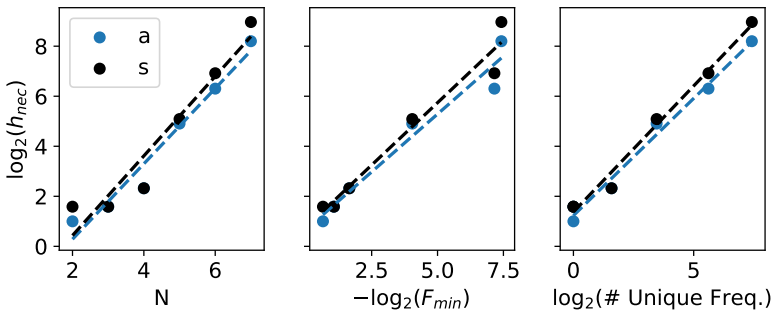


Figure 4.5: Here we show the results for $\Delta = 0.08\pi$. The top panel shows the results with all the qubits in the feature vector and the middle panel shows the results with a single qubit in the feature vector. The error is the mean trace distance and the different colors correspond to the future time predicted. In the bottom panel for both all (a) the qubits and a single (s) qubit in the feature vector we plot the $\log_2$ of the necessary history against the number of qubits, the negative $\log_2$ of the lowest frequency and the $\log_2$ of the number of unique frequencies present in the system.

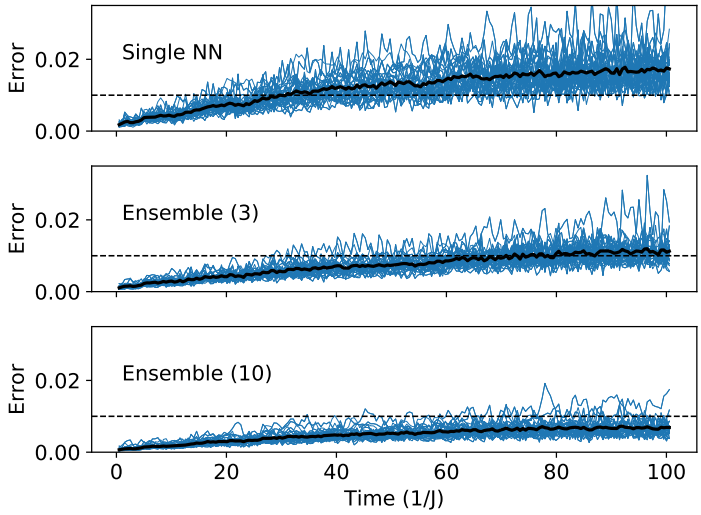


Figure 4.6: Here we show how the mean trace distance error grows over time for autoregression. The bold black line corresponds to the mean over all systems. In the top panel we use one neural net, in the middle panel we use an ensemble of 3 nets and in the bottom panel we use an ensemble of 10 nets. By using a larger ensemble we improve the accuracy of the predictions for longer.

error over time for $N = 6$, random initial states, $\Delta = 0.16\pi$ and all the qubits in the feature vector. Here we compare predictions from a single network and an ensemble of networks. The effect of using an ensemble is dramatic with the predictions staying in a smaller margin of error for longer times. The networks are all different to one another as their weights were initialized randomly.

As discussed in Section 1.5 we can also use an ensemble to estimate the confidence of our predictions by measuring the disagreement between the nets. This is shown in Figure 4.7 where we use the standard deviation of the predictions in the ensemble, as our measure of the variability. This is a very useful feature as when the predictions diverge considerably, we know that our mean prediction is no longer accurate.

Finally although we are using $\Delta = 0.16\pi$ as our time step, we can predict $0.04\pi, 0.08\pi, 0.12\pi, 0.16\pi$ ahead to have a finer scaler, this is again shown in Figure 4.7.

4.4.3 Product State Data

Next we turn to states that were initially in a product state. Recall from Figure 4.4 that now the states gain entropy over time unlike the random data. This

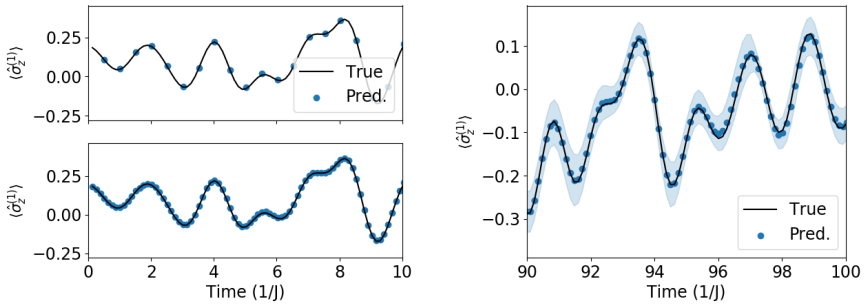


Figure 4.7: Left: Here we show the true and predicted evolution for the z component of the Bloch vector of the first qubit. In the top panel we show the predictions at the time step $\Delta = 0.16\pi$ which is used for autoregression while below we show how the fine grained dynamics can be filled in. Right: The ensemble can be used to generate a confidence interval for our predictions, the shaded blue region, which is simply the standard deviation of the ensemble predictions. All of these predictions were generated using the ensemble of 10 nets.

means that we cannot use autoregression in the same way for the product state data as we would for the random data, since the initial states are qualitatively different from the evolved states. Instead, we use as our initial state for ML, states evolved a time of $0.32\pi \approx 1$ from the initial product state. Figure 4.8 shows the results, noting that the y-axis has a smaller range than the random data, we see that less memory is in general required.

Thus, it seems that the product-state data stays on a small manifold, while the random data explores a larger region. This would explain why the entropy of the product-state data never increases up to the random state levels. To further test this we use an autoencoder to perform non-linear compression on the wavefunctions for the random and product state data. Here we write the wavefunction in the form of Equation (4.18) and use $N = 5$. In the autoencoder, both the encoder and decoder consisted of fully connected neural networks, with two hidden layers each with 64 nodes. For both, the ELU activation function was used. Minimizing the reconstruction error as measured by the Euclidean distance between the vector representations we see how much we can compress the wavefunctions at different times. Figure 4.9 shows the results on a dataset of 2,000 samples with 1,000 in the training set and 500 in the validation and test sets for both sets of data. The figure demonstrates that the product state data can be compressed more efficiently with less loss than the random data

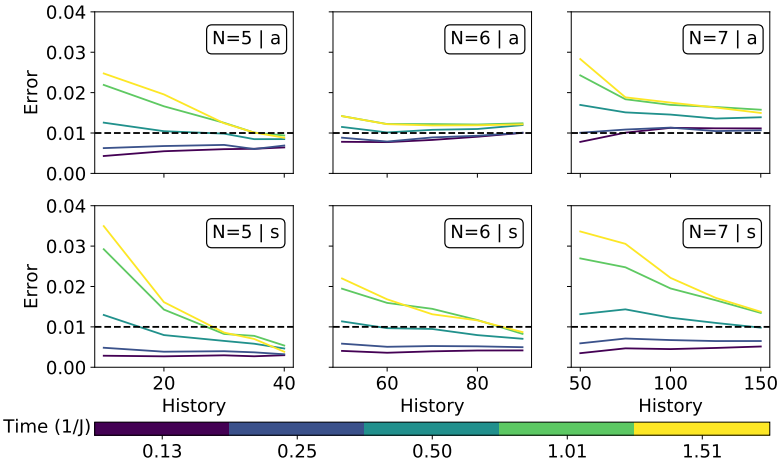


Figure 4.8: Here we plot the error against history for the product state data with all (top) qubits and a single (bottom) qubit. The error is the mean trace distance and the different colors correspond to the future time predicted

over all times, further supporting the reduced manifold hypothesis. Note here that as we are using $N = 5$ the random state data should have zero error when the reduced dimension is 64, as the autoencoder just has to learn the identity function. While the error is low, the reason it is not zero is probably due to the training set not being large enough to cover the possible space of states.

4.4.4 Fine and course graining

From the previous results one may wonder whether there is a sharp discontinuity in the error when a sufficient history is achieved, i.e. the error suddenly drops to zero given a sufficient history. Figure 4.10 shows the error for predictions into the future against the history on a smaller time step of $\Delta = 0.04\pi \approx 0.13$. The figure demonstrates that a discontinuity does in fact exist, with the error sharply falling off when the necessary history is reached. Thus, perhaps the necessary history is in fact a well defined quantity with a finite past containing the all the information required for the dynamics.

A related question is whether there is a limit to how big a time step we can use. Choosing $N = 5$ we plot the error for predicting the state at time $0.48\pi \approx 1.5$ for various values of the time step, Δ, shown in Figure 4.11. We see that for $\Delta = 0.04\pi, 0.08\pi, 0.16\pi$ the amount of time gone back is the same, however note that as we increase Δ we are decreasing the number of points

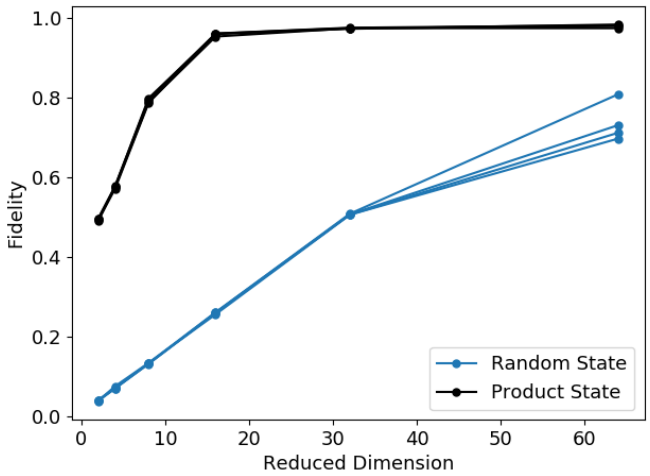


Figure 4.9: Here we plot the autoencoder reconstruction error as measured by the fidelity against reduced dimension both for the product state data and random state data. The plot shows the results for times of 0, 0.25, 0.5 and 1, all in units of J, however there was not any noticeable difference between the times for the most part.

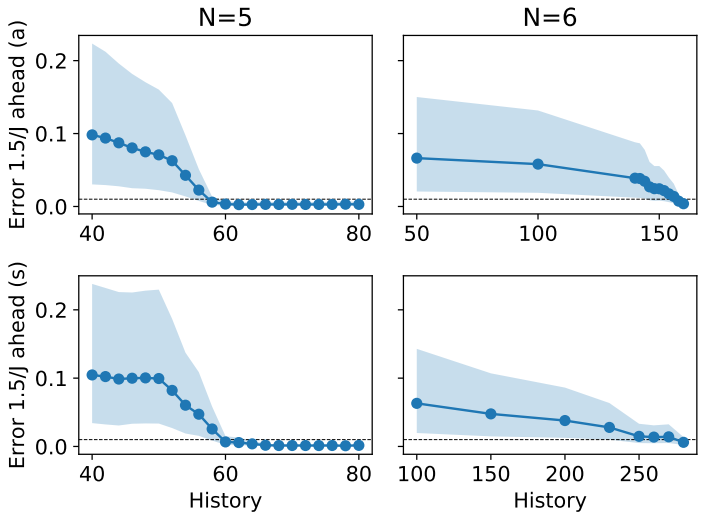


Figure 4.10: For $N = 5$ and $N = 6$ and all-qubits (top) and single-qubit (bottom) in the feature vector we plot the error for a time of $0.48\pi \approx 1.5$ against the history on a fine grained time step of $0.04\pi \approx 0.13$. The edges of the shaded region corresponds to the 25^{th} and 75^{th} percentiles of the error.

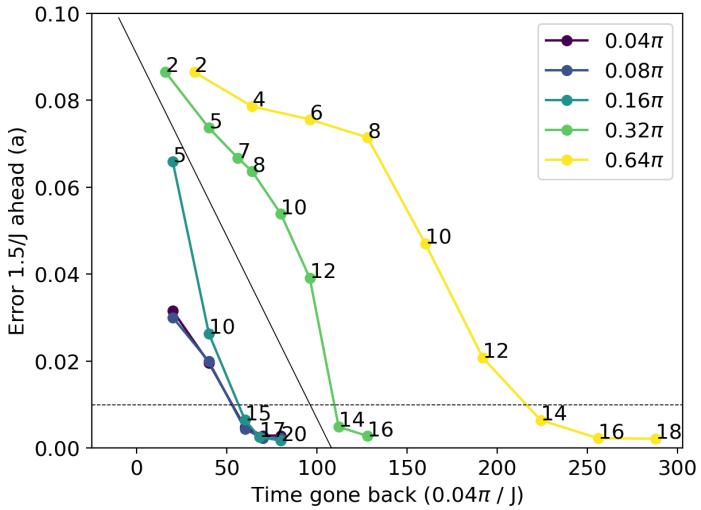


Figure 4.11: Here we plot the error for a time of $0.48\pi \approx 1.5$ and $N = 5$ against the time gone back for a range of time-steps. The black numbers are the number of times sampled in the feature vector and the colors indicate the value of Δ. The black diagonal line separates the results based on whether they are below or above the critical sampling time.

in the feature vector. Hence, for $\Delta = 0.04\pi, 0.08\pi$ the feature vector contains superfluous information that is not needed for predicting the future state. For Δ greater than 0.16π we see that the future dynamics can still be predicted, however we must go back further in the past. Intriguingly at convergence the number of times sampled in the feature vector is roughly the same for $\Delta = 0.16\pi, 0.32\pi, 0.64\pi$.

For $N = 5$ the highest frequency present in the dynamics is 0.99 and by the Shannon-Nyquist Sampling Theorem (see Appendix D) the maximum sampling time is $\Delta_{\max} = \frac{1}{2F_{\max}} \approx 0.16\pi$. This may explain the qualitatively different behaviour seen in Figure 4.11 when Δ is above 0.16π, perhaps, the network must be shown trajectories that go sufficiently far back in time, in order to ascertain the high frequencies. Encouragingly, it seems from the figure, that no matter how large Δ is, the neural network is can still predict the future dynamics. This results means, that efficient propagation far into the future is possible with ML, by training on large time-steps.

In Figure 4.11 we see that the future dynamics for $N = 5$ can be accurately predicted with 14 times sampled in the feature vector. By choosing an irregular series of times in our feature vector can we predict the future dynamics with even

Set	**Error at** 0.32π	**Error at** 0.48π
{ 0 1 2 4 8 16 32 64 }	0.09	0.11
{ 0 1 2 4 8 16 32 64 128 }	0.05	0.05
{ 0 1 2 3 64 65 66 }	0.10	0.11

Table 4.1: The results of the aperiodic experiments. The sets are the time steps in units of 0.04π corresponding to the input. The error corresponds to the mean trace distance error at that time.

less times? Table 4.1 lists the results of several irregular inputs. Disappointingly, we were unable to find a set that had a similar accuracy to the regular times but with less components.

4.4.5 Localization

Finally we investigate how localization effects the ability to predict the future state. Here we use the random initial state data and try to predict the future states of the qubits using different possible sets of the qubits in the feature vector with results shown in Figure 4.12. For $N = 5$ we try the different sets of contiguous qubits. Here, when we have 1 qubit in the feature vector, we are dealing with the single case and when we have all 5 the all case. Concretely, here the size of the feature vector is $3Qh$ where Q varies from 1 to 5, while the target vector has always the same dimension of $3N$. Interestingly we find that, if a qubit is not in the feature vector we cannot accurately predict ts future state. This is surprising, as one might have intuitively thought, that to be able to predict a qubit's evolution, one would have to know how the whole system evolves, but this turns out not to be the case. We also find that only neighbouring qubits are important for the prediction of a qubits future trajectory, as the error on qubit 2 when the first three qubits are used (shown with a dashed line in Figure 4.12) is the same as when all the qubits are used. For $N = 6$ we try two more complex patters, which confirm the previous results.

Intriguingly we also found that including the energy in the feature vector did not help improve the results. As-well we were unable to predict the energy from the feature vector with all the qubits.

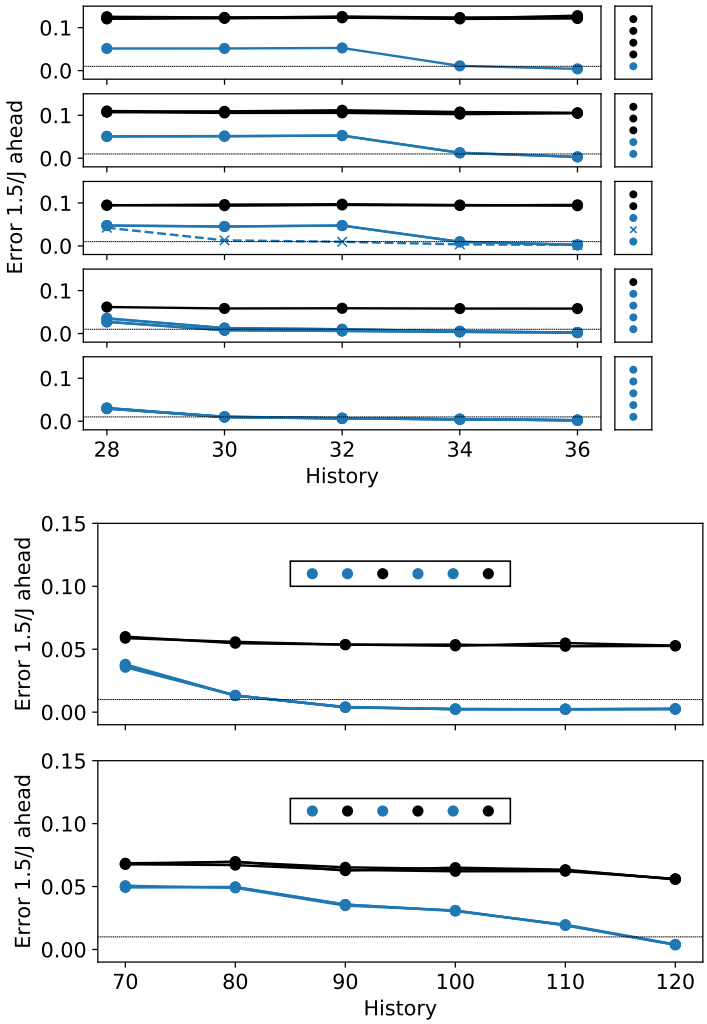


Figure 4.12: Here we plot the future error at a time of $0.48\pi \approx 1.5$ against the history for the random state data and $\Delta = 0.08\pi$. The top panel shows the results for $N = 5$ and the bottom for $N = 6$. Here the blue lines correspond to qubits that were in the feature vector and the black lines correspond to qubits not contained inside the feature vector. The associated diagrams show which qubits in the ring were included.

4.5 Summary

In this work we have explored the use of machine learning to propagate quantum hamiltonians in time. For Markovian systems learning the propagator is simple

and can be done with linear regression as long as there is a sufficiently large number of samples. For the wavefunction the number of samples scales linearly with the Hilbert space dimension and for the density matrix it is a quadratic scaling. For non-Markovian systems we require a history to predict the future dynamics with the length of the history scaling exponentially with the system size. However, the size of the required history can be reduced: by restricting the states to evolve in a smaller subspace via the choice of initial conditions, or perhaps by a careful choice of the hamiltonian. We also explored the nature of the non-Markovian evolution. We found that the history is a well-defined quantity with the error dropping sharply at a sufficient amount of past states and investigated the role of localization in the prediction of the dynamics.

The strength of this investigation is that the supervised learning approach discussed here, can be applied to quantum dynamical data generated from any method, whether it is integration, the density matrix renormalisation group (DMRG) [64] or restricted Boltzmann machines [57]. By learning the propagator for a large time step the system can be evolved into the far future very quickly. There is no limit to the size of the time step for Markovian data and we found no such limit for the non-Markovian data. By using an ensemble of networks instead of a single network we were able to maintain accuracy for a large number of iterations and estimate when our predictions are no longer reliable. Thus this work provides a proof of concept that ML can be used to evolve quantum systems into the very far future efficiently.

Chapter 5

Learning the Curie Temperature

> "Even in the nineteenth century, when we had an impressive theoretical understanding of astronomy, physics and chemistry, the making of iron and steel on which our Industrial Revolution was based was achieved empirically - through intuitive guesswork, careful observation and a huge slice of luck."
>
> - Mark Miodownik, Stuff Matters

The work in this Chapter was published with the title 'Predicting the Curie temperature of ferromagnets using machine learning' in Physical Review Materials [65].

5.1 Introduction

Magnets have been known since antiquity with the ancient Greeks debating the metaphysics behind lodestone [66] - a naturally occurring magnet, composed of magnetite (Fe_3O_4). However it was not until the 1800's that scientists began to understand magnetism, beginning with Oersted's discovery that a current generates a magnetic field, and continuing well into the 20th century with the discovery of quantum mechanics [25].

In ferromagnets, the atomic magnetic moments are aligned with each other creating a long range magnetic order. Ferromagnets have numerous uses from generators and motors to magnetic recording [67]. However, as temperature increases they begin to lose their magnetism, with the magnetization rapidly

falling to zero at the Curie temperature, T_C. For a ferromagnet to have commercial value, its T_C must be much higher than its operating temperature. However the vast majority of known ferromagnets have a low T_C (as shown in Figure 5.2).

Thus it would be very valuable to discover new ferromagnets with large Curie temperatures. For most of human history new magnets have been discovered fortuitously through chance. As this is a slow and random process a more systematic way to discover ferromagnets is preferable. One such method is to do high-throughput calculations [68]. Here one simulates thousands of materials, which are then screened leaving several candidate materials, which could then be experimentally tested. This approach has been successful in discovering new magnets [69]. However the downside to the high-throughput methodology is that it requires large computational resources. Additionally in the case of finding high T_C magnets it is very difficult to estimate the T_C from ab initio computational calculations making screening problematic.

Here we propose a purely data driven approach, where we utilize experimental information. We build up a database of T_C measurements and using ML construct a model to predict the T_C for new materials. This ML model can then be used to screen candidate materials.

5.2 The Curie Temperature

To get a qualitative understanding of how a ferromagnet losses its spontaneous magnetization we will solve a simplistic model. Consider firstly a single spin in a magnetic field, pointing in the z-direction, the hamiltonian is simply $\hat{H} = -B\hat{S}_z$. The eigenvalues are $E_n = \pm B/2$ giving the partition function, $Z = e^{-B/2T} + e^{B/2T} = 2\cosh(B/2T)$, which leads to a magnetization of

$$m = \frac{1}{Z}\sum_n \langle n|\hat{S}_z|n\rangle e^{-E_n/T} = \frac{1}{2}\tanh\left(\frac{B}{2T}\right), \tag{5.1}$$

where $|n\rangle$ are the eigenstates. This is called paramagnetism, in the absence of an external magnetic field, there is no long-range order.

To model the effect of spin interactions we consider the Ising model and applying mean field theory (similarly to section 2.3.3) we then obtain

$$\begin{aligned}\hat{H} &= -J\sum_{\langle i,j\rangle}\hat{S}_z^{(i)}\hat{S}_z^{(j)} \approx -J\sum_{\langle i,j\rangle}(m\hat{S}_z^{(i)} + m\hat{S}_z^{(j)} - m^2) \\ &= m^2 JNr/2 - Jrm\sum_i \hat{S}_z^{(i)}\end{aligned} \tag{5.2}$$

where N is the number of atoms and r the number of nearest neighbours each atom has. Adding in a magnetic field gives the partition function $Z = 2\exp(-m^2 Jr/2T)\cosh\left(\frac{B+Jrm}{2T}\right)$ with magnetization

$$m = \frac{1}{2}\tanh\left(\frac{B+Jrm}{2T}\right). \tag{5.3}$$

This equation must be solved self consistently (note that the magnetization, m, appears on both sides of the equals sign), and we do so for the case of zero magnetic field. At high temperature the right hand side is zero and thus there is no magnetization. While for low temperature the right hand side is $\pm 1/2$. We want to find where there is first some magnetization. By Taylor expanding the right hand side we get $\tanh(Jrm/2T) = Jrm/2T + ...$ and thus

$$T_C = \frac{Jr}{4} \tag{5.4}$$

which is the Curie temperature. Thus we see that at low temperature the spins are aligned in the same direction, but at high temperature they become randomly aligned with no correlation. Figure 5.1 shows the magnetization versus the temperature and demonstrates that the magnetization drops sharply at the T_C.

5.3 Data Collection and Analysis

When we began the project, there was no large database of Curie temperatures available, so our first step was to construct such a dataset. This was done by sifting through the following sources: the AtomWork database [70], Springer Materials [71], the Handbook of Magnetic Materials [72] and the book *Magnetism and Magnetic Materials* [25]. A few additional values have been taken from the references [73–75]. Unfortunately for almost all of this data there was no associated structural information, so we only have the chemical information to use in predicting the T_C. In total we collected 5092 experimental measurements, the breakdown of where the data originated from is:

Source	**Number of data-points**	**%**
Springer Materials	1887	37.1
Handbook of Magnetic Materials	1826	35.9
AtomWork	1220	24
Magnetism and Magnetic Materials	132	2.6
Misc	27	0.53

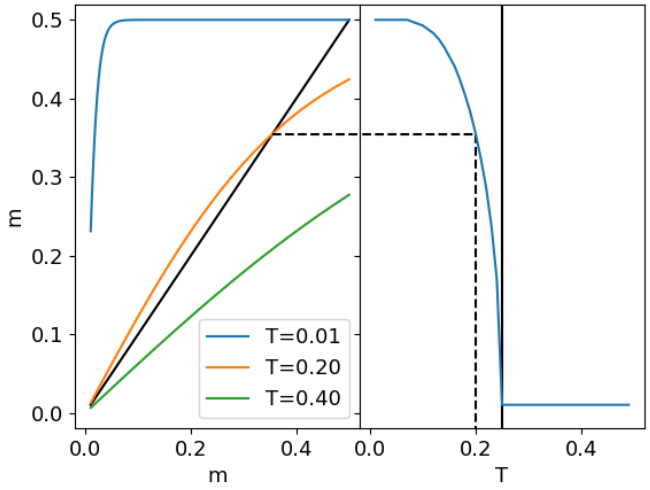


Figure 5.1: Left: Equation (5.3) at zero magnetic field, for various temperatures, recall that m is the expected magnetization, that must be solved self-consistently. For a given temperature where the curve meets the diagonal is the expected magnetization at that temperature. Right: The expected magnetization against temperature, with the Curie temperature occurring at $T_C = \frac{Jr}{4}$. All temperatures are in units of Jr.

Of course extracting all this information from different sources means we invariably end up with duplicates. In general where we have duplicate entries the recorded T_C are similar, with 79% of the compounds with multiple T_C having a difference of less than 50K between the maximum and minimum recorded value. However, there are also a number of compounds with a much larger spread, for 4.9% of the multiple-T_C data the difference between the maximum and minimum T_C value is greater than 300 K. There are numerous possible explanations for such a large spread: the compounds could have different crystal structure or simply there could be recording errors. To deal with this large variance problem we used the median recorded T_C, whenever there were duplicates [1]. The reason for using the median is that it is more robust to outliers than the mean. For example if we have the following T_C for some material $30, 34, 36, 400$ then the median is 35 while the mean is 125. Note that the mean value does not correspond to any measurement unlike the median.

We next applied another stage of preprocessing to the data. Consider the following thought experiment. Suppose we have many entries in the database with very similar chemical compositions. For example we may have many binary alloys of the form A_xB_{1-x} for a small range of x. The problem this leads to is that it biases the generalization error, making it too optimistic. This is because if these compounds are both in the training and test sets, then the test set data will be easy to predict, as its seen something very similar before. However for new data which does not have similar data in the training database, the error will on average be higher then the generalization error. Thus we curate the database, removing data that is too similar. Firstly, we write the chemical formula in a standard notation, by replacing fractional stoichiometry with integer one (e.g. $Cu_{0.5}Ni_{0.5}$ becomes CuNi = Cu_1Ni_1), and by simplify the stoichiometry when possible (e.g. $Ni_{75}Al_{25}$ becomes Ni_3Al = Ni_3Al_1). If there are any duplicates, we take the median as before. Finally we order the database according to the number of atoms in the standardized chemical formula and if the L^1-norm [2] of the elemental atomic fraction vectors between two compounds is less than 0.01 we remove the compound with the larger number of atoms. The reason for doing this is based on our intuition that Fe_5Sn_3 and Fe_8Sn_5, with a distance of 0.019, should be considered as two different compounds, while Co_5Tb_1 and

[1] For a given set of N data-points $\{x_i\}$ the mean is defined as

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

and the median is defined as the middle value in the ordered set of data-points. If there are an even number of data-points the median is the average of the central two data-points.

[2] Given a p-dimensional vector, $\mathbf{x}$, the L^1-norm is given by $\sum_{i=1}^{p} |x_i|$.

$Co_{5.1}Tb_1$, with a distance of 0.005, are essentially the same compound. In this last case we keep only the simpler Co_5Tb_1.

The final stage of preprocessing was to add information about non-magnetic materials. This was done to make the model more robust to data unlike anything it had seen before. To do this we added all the elements of the periodic table that do not have a T_C into the dataset, assigning them a T_C of 0 K. A similar idea was used in the task of machine learning the superconductivity critical temperature [76] where materials which were not found to be superconducting were added to the dataset to improve the generalization of the ML workflow. The elements with a non-zero T_C are:

Name	T_C (K)	**Name**	T_C (K)	**Name**	T_C (K)
Co	1380	Tb	220	Er	20
Fe	1040	Dy	85	Ho	20
Ni	630	Nd	30	Pr	8.7
Gd	290	Tm	30		

After all of this preprocessing we had a total of 2633 data-points. Figure 5.2 shows a histogram of the T_C, which demonstrates that the majority of the data has a T_C below room temperature. The median value is, in fact, 226.9 K. Figure 5.2 also shows the elemental composition of the dataset. Unsurprisingly the most abundant elements are Co, Fe and Gd, which are all ferromagnetic. Note that in both plots we did not include the non-ferromagnetic data.

5.4 Representing Materials

5.4.1 Criteria

When applying machine learning to molecules and crystals the most important difference between the various approaches is how they represent the object. As we are dealing with experimental data it is difficult to increase the size of our database which makes the designing of a good feature vector all the more important. To quote 'Deep Learning' [8]

> the amount of skill required [to create a good feature vector] reduces as the amount of training data increases.

The design of good feature vectors can be though of as incorporating prior expectations into the model. As our dataset increases in size the importance of the prior decreases and our model can become purely data driven.

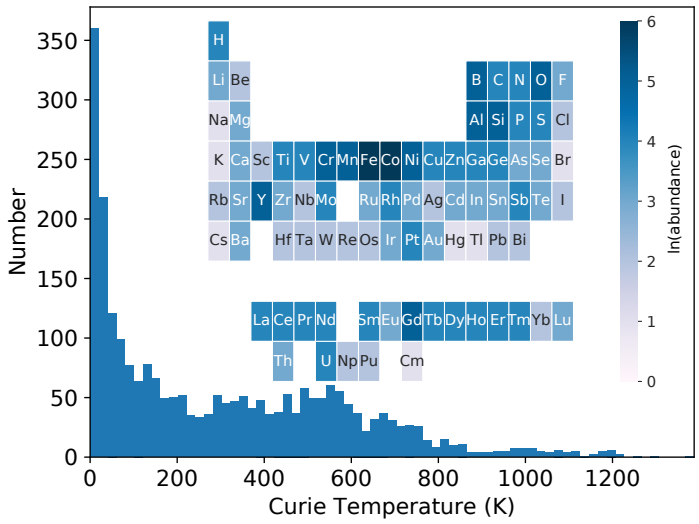


Figure 5.2: A histogram showing the distribution of the Curie temperatures in our dataset. The first quartile, median and 3rd quartile were found to be 7.8 K, 226.9 K and 510 K respectively. The insert shows how the elements are distributed in the database. The quantity here is the log of the abundance, where the abundance is simply how many samples the element appears in. The most abundant elements in decreasing order were Co, Fe, Gd, Al, Ni and Mn.

The number of possible features and sets of features is infinite. To begin designing candidate feature vectors we list what properties the ideal representation should have, with many of these suggestions coming from the literature [77–79]:

1. Two different materials should only have the same representation, if their target variables are the same. If two different materials have the same representation then our model will be unable to distinguish between them and will predict the same target variable for both of them.

2. The descriptor should be invariant under rotations, translations, permutations and different choices of unit cell. Since these operations leave the material unchanged, the representation should not change.

3. The descriptor should be continuous for small variations. Any properties of the material will change by a small degree for minor variations and the representation should reflect this.

4. It should not be extremely computationally expensive to generate the descriptor. If this requirement is not met then it might be best to calculate the target variable directly from first principles.

5. The dimensionality of the feature vector should be as low as possible. A smaller feature vector helps interpretability.

6. The descriptor should be independent of the number of atoms in the cell of the crystal. Then we would be able to compare materials with different numbers of atoms in the unit cell.

As most of our data does not have any structural information we next discuss how to represent materials based on only chemical information. For some of the data we were able to obtain structural information and we discuss the issues of incorporating structure and the results in Section 5.6.

5.4.2 Composition Representation

Only using compositional features violates the first requirement, as features with the same composition but different structure are treated the same, but for most of our data we have no choice.

Given a compound $\mathrm{E}_1 n_1 \mathrm{E}_2 n_2 ...$ where $\{\mathrm{E}_i\}$ are the elements and $\{n_i\}$ their respective atomic fractions, we can divide the compositional features derived from it into three sets

- stoichiometric features which depend on $\{n_i\}$
- elemental features which depend on $\{\mathrm{E}_i\}$
- general features which depend on both $\{n_i, E_i\}$

So for example, given Fe_3O_4 the stoichiometric information is $(3/7, 4/7)$ and the elemental information is $(\mathrm{Fe}, \mathrm{O})$.

Starting with the stoichiometry, possible features could include: norms (as suggested by Ward [80])

$$L^p(\{n_i\}) = \left(\sum_i n_i^p\right)^{1/p} \tag{5.5}$$

or the entropy

$$S(\{n_i\}) = -\sum_i n_i \log n_i. \tag{5.6}$$

In both cases one wonders how informative such features would be. They effectively quantify how mixed the composition is, for unaries (only one element present) the entropy is zero and the norms are 1, while for a composition where there are c elements in equal parts the entropy is $\log c$ and the norms are $c^{1/p-1}$. While intuitively this information does not seem relevant, the philosophy behind data driven modelling is to include as much information as possible and let the models find what information is pertinent.

Next we could include elemental only information, where we ignore the stoichiometry completely. An example of this could be the maximum atomic number present, $\max Z(\{\mathrm{E}_i\})$, or the minimum elemental electronegativity, $\min \zeta(\{\mathrm{E}_i\})$. However features of this kind violate the third requirement, continuity. To see this consider the compound $\mathrm{A}_{1-\epsilon}\mathrm{B}_\epsilon$ and suppose the atomic number of B is greater than A, $Z(B) > Z(A)$. If we have the maximum atomic number as a feature, then for all non-zero values of ϵ it will be $Z(B)$, whereas for $\epsilon = 0$ it will be $Z(A)$, thus discontinuous. This would be very strange, we could have an extremely low concentration of element B yet its playing an important role as a feature. For this reason we have chosen not to use any features of this kind except the mode, which is discussed below.

Finally what about including both stoichiometry and composition? Here we consider two approaches. Firstly we can simply define a vector where each component corresponds to the atomic fraction of an element. If we order the elements by atomic number then the first component would represent the atomic fraction of Hydrogen present. For example we would represent Fe_3O_4 as

$$(0, , ..., 0, \underbrace{4/7}_{\mathrm{O}=8}, 0, ..., 0, \underbrace{3/7}_{\mathrm{Fe}=26}, 0, ..., 0) \tag{5.7}$$

where the under-braces are the positions of the non-zero terms.

Next we can include atomic-weighted (AW) components, where we weight elemental properties by their atomic fractions. Concretely for some property $P(E)$ we compute the mean

$$\langle P \rangle = \sum_i n_i P(E_i) \tag{5.8}$$

the absolute deviation

$$\langle \Delta P \rangle = \sum_i n_i |P(E_i) - \langle P \rangle| \tag{5.9}$$

and finally the mode

$$\langle |P| \rangle = P(E_i) \quad \text{with} \quad n_i = \arg\max(\{n_i\}). \tag{5.10}$$

Note that the mode is a discontinuous descriptor [3], however unlike the max or min previously discussed it takes into account concentration and thus we do not run into the problem of a very small concentration of some element having a large effect in the feature vector. We create these atomic weighted statistics for a variety of properties listed below.

Thus in total, combining all of the features discussed we end up with a feature vector with 129 components. Explicitly it is:

Features	**Symbol**	**Dimension**
L^p stoichiometry norm ($p = 1, 2, 3$)	$\|\|x\|\|_p$	3
Stoichiometry entropy	S	1
Atomic fraction vector	$\mathbf{v}_{\text{chem}}$	84
AW atomic number	$\langle Z \rangle$, $\langle \|Z\| \rangle$, $\langle \Delta Z \rangle$	3
AW valence electrons	$\langle N_{\text{V}} \rangle$, $\langle \|N_{\text{V}}\| \rangle$, $\langle \Delta N_{\text{V}} \rangle$	3
AW period	$\langle P \rangle$, $\langle \|P\| \rangle$, $\langle \Delta P \rangle$	3
AW group	$\langle G \rangle$, $\langle \|G\| \rangle$, $\langle \Delta G \rangle$	3
AW molar volume	$\langle V \rangle$, $\langle \|V\| \rangle$, $\langle \Delta V \rangle$	3
AW melting temp.	$\langle T_{\text{M}} \rangle$, $\langle \|T_{\text{M}}\| \rangle$, $\langle \Delta T_{\text{M}} \rangle$	3
AW electronegativity	$\langle \epsilon \rangle$, $\langle \|\epsilon\| \rangle$, $\langle \Delta \epsilon \rangle$	3

The atomic fraction vector has dimension 84, as not all elements in the periodic table can be found in the ferromagnets included in our database (e.g. He, Ar, etc.). The numerical values of the elemental properties are taken from Pymatgen [81].

[3] the mode of $A_{1-\epsilon}B_\epsilon$ is discontinuous at $\epsilon = 0.5$.

5.5 Results

5.5.1 Methodology

As our dataset is small splitting it into three subsets - train, validation, test - would effect the accuracy, so instead we combine the training and validation sets and use 3-fold cross validation (see Section 1.2). We compare four different models, each operating under different principles, namely Ridge Regression (RR), Neural Network (NN), Kernel Ridge Regression (KRR) and Random Forests (RF). Each model is described in detail in Chapter 1. The neural network is implemented using Keras [47], while for the rest we use Scikit-Learn [18]. The cross-validation set is used to set the hyper-parameters which are the regularization parameters for RR and KRR, the dropout rate for NN and the maximum tree depth for RF.

The accuracy of many algorithms, especially models based on using distance, can be improved by reducing the dimension of the feature space. If our feature vector has dimension p, then there are 2^p possible subsets one could construct [4]. For our case this corresponds to roughly 10^{39} subsets, thus it is impossible to perform an exhaustive search for the optimal subset. Instead, we try two different quick methods of dimension reduction. The first, Correlation (C), ranks the features according to the absolute value of the Pearson correlation coefficient, which for two variables x and y is defined as

$$c_{\mathrm{P}} = \mathrm{cov}(x, y)/\sigma_x\sigma_y \tag{5.11}$$

with $\mathrm{cov}(x, y)$ being the (x, y) covariance and σ the standard deviation. In this case y is the T_C and x corresponds to each feature. The second, Principle Component Analysis (PCA), is described in Section 1.6.1.

5.5.2 Best Model

The results of the 3-fold cross validation score for the different algorithms together with the different feature reduction techniques are shown in Table 5.1. Here the metric used is the R^2 value which is computed as

$$R^2 = 1 - \frac{\sum_i (y^{(i)} - \hat{y}^{(i)})^2}{\sum_i (y^{(i)} - \mu)^2}, \tag{5.12}$$

where y are the true values, $\hat{y}$ the predicted and μ is the mean of the true values. Note that the R^2 has a maximum value of 1, which corresponds to a perfect fit of the data, while it is possible for it to be arbitrarily negative. We

[4] This follows since each feature could be include in the set or excluded, hence 2^p.

$\mathbf{R^2}$	**All**	**C10**	**C20**	**C40**	**C80**	**P10**	**P20**	**P40**	**P80**
RR	0.53	0.48	0.5	0.52	0.52	0.24	0.27	0.31	0.39
KRR	0.69	0.72	0.72	0.72	0.69	0.69	0.72	0.70	0.68
NN	0.76	0.72	0.76	0.77	0.78	0.73	0.77	0.77	0.77
RF	0.81	0.76	0.77	0.78	0.79	0.72	0.74	0.73	0.72

Table 5.1: 3-fold cross-validation R^2 score of all the algorithms chosen combined with the various feature reduction techniques. Here "All" indicates the case where no feature reduction is applied. "C" means Correlation feature reduction and "P" is for PCA. The number beside the type of feature reduction scheme indicates the size of the reduced feature space.

note that KRR, which is based on distances computed in high dimensions, is especially improved by feature reduction techniques. However, in general the feature reduction techniques didn't impact the results greatly.

Overall the best model was random forests with no feature reduction technique. The model has a mean absolute error of 57 K as measured on the test set, with its predictions on the test set shown in Figure 5.3. The absolute error seems to follow an exponential distribution, $f(x) = \lambda e^{-\lambda x}$, with $\lambda = 0.018$. The cumulative distribution of the exponential distribution is $1 - e^{-\lambda x}$, which in this case tells us that about 80% of the data had an error less than 100 K.

A common criticism of machine learning is that the models are black boxes, which operate in ways unknown to us. Of course any complex mapping will always be some what of a black box as the more you simplify the more accuracy you lose. However with random forests we can estimate what the most important features are for making predictions [18]. Each feature is assigned with a non-negative number, the higher the number the more important that feature is, with the numbers summing up to one. As expected, the atomic fraction of Fe and Co are the most important features with respective scores of 0.27 and 0.14. What perhaps was not expected is that no other features stood out as being important. Around 40 features together accounted for 95% of the importance.

5.5.3 Uncertainty and Robustness

For most materials in the test set our model is very accurate, however there are outliers. Thus it would be very helpful if the model could provide a confidence of its prediction. As discussed in Section 1.5 it is straightforward to do this with an ensemble model like random forests. One method is to take the standard

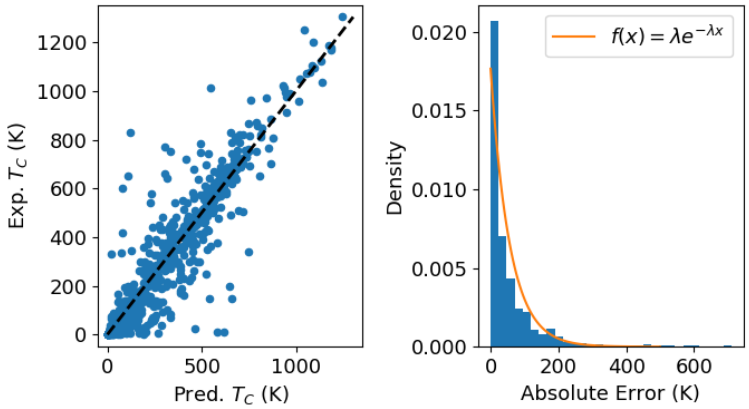


Figure 5.3: Left: Here we plot the test set experimental T_C against the predictions from the best model. Right: A histogram of the absolute test error, $|y - \hat{y}|$, of the model. We fit an exponential distribution to the histogram with with decay coefficient, λ, of 0.018.

deviation of all the predictions as our confidence interval around the mean. However, suppose the mean is small but the standard deviation is large, then the confidence interval will include negative numbers. It makes more sense to use an asymmetric confidence interval. To do this we divide the 50 trees - which comprise the random forest - into 5 subsets, average over these subsets and take the min and max means as our intervals. One should bear in mind that we are just looking for a qualitative estimate of the uncertainty, not for a fully quantitative analysis. Specifically, the most important function of the uncertainty, is to flag predictions that are likely to be widely incorrect. Figure 5.4 shows how accurate the confidence intervals are, measuring the fraction of the test data within some deviation of the confidence intervals. Note that 82% of the experimental values in the test set are within 50 K of the confidence interval.

To further test our model is we give it the task of predicting phases. This requires the model to extrapolate to regions of little training data. Namely we predict the binary phases of Co-Mn, Fe-Ni and Ni-Rh shown in Figure 5.5 and the ternary phase of Al-Co-Fe shown in Figure 5.6. The phase data was collected from the following sources [82–85] with some of the points already being in the training set as they came from other sources. In Figure 5.5 points in the training set are denoted with an 'x', while in Figure 5.6 they are denoted by a 'x' for the binary phases and a square for the heat map.

The manganese and cobalt magnetic phase diagram comprises of both fer-

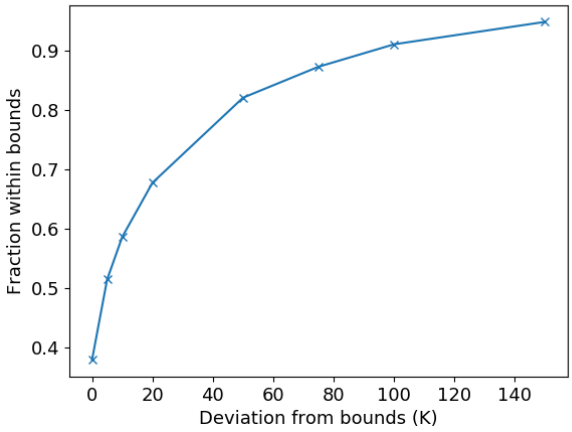


Figure 5.4: Here we show how well the confidence intervals fare, by plotting the fraction of the test set data within a deviation of the bounds. For a given prediction, the deviation from bounds, is how far outside the confidence interval the experimental value is. If the experimental value is inside the confidence interval, then the deviation is zero.

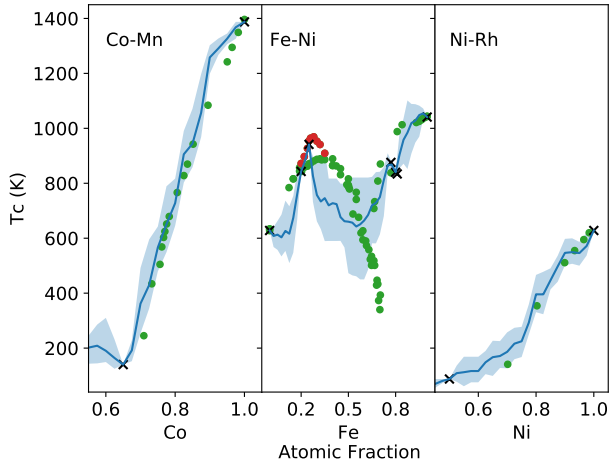


Figure 5.5: From left to right the binary phase predictions for Co-Mn, Fe-Ni and Ni-Rh. The x-axes are the fraction of Co, Fe and Ni respectively. The blue solid line is the mean prediction with the shaded region being the confidence interval. Points that were in the training set are denoted with a black x and experimental data is denoted with a green dot. In the Fe-Ni plot, the red dots correspond to the T_C associated to the $FeNi_3$ intermetallic phase, while the green ones correspond to random Ni-Fe alloys.

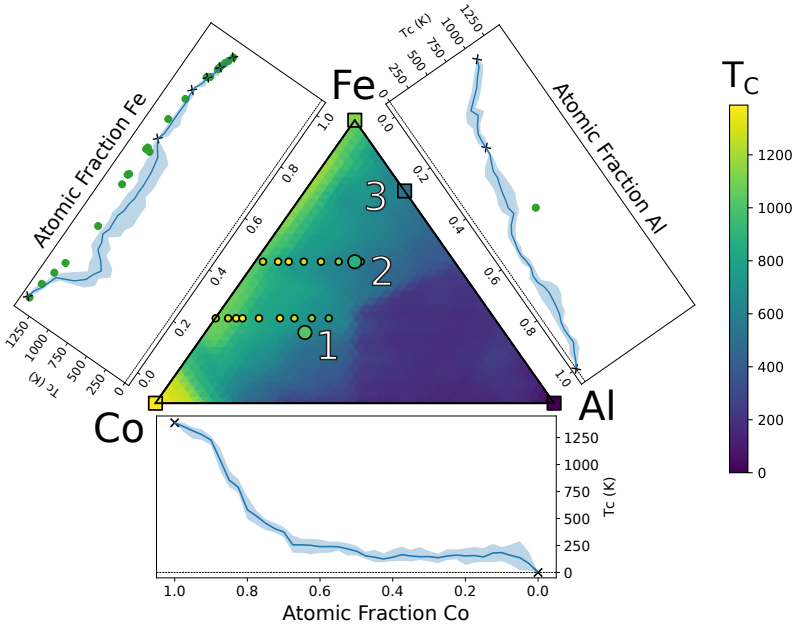


Figure 5.6: The ternary phase predictions for the system Al-Co-Fe. Inside the heat map, points in the training set are denoted by a square, while for the binary phases, they are denoted by a black x. Numbers correspond to three known stoichiometric phases: 1) Co_2FeAl, T_C = 1,000 K, 2) Fe_2CoAl, $T_C > 873$ K, 3) Fe_3Al, T C = 573 K. The plot was partially made using the python-ternary package[86].

romagnetic and antiferromagnetic orders [82, 87]. In particular when the Co atomic fraction is larger than about 0.68 the alloys are ferromagnetic with a T_C that monotonically increases as a function of the Co content. In contrast, Mn-rich alloys are antiferromagnetic with a Néel temperature [5] that this time monotonically increases as the Mn concentration gets larger. Our ML model predicts well the T_C for all the ferromagnetic phases, with a constant minimal error. For this case only two data points were included in the training set, namely elemental Co and the end-of-the-series alloy, $Co_{13}Mn_7$. Intriguingly, the model seems to be able to predict also the upturn in critical temperature occurring for Co atomic fractions lower than 0.68, although these correspond to antiferromagnetic phases. It is worth noting that the spread of values returned by the RF algorithm is certainly larger for these Mn-rich phases.

The Ni-Fe system presents a different level of complexity, with the ferromagnetic order being present over the entire composition range [83]. In the region of temperatures relevant for T_C the Fe-rich phases (for a Fe atomic fraction down to 0.65) are characterized by Ni-doped *bcc* iron with a T_C that grows monotonically as a function of the Fe fraction. In contrast, when the Fe atomic fraction is reduced below 0.65 the relevant phase is an *fcc* random alloy, which can be stabilized up to bulk Ni. In this case the T_C is non-monotonic, it increases with the Ni atomic fraction up to 0.70 (the maximum T_C is about 870 K for $Fe_{30}Ni_{70}$) and then it decreases down to the T_C of bulk Ni. Furthermore, there is also an ordered intermetallic $FeNi_3$ ferromagnetic phase, which persists over a relatively narrow composition range. This means that in $Fe_{1-x}Ni_x$ at $x \sim 0.75$ there are two ferromagnetic phases with different T_Cs. Also in this case our ML model performs rather well. This time six experimental data points were included in the training set, two for the elemental Ni and Fe and four across the $Fe_{1-x}Ni_x$ alloys. In particular we included both the composition corresponding to the minimum T_C and that corresponding to the maximum at the intermetallic $FeNi_3$ phase. Our ML model interpolates well between these points, in particular in the Ni-rich part of the composition diagram. As expected from the fact that structural information are not included in the model, we are not able to distinguish the different T_Cs associated to different structures.

Next, we look at the Ni-Rh binary system, an alloy where only one of the two elements is magnetic. As with several other elements of the Pt group, Rh is highly soluble in Ni and random alloys can be formed over almost the entire composition range. Here the T_C monotonically decreases from that of bulk Ni

[5] For antiferromagnets, the critical temperature at which the magnetic order disappears is called the Néel temperature [25].

as Rh is added to the alloy. This continues up to a critical composition, found for a Ni atomic fraction of around 63%, where the ferromagnetism disappears completely [88]. Our ML model is fully capable of describing such behaviour. In particular the ML model successfully predicts the critical concentration for the suppression of ferromagnetism at a Ni atomic fraction lower then 0.7. This is a rather compelling result.

Now, we turn to the ternary system Al-Co-Fe. Note that this ternary system includes three stoichiometric magnetic phases, namely Co_2FeAl, Fe_2CoAl and Fe_3Al. Furthermore, magnetic solid state solutions can be stabilized over a rather large composition space.

As for the other binary systems investigated our machine learning model appears well able to describe the main features of the three relevant binary alloys, namely Al-Co, Al-Fe and Co-Fe. This is despite the fact that only a rather limited number of compounds across the various phase diagrams were included in our training set. For instance, the model is capable of describing the critical Al concentration for the appearance of ferromagnetism in Al-Co [89], which has been measured to be at an Al atomic fraction of about 50%. In this case only the end points of the composition diagram, namely elemental Co and Al, were present in the training set, so thus, the ML model is able to extrapolate such a critical concentration from the learning obtained in other parts of the chemical space.

The case of Al-Fe is relatively more complex. Our ML model is trained with only three data points from this binary system, namely the end points and the Fe_3Al stoichiometric phase. The resulting T_C-versus-composition curve then predicts a monotonic reduction of the Curie temperature as the Al atomic fraction increases, with a rather smooth approach to $T_C = 0$ for pure Al [90]. Experimental data obtained for rf-sputtered thin films display a well-disordered *bcc* phase for Al atomic fractions up to 70%, followed by an amorphous phase between 75% and 85%, and then by an *fcc* phase for higher Al fractions. The disordered *bcc* phases are all ferromagnetic, while both the amorphous and the *fcc* ones are paramagnetic down to 4.2 K [90]. Although the precise position of such phase boundaries seems to depend somewhat on the details of the growth conditions [91], our ML model appears also in this case to be able to capture such behaviour.

At variance with the Al-Fe and Al-Co case, the T_C of the Co-Fe binary system presents a non-monotonic behaviour with composition. As we move from elemental Co to elemental Fe, first the T_C decreases with increasing the Fe content, up to a Fe atomic fraction of around 30%. In this case the magnetic

transition takes place within the Co-Fe γ-phase (*fcc* structure) [92]. Then, for Fe atomic fractions comprised between ∼30% and ∼80%, T_C first increases and then decreases, reaching up to a maximum at around a 50-50 composition. Such behaviour effectively traces the phase boundary between the γ and the α phases (*bcc* structure). At the end of the composition range, for Fe atomic fractions above 80%, T_C keeps decreasing down to that of elemental Fe, but the alloys remain in the α phase, namely the magnetic phase transition no longer traces the structural phase boundary.

Finally, we take a look at the T_Cs for the ternary phases, namely in the center of the composition diagram. In this case no experimental information was included in the training set, so that the predictions are based only on the knowledge of ternary phases with chemical compositions different from those belonging to the Al-Co-Fe system. Two ternary stoichiometry compounds are known for Al-Co-Fe, namely the Heusler alloys Co_2FeAl and Fe_2CoAl. Their Curie temperature are 1,000 K for Co_2FeAl [93] and at least 873 K for Fe_2CoAl [94] (T_C has not been measured with precision and only a lower bound is available). Our ML model predicts respectively 657 K and 580 K, hence it provides an underestimation of the real T_Cs, although it ranks the materials in the right order. Additional experimental data are available across the Al-Co composition (for an Al atomic fractions not exceeding 30%) and constant Fe atomic fractions of 30% and 50% [95]. These data are reported as full circles in Figure 5.6 showing the ability of our ML model to describe the general trend, namely a decrease in T_C as the Al atomic fraction increases. Also in this case some non-monotonicity is found in the experimental data for small Al concentrations, which is associated to the fact that for such composition range the T_C traces the phase boundary between the α and γ phases. Our ML model appears to be able to trace such non-monotonicity.

5.6 Incorporating Structure

The most straightforward way to improve upon these results is to include structural data. This will allow the ML algorithm to differentiate materials with the same chemical composition but different structure. Thus, in principle, with enough data one could achieve an arbitrarily low error as the T_C is a property of the material. For 792 of the entries in the database we were able to get a structure from the AFLOW ICSD library [96, 97].This is the ground state structure as calculated by Density Functional Theory, which we must bear in mind introduces a source of error. We now proceed to explain how we have

incorporated the structure into our descriptor and then describe results.

5.6.1 Representation

There are several ways in which structure can be incorporated into the descriptor. The simplest approach is to compute several descriptors, which contain information about the structure. For example we could include the density of the material as a feature. Or following Dam et al [98] we could include the average distance between different types of atoms.

Alternatively, we could use a descriptor, which encodes the global structure of the material. Here we use a modified version of the radial distribution descriptor suggested by Schutt et al [99]. The idea is to look at the pairwise atom distances inside some cut-off volume. The most simple form is the function

$$f(r) = \frac{1}{N\text{cell}} \sum_{i=1}^{N_{\text{cell}}} \sum_{j} \theta(d_{ij} - r)\theta(r + \Delta - r_{ij}), \tag{5.13}$$

where N_{cell} is the number of atoms in the unit cell, i runs over atoms in the unit cell, j runs over all atoms within the cut-off volume, d_{ij} is the distance between atoms i and j, $\theta(x)$ is the Heaviside step function [6] and Δ is a parameter specifying how fine grained the function is. In Figure 5.7 we plot this function for a *bcc* material with $\Delta = 0.02$. A problem with this descriptor is that it does not differentiate between atoms of different species. We can include this information and avoid making the feature vector too high dimensional by using the descriptor

$$f_\alpha(r) = \frac{1}{N\text{cell}} \sum_{i=1}^{N_{\text{cell}}} \sum_{j} \theta(d_{ij} - r)\theta(r + \Delta - r_{ij})\delta_{\alpha,ij}. \tag{5.14}$$

Here α specifies the type of interaction and $\delta_{\alpha,ij}$ is zero if atoms i and j are not of that type and one otherwise. Dividing the elements into three categorises: transition metals (T), lanthanide (L), and other (O), we consider 6 types of interactions: T-T, T-L, T-O, L-L, L-O and O-O. Note that $f(r) = \sum_\alpha f_\alpha(r)$.

5.6.2 Results

Table 5.2 shows the three-fold cross validation score of several different descriptors for incorporating the structure. We use both random forests and kernel ridge regression for our models. We compare four different descriptors. "Original" is just the original set of features that describe composition, "Original +

[6] The Heaviside step function is defined as $\theta(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0. \end{cases}$

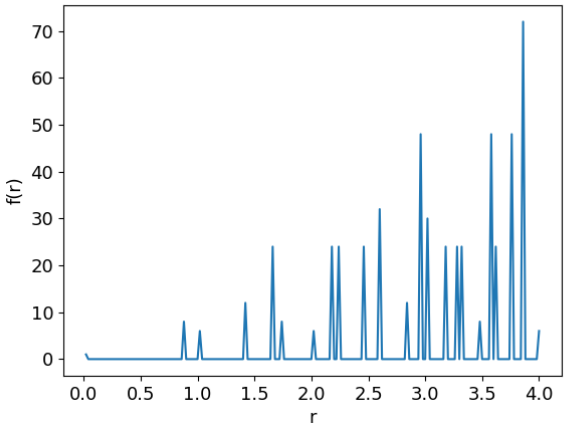


Figure 5.7: Here we show the radial distribution function, $f(r)$, for a *bcc* type material with lattice vectors $\mathbf{a}_1 = \frac{1}{2}(-1,1,1)$, $\mathbf{a}_2 = \frac{1}{2}(1,-1,1)$ and $\mathbf{a}_3 = \frac{1}{2}(1,1,-1)$.

Features	**RF R^2**	**KRR R^2**	**Dimension**
Original	0.75	0.64	102
Original + Volume	0.76	0.64	103
Original + f	0.74	0.63	122
Original + f_α	0.74	0.64	220

Table 5.2: 3-fold cross-validation R^2 score of the RF and KRR models for various features that incorporate the structure and for the original feature set that ignore the structure. Also listed is the dimension of the feature vector.

Volume" is the original set with an additional feature of unit cell volume, finally we try the original set with the two structural functions discussed in the last section.

Disappointingly incorporating the structure was not found to help improve accuracy. While the feature set with volume was the best performing model, the improvement is so slight that it is hard to know if this is a robust result. Reassuringly, recently published work [100] has verified these results also finding that structure was not found to improve performance and obtaining very similar results to ours. Perhaps the mapping between structure and Curie temperature is so complex that it will require a large amount of data to accurately model or a very carefully chosen descriptor. One must also bear in mind here, that we were only able to obtain structural information for a small subset of the dataset

and thus we are training these structural models on a limited amount of data.

5.7 Summary

In this Chapter I detailed our work in developing a model capable of predicting the Curie temperature from a materials chemical composition alone. The model achieved an impressive accuracy, with a mean absolute error of 57 K on an independent test set. As well we are able to assign a confidence to our predictions making the model suitable for real world applications. Ultimately to improve the accuracy the structure of the material must be taken into account and to incorporate this extra information in a data driven approach will require a very large dataset. Our model perhaps is best though of as a rough guide, which could be utilized as part of a pipeline to screen candidate materials.

Chapter 6

Learning to Generate Molecules

> "Base metals can be transmuted into gold by stars, and by intelligent beings who understand the processes that power stars, but by nothing else in the universe."
>
> - David Deutsch, The Beginning of Infinity: Explanations That Transform the World

6.1 Introduction

The space of potential materials and molecules is enormous, if not infinitely big. To give one example, it was estimated that the number of stable small organic molecules is over 10^{60} [101]. Somewhere in the vast chemical space are materials with properties that would revolutionize society, from drugs which could cure currently untreatable diseases, to materials that can store energy more efficiently. This constitutes perhaps the most ambitious needle in a haystack problem of all time.

In the last chapter we saw how ML can be used to quickly predict properties of materials, speeding up the process of material discovery. Here we tackle the more complex problem of creating an AI agent capable of suggesting new compounds, specifically molecules, that satisfy some criteria. In recent years several attempts at using ML to discover new materials have been proposed [102]. While some of these approaches are for crystals [103, 104] most are for molecules. A limitation of the majority of the methods for molecules is that they do not take into account the structure of the molecule, instead they represent it

by either a string [105, 106] or a graph [107, 108]. Obviously these methods will not be adequate for discovering molecules with properties that depend strongly on the structure.

There has been some work on incorporating molecular structure into a ML framework for the discovery of molecules. One method [109] treats the molecules as 3-dimensional images and applies CNNs to learn the chemistry. This has two drawbacks: firstly there is maximum image size and hence the model is limited to what it can extrapolate to, and secondly, treating the molecule as an image does not encode any of the symmetries of the molecule and hence the approach is inefficient. Another method [110] is based on learning conditional probabilities and outputting the molecule atom-by-atom. However, this method is not ideal as the final molecule will be strongly influenced by the initial set of outputted atoms.

This Chapter details my research into creating a ML model that can output the structure of molecules with certain desired properties, and avoids the aforementioned problems. To achieve this I apply generative adversarial networks (GANs) to the task. For the theory behind GANs, see Chapter 1. GANs are not the only method for the unsupervised learning of probability distributions, for example one could use variational autoencoders [111] or Deep Boltzmann Machines [8]. However, we choose GANs over the alternatives for the reason that we want the model to learn the relevant local chemistry, which is straightforward to implement in the GAN framework. As explained later, the reason for this is that if the model learns local correlations then it can generalize to molecules larger or different to what it has seen before. An additional benefit of GANs is that they are considered to produce the best quality samples.

This Chapter begins with a discussion on how to represent molecules within the GAN framework. I then show how GANs can produce molecules from a given distribution and finally how we can produce molecules with certain properties.

6.2 Representing Molecules

In Chapter 5 I considered the problem of how to represent a material for ML. Here we are now dealing with the more formidable task of how to represent a molecule, which has both chemical composition and structure. However, all of the previous criterion still apply.

One might naively consider using the Cartesian coordinates of the molecule as an input into our machine learning model. However, this is very inefficient as the model must learn that the properties of the molecule are independent

of translation, rotation and ordering. Instead, if $\mathbf{r}_i$ denotes the position of the i^{th} atom, then most representations of molecules for ML are made up of inter-atomic distances

$$r_{ij} = |\mathbf{r}_i - \mathbf{r}_j|, \tag{6.1}$$

and inter-atomic angles

$$\theta_{ijk} = \arccos\left[\frac{(\mathbf{r}_i - \mathbf{r}_j)\cdot(\mathbf{r}_k - \mathbf{r}_j)}{r_{ij}r_{kj}}\right], \tag{6.2}$$

since these obviously do not change under translations, $\mathbf{r}_i \to \mathbf{r}_i + \mathbf{a}$, or rotations, $\mathbf{r}_i \to P\mathbf{r}_i$, where $P^T = P^{-1}$.

Many of the representations for molecules find a descriptor for the entire molecule [78, 112]. For example the Coulomb Matrix [113] uses the matrix

$$M_{ij} = \begin{cases} Z_i Z_j / r_{ij} & \text{if} \quad i \neq j \\ 0.5 Z_i^{2.4} & \text{if} \quad i = j \end{cases}. \tag{6.3}$$

Here the off diagonal terms represent the Coulomb repulsion betweens atoms, while the diagonal terms are a polynomial fit of atomic energies to nuclear charge. An issue with the Coulomb Matrix is that it is dependant on a particular ordering of the atoms in the molecule, so for a molecule with N atoms there are a maximum of $N!$ different possible Coulomb Matrices. Each of these different Coulomb Matrices are related to each other via a transformation of the form $M' = A^{-1}MA$. It is easy to show that the set of eigenvalues of M is invariant under such a transformation. Thus the metric $d(M, M') = |\epsilon - \epsilon'|^2$ where ϵ are the ordered eigenvalues can be used to compare molecules. This metric can then be inserted in a kernel ridge regression model (see section 1.3.2) to learn some target property.

The main issue with global descriptors is that they cannot extrapolate to new molecules, which have a similar chemistry but different global structure. In order to avoid this, we would like the ML model to learn local correlations, which capture information about the target property. A way to do this was found by Behler and Parrinello [114, 115]. Suppose our target quantity is y, for example the energy of the molecule, then we write

$$y = \sum_i^N f_{Z_i}(d_i) \tag{6.4}$$

where N is the number of atoms in the molecule, d_i is the local descriptor for atom i and f_{Z_i} is the neural network for element Z_i. Here there is a neural network for each atomic species present in the dataset. Not only does this local descriptor have the advantage of being able to generalize to new molecules,

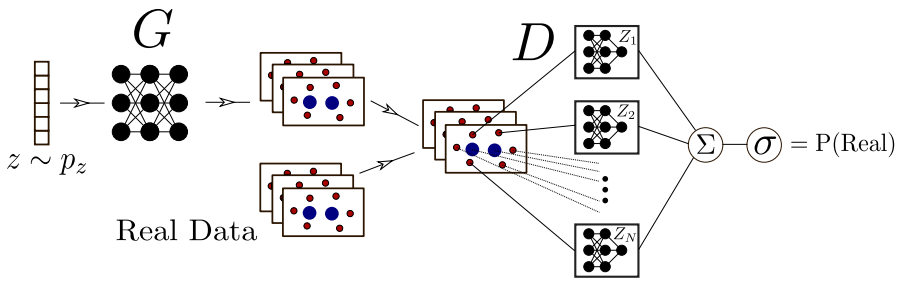


Figure 6.1: The architecture of GANdalf, our GAN for molecular distributions. A latent vector is feed into the generator, which produces a molecule in Cartesian coordinates. The discriminator takes in a given sample and based on the local environment gives the probability of the sample being from the real distribution.

but now each molecule in the dataset contributes a lot more information, as the neural networks learns about all of the atomic environment within the molecule.

6.3 GANdalf

I now introduce our GAN architecture for learning to produce molecules, GANdalf (**G**enerative **A**dversarial **N**etworks for the **D**iscovery of **A**tomic **L**andscapes and **F**ormation of new molecules) is shown in Figure 6.1.

The generator is a fully connected neural network, which takes in a latent vector of some dimension and outputs a molecule in Cartesian coordinates. In practice the choice of normal, $N(\mu = 0, \sigma^2 = 1)$, or uniform, $U[-1, 1]$, latent distribution did not have any significant effect. In the final step of the generator we compute the center of mass of the generated molecule, $\mathbf{m} = \frac{1}{N}\mathbf{r}^{(i)}$, where $\mathbf{r}^{(i)}$ is the coordinate of atom i and centre it at the origin by making the transformation $\mathbf{r}^{(i)} \rightarrow \mathbf{r}^{(i)} - \mathbf{m}$. This makes it easier for the generator to learn to produce physical molecules, as it encodes translational symmetry. Throughout this work the generator had 5 hidden layers, each with 128 nodes, with Leaky ReLU (see Chapter 1) being the activation function of choice, and Adam the optimizer of choice.

For the discriminator, we first compute a descriptor $d^{(i)}$, which describes the local atomic landscape for atom i. Using a neural network with shared weights for each element, we feed $d^{(i)}$ into the network corresponding to the element of atom i, f_{Z_i}. Finally we combine these outputs into a single probability, which

is the probability that a given molecule is from the real distribution

$$P(\text{Real}) = \sigma[\sum_{i=1}^{N} f_{Z_i}(d^{(i)})] = \frac{1}{1 + e^{-\sum_{i=1}^{N} f_{Z_i}(d^{(i)})}}. \tag{6.5}$$

Throughout this work the discriminator had 3 hidden layers, each with 128 nodes, again Leaky ReLU and Adam were used.

There is a novel issue that arises here that does not appear when training a ML model to predict some target, which greatly influences the type of descriptor we can use. Early on in the training, the generator might produce molecules which have atoms very far apart. If the descriptor was local with some effective cut-off radius then although the discriminator would predict the molecules as being fake, the generator would not be able to correct the molecules as there would not be a strong enough gradient to bring the atoms back together. The way we choose to deal with this was to use a descriptor that explicitly uses features based on nearest neighbours. Concretely for each atom in the molecule we use features made up of:

- r_i^{α}: the distance of the i^{th} nearest atom of species α
- $\theta_{ij}^{\alpha\beta}$: the distance between the i^{th} nearest atom of species α and the j^{th} nearest atom of species β

Note that while they are local, they also will not fall to zero if the Generator puts molecules far apart. We leave it to empirical experiments performed in the next chapter to discover the best set comprised of the two features.

6.4 Learning Thermal Distributions

For the first half of our experiments the data was taken from the 'MD Trajectories of small molecules' dataset [116–118]. This dataset is comprised of molecular dynamics trajectories at 500 K for the following molecules:

Formula	Name	Number of atoms
C_2H_5OH	Ethanol	9
C_6H_6	Benzene	12
C_7H_8	Toluene	15
$C_7H_6O_3$	Salicylic Acid	16
$C_{10}H_8$	Naphthalene	18
$C_9H_8O_4$	Aspirin	21

In molecular dynamics the molecule is initialized with velocities drawn from the Maxwell-Boltzmann distribution. The equations of motion are then integrated

to solve for the trajectory of the molecule over time. At each time step Density Functional Theory is used to find the forces on each atom. In this case a temperature of 500 K was used for the initial velocities. For each molecule we randomly chose 1000 samples for our dataset.

We begin with the simplest molecule in the dataset, Benzene. Our first question is whether we need to explicitly include chemical information into the descriptor or is this superfluous. We try two different representations, the first contains no chemical information and represents the Carbon and Hydrogen atomic environments by a vector

$$d^{\mathrm{C6H6}} = (r_1, ..., r_6, \theta_{12}, \theta_{13}, \theta_{32}), \tag{6.6}$$

where we have truncated the number of nearest neighbours at 6. The second does take into account the chemical information and uses different representations for the Carbon and Hydrogen environments

$$\begin{aligned} d_C^{\mathrm{C6H6}} &= (r_1^C, r_2^C, r_3^C, r_4^C, r_1^H, r_2^H, r_3^H, \theta_{12}^{CC}) \\ d_H^{\mathrm{C6H6}} &= (r_1^C, r_2^C, r_3^C, r_1^H, r_2^H, \theta_{12}^{HH}) \end{aligned} \tag{6.7}$$

where d_C^{C6H6} is for Carbon atoms and d_H^{C6H6} is for Hydrogen atoms. Note that the descriptor that contains no chemical information is one component larger and contains more angular information. As Figure 6.2 shows, when we do not include chemical information explicitly the generator is not able to produce realistic examples and its error rises while the discriminator error falls. In contrast, when we include chemical information we reach the Nash equilibrium where the discriminator outputs 1/2 for all samples and the cost functions converge to $\ln(2) \approx 0.69$. Examining the outputs from each generator we can see that when we do not include chemical information the generator seems to get trapped in a local minimum, which is unphysical. Thus from this experiment we conclude that chemical information is necessary.

To see how realistic the range of generated samples are we outputted 1000 samples and compared the resulting bond distribution of the generated data to the real data, as shown in Figure 6.3. We note that while the generator is biased and does not reproduce the real distribution exactly, it always produces molecules with realistic geometries.

As Toluene (C_7H_8) and Naphthalene ($C_{10}H_8$) like Benzene only contain Carbon and Hydrogen we do not have to significantly change the descriptor for them. We did find that for Naphthalene to reach Nash equilibrium we had to include an extra Carbon-Carbon distance, while Toluene reached it with the same descriptor set as Benzene. Figure 6.4 shows a comparison of a generated

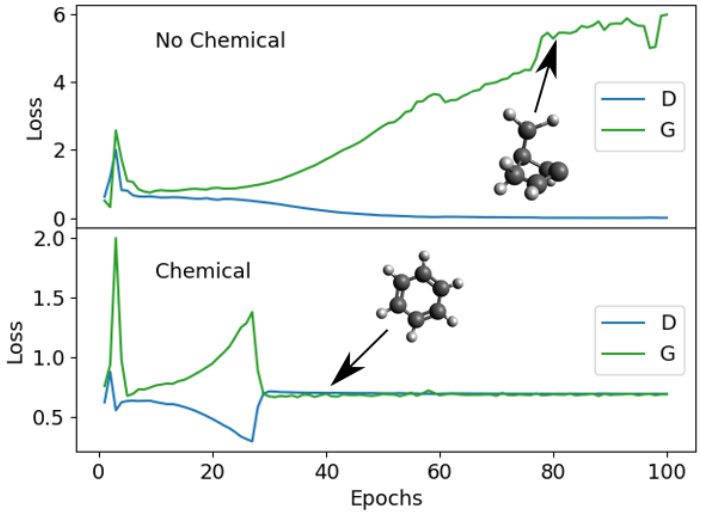


Figure 6.2: Here we plot the cost functions for two GANs constructed with two molecular representations where the chemical information is not included (top) and is included (bottom) in the feature vector. G is the generators cost function and D is cost function of the discriminator. Note that only when we explicitly include chemical information we do reach Nash equilibrium. Inside is shown an example of a generated molecule after 80 epochs for no chemical information feature vector and 40 epochs for the chemical information included feature vector.

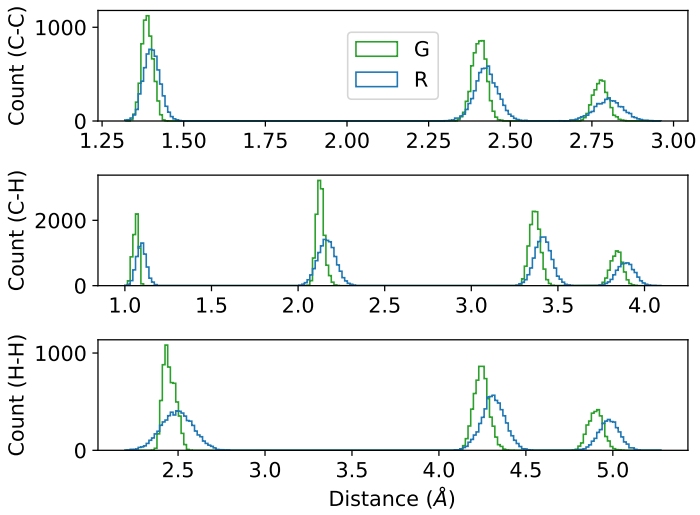


Figure 6.3: The generated (G) and real (R) bond distributions for Benzene. From top to bottom is the Carbon-Carbon, Carbon-Hydrogen and Hydrogen-Hydrogen distributions. Here we are including chemical information in the descriptor.

sample of each to a molecule in the training set. As we can see the generator has learnt to create very realistic molecules for both.

As a final test we take on Aspirin ($C_9H_8O_4$). This offers a much more difficult task than the previous molecules for two reasons. Firstly, it contains an extra element, oxygen, and secondly there is little symmetry in the molecule with several environments for each element. We found it necessary to use the

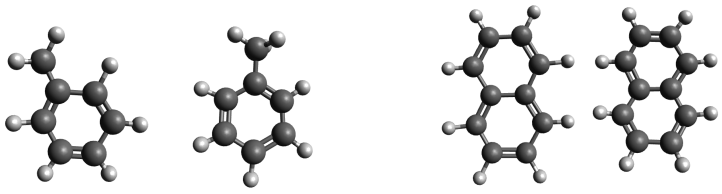
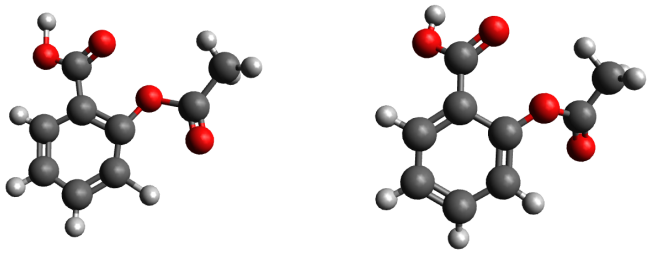


Figure 6.4: Here we compare a generated molecule to a real one for Toluene (left) and Naphthalene (right).



Generated Real

Figure 6.5: Here we compare a generated molecule of Aspirin (left) to one from the dataset (right).

following set of descriptors

$$\begin{aligned} d_{\mathrm{O}}^{\mathrm{C9H8O4}} &= (r_1^O, r_1^C, ..., r_4^C, r_1^H, r_2^H, \beta_{12}^{CC}) \\ d_{\mathrm{C}}^{\mathrm{C9H8O4}} &= (r_1^O, r_1^C, ..., r_5^C, r_1^H, r_2^H, \beta_{12}^{HH}, \beta_{11}^{HC}) \\ d_{\mathrm{H}}^{\mathrm{C9H8O4}} &= (r_1^O, r_1^C, r_2^C, r_3^C, r_1^H, r_2^H, \beta_{12}^{HH}, \beta_{11}^{CC}) \end{aligned} \tag{6.8}$$

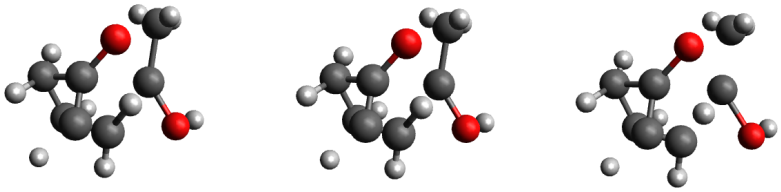
where $d_{\mathrm{X}}^{\mathrm{C9H8O4}}$ is for atoms of specie X. In Figure 6.5 we compare a real Aspirin from the training set to a generated one from the GAN with the two being almost indistinguishable.

6.5 Multiple Molecules And Mode Collapse

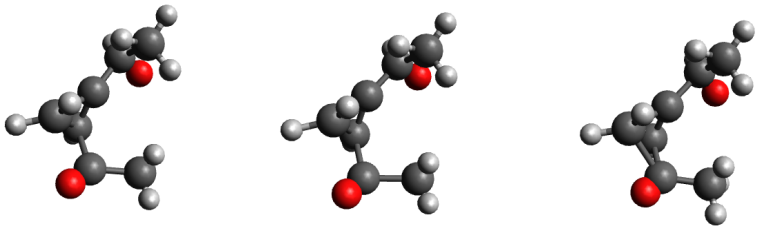
The last section showed that GANdalf can learn to produce individual molecules, however for practical purposes we would need it to be able to produce a diverse range of molecules. Thus we now use a more varied dataset, the QM9 dataset [119, 120], which comprises of all the 6,095 stable configurations of the $C_7O_2H_{10}$ molecule as calculated by Density Functional Theory.

Using 2000 randomly chosen isomers we create a training set for our GAN, which we train as before. Figure 6.6 shows a collection of outputs from the model after 100 and 200 epochs of training. As we can see the generated molecules at each epoch are nearly identical. This is a known problem for GANs and is called mode collapse. At mode collapse the generator losses all variability in its distribution. The distribution does change over time, since the discriminator updates its knowledge of the generator output, but the variability never returns.

One way of measuring when mode collapse takes place is to monitor the



After 100 Epochs



After 200 Epochs

Figure 6.6: Generated samples outputted from the network after 100 (top) and 200 (bottom) training epochs.

distribution of some quantity over time. Here we use the quantity

$$V = \frac{1}{N}\sum_{i=1}^{N} |\mathbf{r}^{(i)} - \mathbf{m}| \tag{6.9}$$

where $\mathbf{m}$ is the centre of mass of the molecule, $\mathbf{m} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{r}^{(i)}$. We call V the molecular volume as it essentially measures how disperse the molecule is [1], although it is not an actual volume as it has units of length. In Figure 6.7 we plot the distributions of the molecular volume for the generated samples at different training times. As we can see from the figure mode collapse arises in the generator very early, before it is able to produce realistic molecules and thus this is a catastrophic problem.

A method of avoiding mode collapse suggested in the literature [121] is to employ batch normalization in the generator and discriminator [122]. Here before the activation function is applied the outputs are normalization to have zero batch mean and a standard deviation of 1. In our case when batch normalization was used in the discriminator the generator was unable to learn. This is probably due to the fact that unlike image tasks, here the features are

[1] Note that it is similar to the radius of gyration, R_g, defined as $R_g^2 = \frac{1}{N}\sum_{i=1}^{N} |\mathbf{r}^{(i)} - \mathbf{m}|^2$.

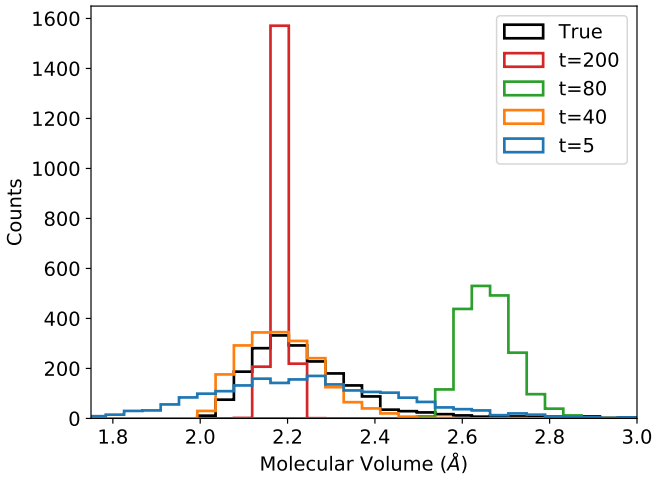


Figure 6.7: The distribution of the molecular volume [as defined in Equation (6.9)] is plotted for the generator at several training times and for the true data. We can see that initially the generator has a variance similar to the true set, however after only 40 epochs we see this variance diminish with mode collapse appearing.

unbounded. When present only in the generator mode collapse still happened however.

An intuitive way to avoid mode collapse which we initially tried is to introduce a term that penalizes the generator when it produces a distribution with low variance. However this comes with its own issues. For example the generator could make two types of molecules, which are very different, thus giving a high variance but not solving the problem. Additionally one has to make sure that the variance penalty is bounded, else the generator can minimize its cost function by making extremely unphysical molecules that are very far apart. In practice we never found good results using this method.

The most successful way we found to combat mode collapse - which is also one of the most popular methods in the literature - requires modifying the GAN itself to the so called Wasserstein GAN (WGAN) [123]. Recall that when the Discriminator is trained optimally its cost function is the Jensen-Shannon divergence. While the Jensen-Shannon divergence is always well defined for any two probability distributions, as shown by Arjovsky et al. [123] it is not a sensible metric for learning distributions. Instead they recommend using the so-called Earth-Mover distance, which can be implemented via an approximation

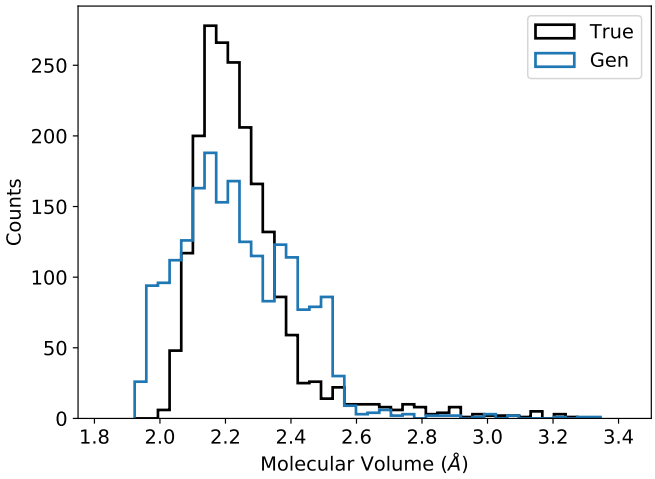


Figure 6.8: The distribution of the molecular volume (as defined in Equation 6.9) is plotted for the WGAN generator after 400 epochs and the true distribution.

using the following cost function for the discriminator

$$J_D = -(\mathrm{E}_{X\sim p_{\mathrm{data}}}[D(X)] - \mathrm{E}_{X\sim p_{\mathrm{gen}}}[D(X)]), \tag{6.10}$$

and the following cost function for the generator

$$J_G = -\mathrm{E}_{X\sim p_{\mathrm{gen}}}[D(X)]. \tag{6.11}$$

Here the Discriminator's output is now unbounded with it 'trying' to output large positive values for the real data and large negative values for the generated data, while the generator 'wants' to close the gap between them. WGANs introduce a clipping parameter, we found best results when this was set to 0.1. Additionally, as per recommendation, the optimizer was changed to RMSProp.

Figure 6.8 shows that the WGAN generator produces molecular volumes with the same range and variance as the true data even after 400 epochs of training, thus demonstrating that we do not have catastrophic mode collapse, unlike with the regular GAN. Figure 6.9 shows a collection of the generated molecules all of which are distinct. However the outputted molecules are not perfect, the most common problems being: a stray Hydrogen atom not attached to anything or the Carbon atoms missing one valence bond. It is likely that better results will be achieved by using deeper networks and training for longer times.

One surprising difference between GANs and WGANs we found is the structure of the latent space. If we have a latent vector of dimension 2 we can visualize

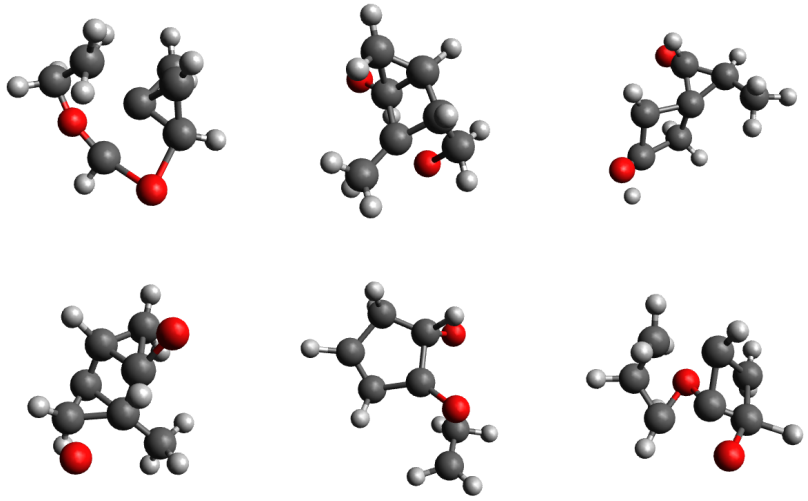


Figure 6.9: Here we show a collection of samples from the WGAN generator after 400 epochs of training on the QM9 dataset.

the latent space by plotting a scalar quantity of the resulting molecules over latent space. Here we use ANI [124] - a neural network trained on DFT energies - to predict the energy of the generated molecules. In Figure 6.10 we show the results for a GAN and a WGAN trained on the thermal distributions of methane. As can be seen the latent space of the WGAN is much more symmetric than the GANs latent space, with the high energy configurations of Benzene occurring at the corners of the latent space.

6.6 E-GANdalf

We can easily modify the GANdalf framework for the discovery of molecules with certain properties. Suppose we have a neural network (or any differentiable function) $f(X)$, which predicts some property given a molecule. We can generate molecules that minimise [2] this property by altering the WGAN generator cost function to

$$J_G = -\mathrm{E}_{X\sim p_{\mathrm{gen}}}[D(X)] + \lambda \mathrm{E}_{X\sim p_{\mathrm{gen}}}[f(X)] \tag{6.12}$$

where λ is a parameter which controls the trade-off between fooling the discriminator and minimizing the desired function. A Nash equilibrium occurs when the generator is able to produce a set of samples that fool the discriminator

[2] If we wish to maximise the property we can use $-f(X)$.

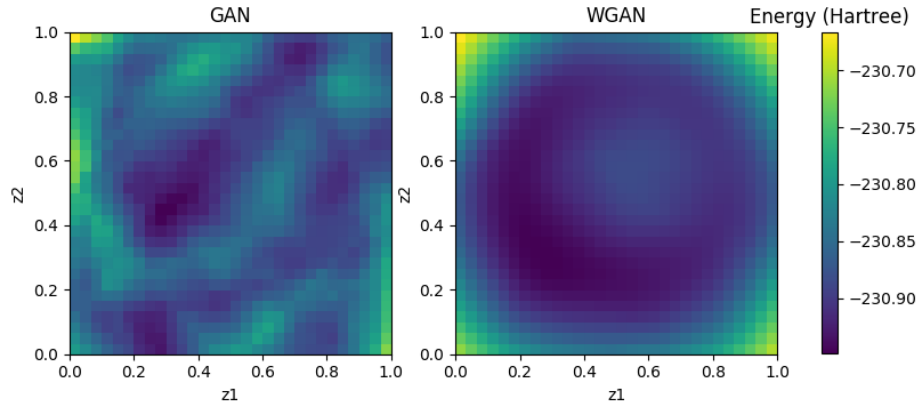


Figure 6.10: Here we show the predict energies with ANI against latent space for both the GAN and WGAN for the methane thermal data. $z1$ and $z2$ are the two components of the latent vector $\mathbf{z}$. The dimension of the latent space was 2 with a uniform distribution used to generate the latent space points. The energies are in units of Hartrees.

and minimize the function. This idea was found to produce good results by Guimaraes et al. [125].

Here we opt to minimize the energy of the molecules calling the new framework E-GANdalf. To predict the energies of the molecules we once again use ANI. Using the energy as the property to minimize will also have the effect of penalizing high energy generated molecules making the outputted molecules more realistic.

Figure 6.11 shows how the cost functions and mean energy vary over epochs for various values of λ. It is disappointing to see that the relationship between λ and these quantities is very complicated as we cannot infer any causal relationship. However it does seem that λ has little effect over the mean energy of the sample which is very surprising. One explanation is that ANI may only be accurate for molecules that are near the equilibrium and thus would not provide suitable feedback for the generator.

6.7 Summary

In this chapter I have introduce GANdalf, a framework based on generative adversarial networks which is designed to produce 3-dimensional molecules. Unlike most approaches to generating molecules using AI, GANdalf outputs the molecules in Cartesian coordinates based only on local environments.

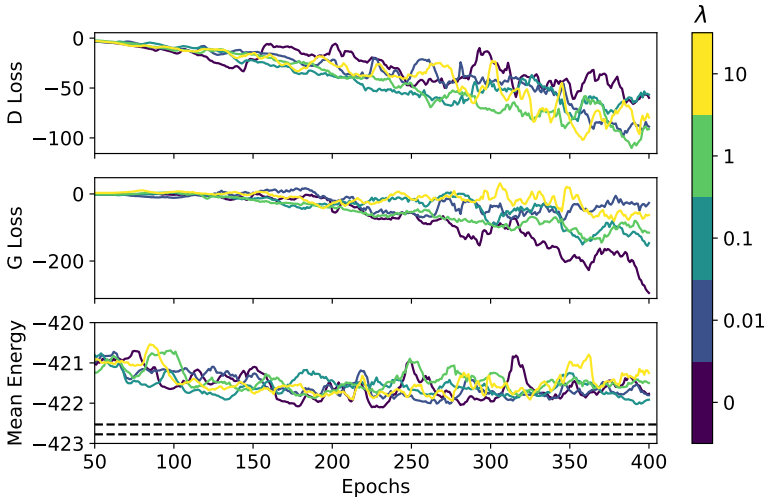


Figure 6.11: Here we show how the cost functions for the discriminator (D) and generator (G), and the mean energy, vary over training epochs. The different colors correspond to different values of λ. In the bottom panel the dashed lines correspond to the minimum and maximum energies in the training set as computed by ANI. The energies are in units of Hartrees.

We have shown that GANdalf is capable of producing individual molecules accurately and when trained on a diverse set is able to avoid mode collapse. However, there is still much work to be done for GANdalf to actually be able to suggest new molecules for a given application. Firstly, the most obvious place for improvement is the descriptor, which most likely does not describe the local environments well enough for GANdalf to learn to a high degree of accuracy. Secondly, using deeper networks should improve the quality of the produced molecules, although this will require more computational resources. Thirdly, to produce molecules with a given property, a neural network would have to be trained first to predict this property, this would then be implemented in the GANdalf framework as shown in the last chapter.

To have AI be able to suggest new molecules is an ambitious task. While we most likely already have sufficient data to train the model, the difficulty arises in choosing an architecture capable of learning the information. GANdalf is a proof of concept that it is possible to generate molecules based on local atomic environments.

Chapter 7

Summary and Future Work

> "The capacity to reason is a special sort of capacity because it leads us to places we did not expect to go ... Beginning to reason is like stepping onto an escalator that leads us upwards and out of sight"
>
> - Peter Singer, The Expanding Circle: Ethics, Evolution, and Moral Progress

In this work I have explored the application of machine learning techniques to condensed matter physics. I now proceed to review the main results found and suggest possible directions for future work.

Chapter 3 explored using ML to construct density functional theory schemes for the Hubbard model. By using the electron density as the primary variable, we avoid the exponential scaling of the wavefunction with system size. We found that we can accurately map the ground-state electron density to the universal function, with this mapping obeying the Hohenberg-Kohn theorems. Further, we showed that using a local approximation allows one to train on small lattices, which are cheap to solve, and then generalize to larger lattices. Finally we showed that we are not limited to predicting ground-state properties, we can also access finite temperature properties.

I see three avenues for future work, firstly, replicating the results in two dimensions. The physics of the Hubbard model is much more interesting in two dimensions, with phenomena like superconductivity occurring [126]. If the local approximation can perform accurately here, this would be a powerful method to investigate exotic properties of the Hubbard model in the large site limit. A second direction, would be to try to predict other quantities like the density-density correlation matrix or bipartite entropy (which measures the entanglement in the

system), as these quantities can indicate the presence of new phases [1]. It would be interesting to see if one could build a phase classifier, which although trained on small lattices, was accurate for large systems. The final idea, would be to try to generalize from small lattices using the Hubbard model with spin. Then, the question is whether one could correctly rank the magnetization energies of larger lattices. This might turn out to be a difficult problem, since large systems have magnetizations that do not exist in smaller systems.

Chapter 4 was an investigation into using machine learning to propagate quantum systems in time. We found that when using the wavefunction or the density matrix to represent the system, learning the propagator is trivial using linear regression. The number of samples required is linear with the size of the Hilbert space for the wavefunction and quadratic for the density matrix. The advantage of using machine learning here is that we could learn the propagator for a large time step and thus efficiently propagate the system far into the future. When we used a compressed representation of the system, to avoid the aforementioned exponentially scaling, the dynamics become non-Markovian with a history of past states being required to predict the future evolution. This history scales exponentially with the number of qubits, however one can achieve a better scaling by choosing appropriately the initial state or we conjecture, the hamiltonian. We investigated the properties of this non-Markovian evolution and found that there was no limit to the size of the time step possible and that the evolution was localized.

One direction of future research would be to see if we can make the non-Markovian systems more Markovian. If we just concern ourselves with a single qubit, one way of doing this would be to weaken the coupling between that qubit and the rest, or to add a noisy magnetic time-dependent field to the other qubits. Another direction, would be to investigate whether the scaling of the memory can be lessened, by the choice of the hamiltonian. Our findings indicated that the memory scales with the number of frequencies, suggesting that this might be possible. If either of these avenues were promising, it would make using the compressed representation more attractive, as less memory would be required.

In Chapter 5 I used machine learning to predict the Curie temperature of ferromagnets. This was achieved by first creating a dataset of experimental Curie temperature and then using a suitable descriptor to represent the material. Our final model, which only used chemical information, had a mean absolute error of around 50 K. Further, we were able to predict the uncertainty of the model, making the predictions more robust and the model more useful.

[1] For an investigation into predicting the density-density correlation matrix see Ref. [127]

While only using chemical information has the advantage of not needing to know the materials structure to make a prediction, ultimately to improve the accuracy structural information will need to be incorporated. Adding this extra information is a non-trivial task, the mapping to the structure is very likely highly complex, as evidenced by our findings that structure did not improve the error of the subset we had structures for. Thus a very large dataset will need to be constructed. However, there is not a sufficient amount of Curie temperature measurements to build such a dataset. Thus, perhaps, one might have to turn to some ab-initio method for calculating the Curie temperature to build this dataset. A more manageable way to improve the accuracy, would be to monitor published papers and add to the dataset every time a Curie temperature measurement was found. Increasing the size of the dataset is the most straightforward way to improve performance for a machine learning model.

In Chapter 6 I looked at using generative adversarial networks (GANs) to discover new molecules. To do this I treated it as a generative modelling task, where given a dataset of molecules, the GAN was trained to generate new molecules similar to the ones in the dataset. I found that while I could replicate thermal distributions of singular molecules, moving to datasets with multiple molecules was harder, with mode collapse occurring. By using Wasserstin GANs, I was able to restore the variability in the generated samples, but the outputs are still not perfectly chemically accurate.

Future work for this project would mainly consist in trying to improve the results for the multiple molecule datasets. Possible ways of doing this would be to use deeper networks and train for longer. Although, possibly the most important alteration would be to improve the descriptor. The descriptor we used was the simplest choice to avoid the vanishing gradient problem, that occurs early in the training. Using a descriptor that described the local environments in more detail or more compactly, could significantly improve the results. If the previous issues are fixed and the GAN can produce a varied and chemically accurate output, then the next step would be to train another network to predict some quantity. Then as I showed, it is straightforward, in theory, to combine the GAN and predictor network to bias the dataset, for the discovery of novel molecules. Another modification would be to allow the generator to produce molecules of different sizes. This could be done by having the generator output a fixed amount of atoms but also output which atoms would be in the molecule. Atoms that are not in the molecule would then be simply ignored. While this would increase the complexity of the training, it would not be very computationally expensive.

Appendices

Appendix A

Second Quantization

For more detail see Ref. [22].

A.1 Indistinguishable particles

In quantum physics there is a very important distinction between distinguishable and indistinguishable particles. Particles are distinguishable if they differ in some measurable quantity like spin or mass.

Suppose we have two particles A and B. If A is in state n and b in state m then the whole system will be described by the ket $|n\rangle|m\rangle = |nm\rangle$. If the two particles are distinguishable then $\langle n'm'|nm\rangle = \delta_{nn'}\delta_{mm'}$ since a particle cannot be in the two different states at the same time. The ket $|nm\rangle$ thus forms a basis and we can write any wavefunction for the system as $|\psi\rangle = \sum_{nm} |nm\rangle\langle nm|\psi\rangle$.

What about if A and B are indistinguishable particles? Now we are unable to tell the difference between kets $|nm\rangle$ and $|mn\rangle$ and thus they must differ by a phase $|nm\rangle = e^{i\alpha}|mn\rangle$. Applying the same reasoning again we find $(e^{i\alpha})^2 = 1$ and thus

$$|nm\rangle = \pm|mn\rangle. \tag{A.1}$$

Particles which obey the plus sign rule are called bosons, while the minus sign rule corresponds to fermions. From this we can see that for fermions the ket $|nn\rangle$ is zero, a result that encodes the Pauli exclusion principle, two fermions cannot have the same quantum numbers. It also implies that exchanging any two particles always results in a minus sign for fermions as there is always an odd number of swaps to trade the particles [1].

[1] For example suppose we swap particles i and j with $j > i$. To get i next to but not after we need $j - i - 1$ swaps. To switch i and j we need 1 swap and to bring j to i's original place

It is straightforward to check that we now have

$$\langle n'm'|nm\rangle = \delta_{nn'}\delta_{mm'} \pm \delta_{m'n}\delta_{n'mm} \tag{A.2}$$

with the convention that for bosons $\langle nn|nn\rangle = 2$. Hence our basis is the set $\{|n\rangle|m\rangle\}$ with $n \geq m$, which gives

$$\sum_{n>m} |nm\rangle\langle nm| + \frac{1}{2}\sum_{n} |nn\rangle\langle nn| = \frac{1}{2}\sum_{nm} |nm\rangle\langle nm| = \mathbb{1}. \tag{A.3}$$

A.2 The Second Quantization Formalism

For many-particle systems it is convenient to introduce creation operators $\hat{\psi}^\dagger(\mathbf{x})$ with

$$\begin{aligned} |\mathbf{x}\rangle &= \hat{\psi}^\dagger(\mathbf{x})|0\rangle \\ |\mathbf{x}_1\mathbf{x}_2\rangle &= \hat{\psi}^\dagger(\mathbf{x}_2)|\mathbf{x}_1\rangle = \hat{\psi}^\dagger(\mathbf{x}_2)\hat{\psi}^\dagger(\mathbf{x}_1)|0\rangle \end{aligned} \tag{A.4}$$

where $\mathbf{x}$ represents a particle at position $\mathbf{r}$ with spin σ, $\mathbf{x} = (\mathbf{r}, \sigma)$. Since $|\mathbf{x}_1\mathbf{x}_2\rangle = \pm|\mathbf{x}_2\mathbf{x}_1\rangle$ we must have

$$[\hat{\psi}^\dagger(\mathbf{x}), \hat{\psi}^\dagger(\mathbf{y})]_\mp = 0 \tag{A.5}$$

where $[a, b]_\mp = ab \mp ba$. We can take the adjoint of the creation operator and define the annihilation operator, $\hat{\psi}(\mathbf{x})$, with

$$[\hat{\psi}(\mathbf{x}), \hat{\psi}(\mathbf{y})]_\mp = 0. \tag{A.6}$$

Note that $\langle 0|\hat{\psi}^\dagger(\mathbf{x})|\psi\rangle$ is zero for any ket $|\psi\rangle$ as $\hat{\psi}^\dagger(\mathbf{x})|\psi\rangle$ will never be in the zero particle Hilbert space. Similarly $\langle\psi|\hat{\psi}(\mathbf{x})|0\rangle = 0$ which leads to $\hat{\psi}(\mathbf{x})|0\rangle = 0$.

Again since a particle cannot be in two different states at the same time we must have

$$\langle \mathbf{x}|\mathbf{y}\rangle = \delta(\mathbf{x} - \mathbf{y}) \equiv \delta_{\sigma_x\sigma_y}\delta(\mathbf{r}_x - \mathbf{r}_y). \tag{A.7}$$

However, since $\langle\mathbf{x}| = \langle 0|\hat{\psi}(\mathbf{x})$, we conclude that $\langle 0|\hat{\psi}(\mathbf{x})|\mathbf{y}\rangle = \delta(\mathbf{x} - \mathbf{y})$ from which it follows that $\hat{\psi}(\mathbf{x})|\mathbf{y}\rangle = \delta(\mathbf{x} - \mathbf{y})|0\rangle$. Thus we see that the annihilation operator removes a particle.

From before we know that

$$\langle \mathbf{x}_1\mathbf{x}_2|\mathbf{y}_1\mathbf{y}_2\rangle = \delta(\mathbf{x}_1 - \mathbf{y}_1)\delta(\mathbf{x}_2 - \mathbf{y}_2) \pm \delta(\mathbf{x}_1 - \mathbf{y}_2)\delta(\mathbf{x}_2 - \mathbf{y}_1) \tag{A.8}$$

and hence

$$\hat{\psi}(\mathbf{x})|\mathbf{y}_1\mathbf{y}_2\rangle = \delta(\mathbf{x} - \mathbf{y}_2)|\mathbf{y}_1\rangle \pm \delta(\mathbf{x} - \mathbf{y}_1)|\mathbf{y}_2\rangle. \tag{A.9}$$

---

we need another $j - i - 1$ swaps. Thus we require a total of $2(j - i) - 1$ swaps which is an odd number.

In general for N particles this becomes

$$\hat{\psi}(\mathbf{x})|\mathbf{y}_1...\mathbf{y}_N\rangle = \sum_{i=1}^{N}(\pm 1)^{N+i}\delta(\mathbf{x}-\mathbf{y}_i)|\mathbf{y}_1...\mathbf{y}_{i-1}\mathbf{y}_{i+1}...\mathbf{y}_N\rangle. \tag{A.10}$$

It follows from this last relation that

$$\hat{\psi}(\mathbf{x})\hat{\psi}^\dagger(\mathbf{y})|\mathbf{y}_1...\mathbf{y}_N\rangle = \delta(\mathbf{x}-\mathbf{y})|\mathbf{y}_1...\mathbf{y}_N\rangle \pm \hat{\psi}^\dagger(\mathbf{y})\hat{\psi}(\mathbf{x})|\mathbf{y}_1...\mathbf{y}_N\rangle \tag{A.11}$$

from which we get the important result

$$[\hat{\psi}(\mathbf{x}), \hat{\psi}^\dagger(\mathbf{y})]_\mp = \delta(\mathbf{x}-\mathbf{y}). \tag{A.12}$$

A useful definition is the density operator

$$\hat{n}(\mathbf{x}) = \hat{\psi}^\dagger(\mathbf{x})\hat{\psi}(\mathbf{x}). \tag{A.13}$$

The density operator satisfies $[\hat{n}(\mathbf{x}), \hat{\psi}(\mathbf{y})] = -\delta(\mathbf{x}-\mathbf{y})\hat{\psi}(\mathbf{x})$ and $[\hat{n}(\mathbf{x}), \hat{\psi}^\dagger(\mathbf{y})] = \delta(\mathbf{x}-\mathbf{y})\hat{\psi}^\dagger(\mathbf{x})$ and hence it acts on ket $|\mathbf{x}_1\mathbf{x}_2...\mathbf{x}_N\rangle$ as follows

$$\hat{n}(\mathbf{x})|\mathbf{x}_1\mathbf{x}_2...\mathbf{x}_N\rangle = \sum_{i=1}^{N}\delta(\mathbf{x}-\mathbf{x}_i)|\mathbf{x}_1\mathbf{x}_2...\mathbf{x}_N\rangle. \tag{A.14}$$

Finally, suppose we have a one particle ket $|n\rangle$, it is straightforward to check that $|n\rangle = \hat{c}_n^\dagger|0\rangle$ where

$$\hat{c}_n^\dagger = \int d\mathbf{x}\phi_n(\mathbf{x})\hat{\psi}^\dagger(\mathbf{x}) \tag{A.15}$$

with $\phi_n(\mathbf{x}) = \langle\mathbf{x}|n\rangle$, $\int d\mathbf{x}|\phi_n(\mathbf{x})|^2 = 1$ and the $\hat{c}_n^\dagger$ operators satisfying

$$[\hat{c}_n^\dagger, \hat{c}_m^\dagger]_\mp = [\hat{c}_n, \hat{c}_m]_\mp = 0 \quad , \quad [\hat{c}_n, \hat{c}_m^\dagger]_\mp = \delta_{nm}. \tag{A.16}$$

Appendix B

The Heisenberg Model as a limit of the Hubbard Model

Here we follow Essler et al. [23] for an alternate derivation see Fazekas [24]. The derivation consists in first calculating the second order degenerate perturbation expansion and then applying this to the Hubbard model.

B.1 Degenerate Perturbation Theory

Consider a Hamiltonian

$$\hat{H} = \hat{H}_0 + \lambda \hat{V}, \tag{B.1}$$

where we view $\hat{V}$ as a perturbation of $\hat{H}_0$, with λ controlling the strength of the perturbation. Here $\hat{H}_0$ may have degenerate energy eigenvalues. We can use a spectral decomposition to write

$$\hat{H}_0 = \sum_n E_n \hat{P}_n \tag{B.2}$$

where $\hat{P}_n$ is the projector for energy eigenstates E_n. $\hat{P}_n$ satisfies $\hat{P}_n^2 = \hat{P}_n$ and $\hat{P}_n \hat{P}_m = \delta_{mn} \hat{P}_n$. We also define the projector complement $\hat{Q}_n = \mathbb{1} - \hat{P}_n$.

If the full Hamiltonian has eigenvalue E with eigenvector $|\psi\rangle$,

$$\hat{H}|\psi\rangle = E|\psi\rangle, \tag{B.3}$$

then inserting $\hat{P}_n + \hat{Q}_n = \mathbb{1}$ into this and using the fact that $\hat{P}_n \hat{Q}_n = \hat{Q}_n \hat{P}_n = 0$ we obtain

$$\begin{aligned} \hat{P}_n \hat{H} \hat{P}_n |\psi\rangle + \hat{P}_n \hat{H} \hat{Q}_n |\psi\rangle &= E \hat{P}_n |\psi\rangle, \\ \hat{Q}_n \hat{H} \hat{P}_n |\psi\rangle + \hat{Q}_n \hat{H} \hat{Q}_n |\psi\rangle &= E \hat{Q}_n |\psi\rangle. \end{aligned} \tag{B.4}$$

From this it follows

$$\hat{Q}_n|\psi\rangle = (E - \hat{Q}_n\hat{H})^{-1}\hat{Q}_n\hat{H}\hat{P}_n|\psi\rangle \tag{B.5}$$

and thus

$$\hat{H}_n(E)|\phi\rangle = E|\phi\rangle \tag{B.6}$$

where

$$\hat{H}_n(E) = \hat{P}_n\hat{H}\left[\mathbb{1} + (E - \hat{Q}_n\hat{H})^{-1}\hat{Q}_n\hat{H}\right]\hat{P}_n \tag{B.7}$$

and $|\phi\rangle = \hat{P}_n|\psi\rangle$. Thus $|\phi\rangle$ is the projection of $|\psi\rangle$ onto the Hilbert space with E_n energy eigenvalue. Then, on this space we can solve the eigenvalue equation (B.6) to compute the energy eigenvalue of the original problem. We can obtain the original eigenvector using Equation (B.5) with

$$|\psi\rangle = \left[\mathbb{1} + (E - \hat{Q}_n\hat{H})^{-1}\hat{Q}_n\hat{H}\right]|\phi\rangle. \tag{B.8}$$

The next step is to substitute $\hat{H} = \hat{H}_0 + \lambda\hat{V}$ into Equation (B.7). First we note that

$$\begin{aligned} E - \hat{Q}_n\hat{H} &= E - \lambda\hat{Q}_n\hat{V} - \hat{Q}_n\hat{H}_0 \\ &= \sum_m (E - E_m)\hat{P}_m - (\lambda\hat{Q}_n\hat{V} - E_n\hat{P}_n), \end{aligned} \tag{B.9}$$

and using the fact that

$$(B - A)^{-1} = (\mathbb{1} - B^{-1}A)^{-1}B^{-1} = \sum_{i=0}^{\infty}(B^{-1}A)^i B^{-1}, \tag{B.10}$$

we write

$$(E - \hat{Q}_n\hat{H})^{-1} = \sum_{i=0}^{\infty}\left(\sum_{m\neq n}\frac{\lambda\hat{P}_m\hat{V}}{E - E_m} - \frac{E_n\hat{P}_n}{E - E_n}\right)^i \sum_m \frac{\hat{P}_m}{E - E_m}. \tag{B.11}$$

By inserting this into Equation (B.7) and using the fact that $\hat{P}_n\hat{P}_m = \delta_{nm}\hat{P}_n$ allows us to write

$$\hat{H}_n(E) = \left[E_n + \hat{P}_n\hat{V}\sum_{i=0}^{\infty}\lambda^{i+1}\left(\sum_{m\neq n}\frac{\hat{P}_m\hat{V}}{E - E_m}\right)^i\right]\hat{P}_n. \tag{B.12}$$

Thus, we obtain the eigenvalue equation

$$\hat{P}_n\hat{V}\sum_{i=0}^{\infty}\lambda^{i+1}\left(\sum_{m\neq n}\frac{\hat{P}_m\hat{V}}{E - E_m}\right)^i|\phi\rangle = (E - E_n)|\phi\rangle. \tag{B.13}$$

Next we write the energy as an expansion in powers of λ around E_n

$$E = E_n + \lambda E_n^{(1)} + \lambda^2 E_n^{(2)} + ... \tag{B.14}$$

and thus up to quadratic terms in λ we arrive at

$$\left[\lambda \hat{P}_n \hat{V} \hat{P}_n + \lambda^2 \sum_{m \neq n} \frac{\hat{P}_n \hat{V} \hat{P}_m \hat{V} \hat{P}_n}{E_n - E_m} \right] |\phi\rangle = (E - E_n)|\phi\rangle. \tag{B.15}$$

B.2 Hubbard Model in the Large U Limit

Returning to the Hubbard model, we set the onsite terms, v_i, to zero and write the Hamiltonian as

$$\hat{H}/U = \hat{T}/U + \hat{D} \tag{B.16}$$

where

$$\hat{D} = \sum_{j=1}^{L} \hat{n}_{j\uparrow} \hat{n}_{j\downarrow} \tag{B.17}$$

and

$$\hat{T} = -t \sum_{\langle i,j \rangle} \sum_{\sigma} \hat{c}_{i\sigma}^{\dagger} \hat{c}_{j\sigma}. \tag{B.18}$$

Given a basis ket $|\mathbf{x}, \mathbf{a}\rangle$ [Equation 2.27] the $\hat{D}$ operator simply counts the number of doubly occupied sites with

$$\hat{D}|\mathbf{x}, \mathbf{a}\rangle = d|\mathbf{x}, \mathbf{a}\rangle \tag{B.19}$$

where d is the number of doubly occupied sites in $|\mathbf{x}, \mathbf{a}\rangle$. Thus we can write

$$\hat{D} = \sum_{d=0}^{L} d\hat{P}_d \tag{B.20}$$

where $\hat{P}_d$ is the projector onto the Hilbert space with d doubly occupied sites with

$$\hat{P}_d|\mathbf{x}, \mathbf{a}\rangle = \begin{cases} |\mathbf{x}, \mathbf{a}\rangle & \text{if } |\mathbf{x}, \mathbf{a}\rangle \text{ has } d \text{ doubly occupied sites} \\ 0 & \text{otherwise} \end{cases}. \tag{B.21}$$

We then substitute $\hat{T}$ into Equation (B.15) where now $\lambda = 1/U$, $E_n = n$ and we have replaced E by E/U, getting

$$\left[\frac{1}{U} \hat{P}_n \hat{T} \hat{P}_n + \frac{1}{U^2} \sum_{m \neq n} \frac{\hat{P}_n \hat{T} \hat{P}_m \hat{T} \hat{P}_n}{n - m} \right] |\phi\rangle = (\frac{E}{U} - n)|\phi\rangle. \tag{B.22}$$

As we are only interested in the ground state perturbation we set $n = 0$

$$\left[\hat{P}_0\hat{T}\hat{P}_0 - \frac{1}{U}\sum_{m\neq n}\frac{\hat{P}_0\hat{T}\hat{P}_m\hat{T}\hat{P}_0}{m}\right]|\phi\rangle = E|\phi\rangle. \tag{B.23}$$

Thus our effective Hamiltonian is

$$\hat{H} = \hat{P}_0\left[\hat{T} - \frac{1}{U}\sum_{m\neq n}\frac{\hat{T}\hat{P}_m\hat{T}}{m}\right]\hat{P}_0. \tag{B.24}$$

In order to derive the Heisenberg model we now specify that the lattice is half filled, with $N = L$. Thus the states $|\phi\rangle$ will have one particle per site, as there are no doubly occupied sites. Suppose we have one such state

$$|\theta\rangle = |..., \sigma_i, \sigma_j, ...\rangle \tag{B.25}$$

where $\sigma_i \in \{\uparrow, \downarrow\}$ and consider the action of $\hat{T}$ on the i and j particles with

$$\hat{T}_{ij} = -t\sum_{\sigma}\left(\hat{c}^{\dagger}_{i\sigma}\hat{c}_{j\sigma} + \hat{c}^{\dagger}_{j\sigma}\hat{c}_{i\sigma}\right) \tag{B.26}$$

and $\hat{T} = \frac{1}{2}\sum_{\langle i,j\rangle}\hat{T}_{ij}$. If $\sigma_i = \sigma_j$ then $\hat{T}_{ij}|\theta\rangle = 0$, otherwise a superposition of doubly occupied sites will be created. Thus $\hat{P}_m\hat{T}\hat{P}_0 = 0$ is only non-zero for $m = 1$ and the Hamiltonian reduces to

$$\hat{H} = -\frac{1}{U}\hat{P}_0\hat{T}\hat{P}_1\hat{T}\hat{P}_0. \tag{B.27}$$

Suppose $\sigma_i \neq \sigma_j$ then

$$\hat{T}_{ij}|\theta\rangle = -t|..., d, 0, ...\rangle - t|..., 0, d, ...\rangle, \tag{B.28}$$

where d indicates a doubly occupied site. As we have one doubly occupied site in each ket, it follows that $\hat{P}_1\hat{T}_{ij}|\theta\rangle = \hat{T}_{ij}|\theta\rangle$. However, when we apply $\hat{P}_0\hat{T}$ to this, the only non-zero terms will be kets with no doubly occupied sites and hence

$$\hat{P}_0\hat{T}\hat{P}_1\hat{T}_{ij}|\theta\rangle = -2t^2\left(|..., \sigma_i, \sigma_j, ...\rangle + |..., \sigma_j, \sigma_i, ...\rangle\right) \tag{B.29}$$

where here one must be careful to obtain the correct sign. We can see that $-4(2t^2)\left(\hat{\mathbf{S}}^{(i)}\cdot\hat{\mathbf{S}}^{(j)} - \frac{1}{4}\right)$ reproduces the behaviour of $\hat{P}_0\hat{T}\hat{P}_1\hat{T}_{ij}$ on $|\theta\rangle$ and thus we can write

$$\hat{H} = -\frac{1}{U}\frac{1}{2}\sum_{\langle i,j\rangle}\hat{P}_0\hat{T}\hat{P}_1\hat{T}_{ij}\hat{P}_0 = -J\sum_{\langle i,j\rangle}\left(\hat{\mathbf{S}}^{(i)}\cdot\hat{\mathbf{S}}^{(j)} - \frac{1}{4}\right), \tag{B.30}$$

which is the Heisenberg model with $J = -4t^2/U$.

Appendix C

Statistical Mechanics

For more details on statistical physics see Sethna [128]. Here we have set $k_B = 1$.

C.1 Microcanonical Ensemble

In the microcanonical ensemble, we consider a closed system with energy eigenstates $|n\rangle$, each with energy E_n. We make the fundamental assumption that when the system is in equilibrium with energy ϵ_0 all eigenstates $|n\rangle$ with $E_n = \epsilon_0$ are equally likely. If $\Omega(E)$ is the number of eigenstates at energy E, then the probability of the system being in state $|n\rangle$ is

$$p(n) = \begin{cases} 1/\Omega(\epsilon_0) & E_n = \epsilon_0 \\ 0 & E_n \neq \epsilon_0. \end{cases}$$

From this we can work out the expected value of an operator $\hat{O}$ using

$$\langle \hat{O} \rangle = \sum_n p(n) \langle n|\hat{O}|n\rangle. \tag{C.1}$$

An important quantity is the thermodynamic entropy defined as

$$S(E) = \log \Omega(E). \tag{C.2}$$

Finally we say a system is ergodic if the time averaged value of an operator

$$\langle \hat{O} \rangle_T = \lim_{T\to\infty} \frac{1}{T} \int_0^T dt \langle \psi(t)|\hat{O}|\psi(t)\rangle \tag{C.3}$$

is equal to the microcanonical expectation value.

C.2 Canonical Ensemble

In the canonical ensemble we consider two interacting systems A and B. We assume B is very large compared to A. The number of states of the entire system at energy E is

$$\Omega(E) = \sum_{E_n} \Omega_A(E_n)\Omega_B(E - E_n). \tag{C.4}$$

Since the B system is very large compared to A we can write

$$\Omega_B(E - E_n) = e^{S_B(E-E_n)} \approx e^{S_B(E)-S'_B(E)E_n} = e^{S_B(E)}e^{-\beta E_n} \tag{C.5}$$

where we have associated $S'_B(E)$ with the inverse temperature $\beta = 1/T$. Thus

$$\Omega(E) = e^{S_B(E)} \sum_{E_n} \Omega_A(E_n)e^{-\beta E_n} = e^{S_B(E)} \sum_n e^{-\beta E_n} \tag{C.6}$$

and hence the probability system A is in state n with energy E_n is

$$p_n = \frac{e^{-\beta E_n}}{\sum_s e^{-\beta E_s}} = \frac{e^{-\beta E_n}}{Z} \tag{C.7}$$

where Z is the partition function

$$Z = \sum_n e^{-\beta E_n}. \tag{C.8}$$

One can use Equation C.7 to compute thermal averages, for example quantities of interest include the thermodynamic energy

$$\langle E \rangle = \sum_n p_n E_n = -\frac{\partial}{\partial \beta} \log Z, \tag{C.9}$$

the entropy

$$S = -\sum_n p_n \log p_n = \beta \langle E \rangle + \log Z \tag{C.10}$$

and the heat capacity

$$C = \frac{\partial}{\partial T}\langle E \rangle = \beta^2 \left(\langle E^2 \rangle - \langle E \rangle^2 \right) \tag{C.11}$$

where $\langle E^2 \rangle = \sum_n p_n E_n^2$.

We define the free energy to be

$$H = E - TS = -T \log Z. \tag{C.12}$$

As this implies $Z = e^{-\beta H}$ we see that the free energy contains the same amount of information as the partition function. All of the previous quantities can be

rewritten in terms of the free energy with $\langle E \rangle = \frac{\partial}{\partial \beta}(\beta H)$, $S = \beta^2 \frac{\partial}{\partial \beta} H$ and $C = \frac{\partial}{\partial T}\frac{\partial}{\partial \beta}(\beta H)$.

Finally, we can see that

$$\hat{\rho} = e^{-\beta \hat{H}}/Z \tag{C.13}$$

is the density matrix (see Section 4.2.1 for more details) for the canonical ensemble, as this reproduces the correct thermal averages, with the average value for an operator $\hat{O}$ being $\mathrm{Tr}[\hat{O}\hat{\rho}]$. As well, the entropy is given by, $S = -\mathrm{Tr}(\hat{\rho} \log \hat{\rho})$.

C.3 Grand Canonical Ensemble

Finally for the grand canonical ensemble we proceed similarly to the canonical ensemble, with the exception that the number of particles in A and B is allowed to vary. Thus the probability that system A is in state s with energy E_s and number of particles N_s is

$$p(E_s, N_s) = \frac{e^{-\beta(E_s - \mu N_s)}}{\sum_n e^{-\beta(E_n - \mu N_n)}} = \frac{e^{-\beta(E_s - \mu N_s)}}{\Omega} \tag{C.14}$$

where we have the chemical potential $\mu = -T\partial S/\partial N$ and partition function

$$\Omega = \sum_n e^{-\beta(E_n - \mu N_n)}. \tag{C.15}$$

Appendix D

The Shannon-Nyquist Theorem

For more detail see Allen et al. [129].

D.1 The Fourier Transform

For a function $x(t)$ the Fourier Transform is defined as

$$\tilde{x}(\omega) = \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt, \tag{D.1}$$

with the inverse transformation

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \tilde{x}(\omega) e^{i\omega t} d\omega. \tag{D.2}$$

These definitions are consistent since

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} e^{i(\omega-\omega')t} d\omega = \delta(\omega - \omega'), \tag{D.3}$$

where $\delta(x)$ is the Dirac Delta distribution.

D.2 The Time Discrete Fourier Transform

If x_n is a discrete signal then the Time Discrete Fourier Transform is

$$X(\omega) = \sum_{n=-\infty}^{\infty} x_n e^{-in\omega} \tag{D.4}$$

with

$$x_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{in\omega} d\omega. \tag{D.5}$$

Thus we see that the Time Discrete Fourier Transform takes a discrete signal, x_n, and produces a analogue one, $X(\omega)$.

D.3 The Sampling Theorem

An analogue signal $x(t)$ is band-limited if its Fourier Transform $\tilde{x}(\omega)$ exists and has finite support [1]. Suppose $x(t)$ is an band-limited analogue signal that we sample at regular intervals of time T. Defining $s_n = x(nT)$ we find

$$s_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} S(\omega)e^{in\omega}d\omega = x(nT) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\tilde{x}(\omega)e^{i\omega nT}d\omega. \tag{D.6}$$

Since $x(t)$ is band-limited we may suppose the non-zero values of $\tilde{x}(\omega)$ are confined to $[-b, b]$ then

$$s_n = x(nT) = \frac{1}{2\pi}\int_{-b}^{b}\tilde{x}(\omega)e^{i\omega nT}d\omega. \tag{D.7}$$

By changing variables with $\theta = \omega T$ we obtain

$$s_n = \frac{1}{2\pi T}\int_{-bT}^{bT}\tilde{x}(\theta/T)e^{in\theta}d\theta. \tag{D.8}$$

By choosing $T < \frac{\pi}{b}$ this becomes

$$s_n = \frac{1}{2\pi T}\int_{-\pi}^{\pi}\tilde{x}(\theta/T)e^{in\theta}d\theta, \tag{D.9}$$

and we have

$$\tilde{x}(\omega) = TS(\omega T). \tag{D.10}$$

Thus, we can recover the full signal from the discrete samples when $T < \frac{\pi}{b}$. Here b is given in radians per second, in hertz the sampling time condition becomes

$$T < T_{\text{crit}} = \frac{1}{2F_{\max}} \tag{D.11}$$

where T_{crit} is the critical sampling time and $F_{\max}$ is the maximum frequency with $b = 2\pi F_{\max}$. This concludes the Shannon-Nyquist Sampling Theorem, to recover the analogue signal we use

$$\begin{aligned} x(t) &= \frac{1}{2\pi}\int_{-\infty}^{\infty}\tilde{x}(\omega)e^{iwt}d\omega = \frac{T}{2\pi}\int_{-\pi/T}^{\pi/T}S(\omega T)e^{iwt}d\omega \\ &= \frac{T}{2\pi}\sum_{n=-\infty}^{\infty}s_n\int_{-\pi/T}^{\pi/T}e^{i\omega(t-nT)}d\omega \\ &= \sum_{n=-\infty}^{\infty}s_n\text{sinc}\left(\frac{t}{T}-n\right) \end{aligned} \tag{D.12}$$

where $\text{sinc}(x) = \sin(\pi x)/\pi x$.

[1] The support of a function is set of inputs for which the function is non-zero

Bibliography

[1] Alan M Turing. Computing machinery and intelligence. In *Parsing the Turing test*, pages 23–65. Springer, 2009.

[2] Tom Mitchell. *Machine learning.* McGraw-Hill New York, 1997.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[4] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[5] Daniele Ravì, Charence Wong, Fani Deligianni, Melissa Berthelot, Javier Andreu-Perez, Benny Lo, and Guang-Zhong Yang. Deep learning for health informatics. *IEEE journal of biomedical and health informatics*, 21(1):4–21, 2016.

[6] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182, 2016.

[7] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning.* Springer Series in Statistics, second edition, 2009.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[9] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. 2002.

[10] Cameron Buckner and James Garson. Connectionism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2019 edition, 2019.

[11] Kurt Hornik, Maxwell Stinchcombe, Halbert White, et al. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[12] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

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

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[15] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[16] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[19] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[20] Michael A Nielsen and Isaac Chuang. *Quantum computation and quantum information*. AAPT, 2002.

[21] E. Merzbacher. *Quantum Mechanics*. Wiley, 1998.

[22] Gianluca Stefanucci and Robert Van Leeuwen. *Nonequilibrium many-body theory of quantum systems: a modern introduction*. Cambridge University Press, 2013.

[23] Fabian HL Essler, Holger Frahm, Frank Göhmann, Andreas Klümper, and Vladimir E Korepin. *The one-dimensional Hubbard model*. Cambridge University Press, 2005.

[24] Patrik Fazekas. *Lecture notes on electron correlation and magnetism*, volume 5. World scientific, 1999.

[25] John MD Coey. *Magnetism and magnetic materials*. Cambridge University press, 2010.

[26] Richard M Martin. *Electronic structure: basic theory and practical methods*. Cambridge University press, 2004.

[27] Steven R White. Density matrix formulation for quantum renormalization groups. *Physical review letters*, 69(19):2863, 1992.

[28] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal. Quantum monte carlo simulations of solids. *Rev. Mod. Phys.*, 73:33–83, Jan 2001.

[29] Emanuel Gull, Andrew J. Millis, Alexander I. Lichtenstein, Alexey N. Rubtsov, Matthias Troyer, and Philipp Werner. Continuous-time monte carlo methods for quantum impurity models. *Rev. Mod. Phys.*, 83:349–404, May 2011.

[30] E. Y. Loh, J. E. Gubernatis, R. T. Scalettar, S. R. White, D. J. Scalapino, and R. L. Sugar. Sign problem in the numerical simulation of many-electron systems. *Phys. Rev. B*, 41:9301–9307, May 1990.

[31] Paul Adrien Maurice Dirac. Quantum mechanics of many-electron systems. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 123(792):714–733, 1929.

[32] James Nelson, Rajarshi Tiwari, and Stefano Sanvito. Machine learning density functional theory for the hubbard model. 99:075132, Feb 2019.

[33] Walter Kohn and Lu Jeu Sham. Self-consistent equations including exchange and correlation effects. *Physical review*, 140(4A):A1133, 1965.

[34] Pierre Hohenberg and Walter Kohn. Inhomogeneous electron gas. *Physical review*, 136(3B):B864, 1964.

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

[36] Kelvin Ch'ng, Nick Vazquez, and Ehsan Khatami. Unsupervised machine learning account of magnetic transitions in the hubbard model. *Physical Review E*, 97(1):013306, 2018.

[37] Hiroki Saito. Solving the bose–hubbard model with machine learning. *Journal of the Physical Society of Japan*, 86(9):093001, 2017.

[38] Jianhua Ma, Puhan Zhang, Yaohua Tan, Avik W Ghosh, and Gia-Wei Chern. Machine learning electron correlation in a disordered medium. *Physical Review B*, 99(8):085118, 2019.

[39] Kevin Ryczko, David A Strubbe, and Isaac Tamblyn. Deep learning and density-functional theory. *Physical Review A*, 100(2):022512, 2019.

[40] Felix Brockherde, Leslie Vogt, Li Li, Mark E. Tuckerman, Kieron Burke, and Klaus-Robert Müller. Bypassing the kohn-sham equations with machine learning. page 872, 2017.

[41] Jonathan Schmidt, Carlos L Benavides-Riveros, and Miguel AL Marques. Machine learning the physical nonlocal exchange–correlation functional of density-functional theory. *The journal of physical chemistry letters*, 10(20):6425–6431, 2019.

[42] Javier Robledo Moreno, Giuseppe Carleo, and Antoine Georges. Deep learning the hohenberg-kohn maps of density functional theory. *Physical Review Letters*, 125(7):076402, 2020.

[43] K Schönhammer, O Gunnarsson, and RM Noack. Density-functional theory on a lattice: Comparison with exact numerical results for a model with strongly correlated electrons. *Physical Review B*, 52(4):2504, 1995.

[44] Klaus Capelle, Carsten A Ullrich, and Giovanni Vignale. Degenerate ground states and nonunique potentials: Breakdown and restoration of density functionals. *Physical Review A*, 76(1):012508, 2007.

[45] R Gaudoin and K Burke. Lack of hohenberg-kohn theorem for excited states. *Physical review letters*, 93(17):173001, 2004.

[46] Medha Sharma and MAH Ahsan. Organization of the hilbert space for exact diagonalization of hubbard model. *Computer Physics Communications*, 193:19–29, 2015.

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

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[49] Kyle Mills, Kevin Ryczko, Iryna Luchak, Adam Domurad, Chris Beeler, and Isaac Tamblyn. Extensive deep neural networks for transferring small scale learning to large scale systems. *Chem. Sci.*, 10:4129–4140, 2019.

[50] N David Mermin. Thermal properties of the inhomogeneous electron gas. *Physical Review*, 137(5A):A1441, 1965.

[51] Johnnie Gray, Leonardo Banchi, Abolfazl Bayat, and Sougato Bose. Machine-learning-assisted many-body entanglement measurement. *Physical review letters*, 121(15):150503, 2018.

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

[53] Marin Bukov, Alexandre GR Day, Dries Sels, Phillip Weinberg, Anatoli Polkovnikov, and Pankaj Mehta. Reinforcement learning in different phases of quantum control. *Physical Review X*, 8(3):031086, 2018.

[54] Emmanuel Flurin, Leigh S Martin, Shay Hacohen-Gourgy, and Irfan Siddiqi. Using a recurrent neural network to reconstruct quantum dynamics of a superconducting qubit from physical observations. *Physical Review X*, 10(1):011006, 2020.

[55] Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. Vampnets for deep learning of molecular kinetics. *Nature communications*, 9(1):1–11, 2018.

[56] Philip G Breen, Christopher N Foley, Tjarda Boekholt, and Simon Portegies Zwart. Newton versus the machine: solving the chaotic three-body problem using deep neural networks. *Monthly Notices of the Royal Astronomical Society*, 494(2):2465–2470, 2020.

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

[58] Michael J Hartmann and Giuseppe Carleo. Neural-network approach to dissipative quantum many-body dynamics. *Physical review letters*, 122(25):250502, 2019.

[59] Leonardo Banchi, Edward Grant, Andrea Rocchetto, and Simone Severini. Modelling non-markovian quantum processes with recurrent neural networks. *New Journal of Physics*, 20(12):123030, 2018.

[60] IA Luchnikov, SV Vintskevich, DA Grigoriev, and SN Filippov. Machine learning non-markovian quantum dynamics. *Physical Review Letters*, 124(14):140502, 2020.

[61] Heinz-Peter Breuer, Francesco Petruccione, et al. *The theory of open quantum systems*. Oxford University Press on Demand, 2002.

[62] J Robert Johansson, Paul D Nation, and Franco Nori. Qutip 2: A python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 184(4):1234–1240, 2013.

[63] Zewang Zhang, Shuo Yang, Chenxi Liu, Yimin Han, Ching Hua Lee, Zheng Sun, Guangjie Li, and Xiao Zhang. Predicting quantum many-body dynamics with long short-term memory based neural networks. *arXiv preprint arXiv:1905.09168*, 2019.

[64] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, 2011.

[65] James Nelson and Stefano Sanvito. Predicting the curie temperature of ferromagnets using machine learning. *Physical Review Materials*, 3(10):104405, 2019.

[66] Daniel C Mattis. History of magnetism. In *The Theory of Magnetism I*, pages 1–38. Springer, 1981.

[67] Stephen Blundell. *Magnetism in Condensed Matter (Oxford master series in condensed matter physics)*. Oxford University Press, 2001.

[68] Stefano Curtarolo, Gus LW Hart, Marco Buongiorno Nardelli, Natalio Mingo, Stefano Sanvito, and Ohad Levy. The high-throughput highway to computational materials design. *Nature materials*, 12(3):191–201, 2013.

[69] Stefano Sanvito, Corey Oses, Junkai Xue, Anurag Tiwari, Mario Zic, Thomas Archer, Pelin Tozman, Munuswamy Venkatesan, Michael Coey, and Stefano Curtarolo. Accelerated discovery of new magnets in the heusler alloy family. *Science advances*, 3(4):e1602241, 2017.

[70] Yibin Xu, Masayoshi Yamazaki, and Pierre Villars. Inorganic materials database for exploring the nature of material. *Jpn. J. App. Phys.*, 50(11S):11RH02, 2011.

[71] Thomas F Connolly. *Bibliography of magnetic materials and tabulation of magnetic transition temperatures*. Springer Science & Business Media, 2012.

[72] K.H.J. Buschow and E.P. Wohlfarth, editors. *Handbook of Magnetic Materials, Volumes 4-16 & 18*. Elsevier, 1988-2009.

[73] HJ Albert and LR Rubin. Magnetic properties of platinum metals and their alloys. *Advances in Chemistry series*, 98:1, 1971.

[74] Francoise Givord and Remy Lemaire. Proprietes cristallographiques et magnetiques des composes entre le cobalt et le lutecium. *Solid State Commun.*, 9(5):341–346, 1971.

[75] H Okamoto. The as- mn (arsenic-manganese) system. *Bulletin of Alloy Phase Diagrams*, 10(5):549–554, 1989.

[76] Valentin Stanev, Corey Oses, A Gilad Kusne, Efrain Rodriguez, Johnpierre Paglione, Stefano Curtarolo, and Ichiro Takeuchi. Machine learning modeling of superconducting critical temperature. *npj Computational Materials*, 4(1):1–14, 2018.

[77] Luca M Ghiringhelli, Jan Vybiral, Sergey V Levchenko, Claudia Draxl, and Matthias Scheffler. Big data of materials science: Critical role of the descriptor. *Phys. Rev. Lett.*, 114(10):105503, 2015.

[78] O Anatole Von Lilienfeld, Raghunathan Ramakrishnan, Matthias Rupp, and Aaron Knoll. Fourier series of atomic radial distribution functions: A molecular fingerprint for machine learning models of quantum chemical properties. *International Journal of Quantum Chemistry*, 115(16):1084–1093, 2015.

[79] Felix Faber, Alexander Lindmaa, O Anatole von Lilienfeld, and Rickard Armiento. Crystal structure representations for machine learning models of formation energies. *Int. J. Quant. Chem.*, 115(16):1094–1101, 2015.

[80] Logan Ward, Ankit Agrawal, Alok Choudhary, and Christopher Wolverton. A general-purpose machine learning framework for predicting properties of inorganic materials. *npj Computational Materials*, 2(1):1–7, 2016.

[81] Shyue Ping Ong, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L Chevrier, Kristin A Persson, and Gerbrand Ceder. Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Computational Materials Science*, 68:314–319, 2013.

[82] AZ Menshikov, G Takzei, Yu A Dorofeev, VA Kazantsev, AK Kostyshin, and II Sych. Magnetic phase diagram of cobalt–manganese alloys. *Zh. Eksp. Teor. Fiz.*, 89(4):1269–1279, 1985.

[83] LJ Swartzendruber, VP Itkin, and CB Alcock. The fe-ni (iron-nickel) system. *Journal of phase equilibria*, 12(3):288–312, 1991.

[84] J Crangle and D Parsons. The magnetization of ferromagnetic binary alloys of cobalt or nickel with elements of the palladium and platinum groups. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 255(1283):509–519, 1960.

[85] V Raghavan. Al-co-fe (aluminum-cobalt-iron). *Journal of Phase Equilibria and Diffusion*, 26(1):57, 2005.

[86] M Harper, B Weinstein, C Simon, et al. Python-ternary: ternary plots in python. zenodo, 2015.

[87] K. Ishida and T. Nishizawa. The co-mn system. *Bulletin of Alloy Phase Diagrams*, 11(2):125, 1990.

[88] W. C. Muellner and J. S. Kouvel. Magnetic properties of ni-rh alloys near the critical composition for ferromagnetism. *Phys. Rev. B*, 11:4552–4559, Jun 1975.

[89] A.J. McAlieter. The al-co system. *Bulletin of Alloy Phase Diagrams*, 10(6):646, 1989.

[90] M. Shiga, T. Kikawa, K. Sumiyama, and Y. Nakamura. Magnetic properties of metastable fe-al alloys produced by vapor quenching. *J. Magn. Soc. Jpn. Mag.*, 9:187, 1985.

[91] K. Sumiyama, Y. Hirose, and Y. Nakamura. Structural and magnetic properties of nonequilibrium disordered fe-al alloys produced by facing target type dc sputtering. *J. Phys. Soc. Jpn.*, 59:2963, 1990.

[92] T. Nishizawa and K. Ishida. The co-fe (cobalt-iron) system. *Bulletin of Alloy Phase Diagrams*, 5(3):250, 1984.

[93] T. Graf, C. Felser, and S.S.P. Parkin. Simple rules for the understanding of heusler compounds. *Prog. Sol. State Chem.*, 39:1, 2011.

[94] M. Yin, P. Nash, and S. Chen. *Intermetallics*, 57:34, 2015.

[95] V. Raghavan. Al-co-fe (aluminum-cobalt-iron). *J. Phase Equilib. Diff.*, 26(1):57, 2005.

[96] Stefano Curtarolo, Wahyu Setyawan, Gus LW Hart, Michal Jahnatek, Roman V Chepulskii, Richard H Taylor, Shidong Wang, Junkai Xue, Kesong Yang, Ohad Levy, et al. Aflow: an automatic framework for high-throughput materials discovery. *Computational Materials Science*, 58:218–226, 2012.

[97] A. Belsky, M. Hellenbrandt, V.L. Karen, and P. Luksch. New developments in the inorganic crystal structure database (icsd): accessibility in support of materials research and design. *Acta Cryst. B*, 58:364, 2002.

[98] Hieu Chi Dam, Viet Cuong Nguyen, Tien Lam Pham, Anh Tuan Nguyen, Hiori Kino, Kiyoyuki Terakura, and Takashi Miyake. A regression-based feature selection study of the curie temperature of transition-metal rare-earth compounds: prediction and understanding. *arXiv preprint arXiv:1705.00978*, 2017.

[99] K. T. Schütt, H Glawe, F Brockherde, A Sanna, K. R. Müller, and E. K. U. Gross. How to represent crystal structures for machine learning: Towards fast prediction of electronic properties. *Phys. Rev. B*, 89(20):205118, 2014.

[100] T Long, NM Fortunato, Yixuan Zhang, O Gutfleisch, and H Zhang. An accelerating approach of designing ferromagnetic materials via machine learning modeling of magnetic ground state and curie temperature. *arXiv preprint arXiv:1908.00926*, 2019.

[101] Peter Kirkpatrick and Clare Ellis. Chemical space. *Nature*, 432(7019):823, 2004.

[102] Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018.

[103] Yabo Dan, Yong Zhao, Xiang Li, Shaobo Li, Ming Hu, and Jianjun Hu. Generative adversarial networks (gan) based efficient sampling of chemical composition space for inverse design of inorganic materials. *npj Computational Materials*, 6(1):1–7, 2020.

[104] Juhwan Noh, Jaehoon Kim, Helge S Stein, Benjamin Sanchez-Lengeling, John M Gregoire, Alan Aspuru-Guzik, and Yousung Jung. Inverse design of solid-state materials via a continuous representation. *Matter*, 1(5):1370–1384, 2019.

[105] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[106] Mariya Popova, Olexandr Isayev, and Alexander Tropsha. Deep reinforcement learning for de novo drug design. *Science advances*, 4(7):eaap7885, 2018.

[107] Yibo Li, Liangren Zhang, and Zhenming Liu. Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics*, 10(1):33, 2018.

[108] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[109] Denis Kuzminykh, Daniil Polykovskiy, Artur Kadurin, Alexander Zhebrak, Ivan Baskov, Sergey Nikolenko, Rim Shayakhmetov, and Alex Zhavoronkov. 3d molecular representations based on the wave transform for convolutional neural networks. *Molecular pharmaceutics*, 15(10):4378–4385, 2018.

[110] Niklas WA Gebauer, Michael Gastegger, and Kristof T Schütt. Generating equilibrium molecules with deep neural networks. *arXiv preprint arXiv:1810.11347*, 2018.

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

[112] Katja Hansen, Franziska Biegler, Raghunathan Ramakrishnan, Wiktor Pronobis, O Anatole Von Lilienfeld, Klaus-Robert Muller, and Alexandre Tkatchenko. Machine learning predictions of molecular properties: Accurate many-body potentials and nonlocality in chemical space. *The journal of physical chemistry letters*, 6(12):2326–2331, 2015.

[113] Matthias Rupp, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole Von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical review letters*, 108(5):058301, 2012.

[114] Jörg Behler and Michele Parrinello. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Physical review letters*, 98(14):146401, 2007.

[115] Jörg Behler. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *The Journal of chemical physics*, 134(7):074106, 2011.

[116] Stefan Chmiela, Alexandre Tkatchenko, Huziel E Sauceda, Igor Poltavsky, Kristof T Schütt, and Klaus-Robert Müller. Machine learning of accurate energy-conserving molecular force fields. *Science advances*, 3(5):e1603015, 2017.

[117] Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8:13890, 2017.

[118] Stefan Chmiela, Huziel E Sauceda, Klaus-Robert Müller, and Alexandre Tkatchenko. Towards exact molecular dynamics simulations with machine-learned force fields. *Nature communications*, 9(1):3887, 2018.

[119] Lars Ruddigkeit, Ruud Van Deursen, Lorenz C Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17. *Journal of chemical information and modeling*, 52(11):2864–2875, 2012.

[120] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole Von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific data*, 1:140022, 2014.

[121] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[122] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[123] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[124] Justin S Smith, Olexandr Isayev, and Adrian E Roitberg. Ani-1: an extensible neural network potential with dft accuracy at force field computational cost. *Chemical science*, 8(4):3192–3203, 2017.

[125] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017.

[126] J Beenen and DM Edwards. Superconductivity in the two-dimensional hubbard model. *Physical Review B*, 52(18):13636, 1995.

[127] M Michael Denner, Mark H Fischer, and Titus Neupert. Efficient learning of a one-dimensional density functional theory. *Physical Review Research*, 2(3):033388, 2020.

[128] James Sethna. *Statistical mechanics: entropy, order parameters, and complexity*, volume 14. Oxford University Press, 2006.

[129] Ronald L Allen and Duncan Mills. *Signal analysis: time, frequency, scale, and structure*. John Wiley & Sons, 2004.