Quantum mechanical methods for in silico drug design

Force predictions via machine learning with applications to molecular dynamics simulations

A thesis submitted to The University of Manchester for the degree of Doctor of Philosophy in the Faculty of Biology Medicine and Health

2021

Ismaeel Ramzan
School of Health Sciences, Division of Pharmacy and Optometry.

Contents

Word count: 27,979

Abstract

A method for decomposing Cartesian forces into pairwise interactions is described and investigated. These forces are then fitted by a variety of machine learning techniques with the most notable being a residual neural network which grows in size during the course of training. We also construct a machine learning model that predicts Cartesian forces via the predictions of pairwise forces whilst evaluating the error on the Cartesian predictions.

We then apply these machine learned force fields to molecular dynamics simulations and evaluate their performance, investigating the issue of energy gain via calculation of intermediary structures along the trajectory.

Finally we show that our decomposition scheme is useful not only for the prediction of forces, but for other properties where the sum of the properties should be zero. The example used is charges.

Declaration

no portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning

Copyright statement

Acknowledgments

I would like to thank my supervisors Dr. Richard Bryce and Dr. Neil Burton for their continuous support over the course of this project. Collaboration with Dr. Neil Burton was crucial in the development of the work presented in this thesis. I would additionally like to thank Linghan Kong for helping to interface our force field into the AMBER software.

I would like to thank my parents for the patients and unwavering support and finally the multitude of people who have impacted my life and supported me in the completion of this work.

Chapter 1

Introduction

1.1 An overview of simulation

Since the invention of the computer, computational power has continuously increased. This has enabled a revolution in science, technology and engineering with computer simulations forming a routine part of a plethora of fields. The applications of these simulations are extremely varied, from simulating proteins with applications to drug discovery[1] such as virtual screening[2] to simulations important to safety such as crack propagation in materials[3].

Techniques for simulation are equally varied and use different levels of approximation to model the real world system as appropriate for the application.

The two main limiting factors for the usefulness of any simulation are the time taken to obtain results from the simulation and the accuracy to which the model predicts the phenomena under investigation, with the latter naturally being informed by the former and both being affected by availability of computational power. With more computational power comes the ability to make fewer potentially inaccurate assumptions whilst completing the simulation in an acceptable time frame.

Though computational power is continually increasing, as noted, through careful design of simulation methods we can maximise the usefulness of any model.

Advances in computational power will not render such efforts to improve simulation techniques in vain, for the developed techniques may be applied to ever increasingly complicated systems.

Simulation represents a cost effective method of replacing expensive experimentation, improving safety and acquiring new insights into systems. In the work presented here we focus on a class of simulations used to model small drug like molecules using a technique known as molecular dynamics; however the applications of the developed techniques may be beyond this scope as well.

In particular, we focus on the use of machine learning in order to accurately predict the forces between atoms to a high degree of accuracy with a minimal impact on the time required to make a prediction compared with existing less accurate methods.

1.2 Molecular Dynamics

Molecular dynamics (MD) involves simulating the real time motion of systems by Newtonian mechanics. A very general overview of the procedure is, calculate the forces for the current structure, use these forces to calculate an acceleration and combine this with velocity and position information to make a perturbation to the atomic positions r over a time step δt.

1.2.1 Taking a step

Algorithms for the calculation of the next structure $\mathbf{r}(t + \delta t)$ are based on the assumption that dynamics properties (positions,velocities, accelerations, jerk, jounce, etc) can be approximated by a Taylor expansion series (equation 1.1).

$$\sum_{n=0}^{\infty} \frac{f^n(a)}{n!}(x-a)^n \tag{1.1}$$

These include the Verlet algorithm (equation 1.2), which does not require the velocities explicitly

$$\mathbf{r}(t+\delta t) = 2\mathbf{r}(t) - \mathbf{r}(t-\delta t) + \delta^2\mathbf{a}(t) \tag{1.2}$$

and is derived from the addition of the expansion series taking a step backwards and forwards (equations 1.4 and 1.3).

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \mathbf{v}(t)\delta t + \frac{1}{2}\delta t^2\mathbf{a}(t) \tag{1.3}$$

$$\mathbf{r}(t-\delta t) = \mathbf{r}(t) - \mathbf{v}(t)\delta t + \frac{1}{2}\delta t^2\mathbf{a}(t) \tag{1.4}$$

A potential issue of the Verlet algorithm is it requires the computation of the difference of potentially large numbers $2\mathbf{r} - \mathbf{r}(t - \delta t)$, also it does not explicitly calculate velocities which are required for the kinetic part of total energy calculations.

These disadvantages are overcome by the leap-frog algorithm[4], which gets its name from the fact that the positions and the velocities are calculated at staggered time steps, which are defined using equations 1.5 and 1.6.

$$\mathbf{r}\left(t + \delta t\right) = \mathbf{r}(t) + \delta t \mathbf{v}\left(t + \frac{1}{2}\delta t\right) \tag{1.5}$$

$$\mathbf{v}\left(t + \frac{1}{2}\delta t\right) = \mathbf{v}\left(t - \frac{1}{2}\delta t\right) + \delta t \mathbf{a}(t) \tag{1.6}$$

In order to implement this algorithm one first applies equation 1.6 to work out $\mathbf{v}(t + \frac{1}{2}\delta t)$ then equation 1.5 to work out the new positions at timestep t, then the velocities are $t + \frac{1}{2}\delta t$ are recalculated and so on, with the positions and velocities being staggered by half the timestep.

The staggered nature of the velocities and distances adds extra difficulty to the calculation of the kinetic energy part of the total energy. For the sake of simplicity in our implementation of MD in chapter 4 we opt to use the Verlet algorithm. There are other methods such as the velocity Verlet algorithm, which gives velocities and accelerations without compromising on the accuracy of the verlet algorithm[5] but are all based on the Taylor series.

Note in any method for taking a step the next structure is dependent on the current one. This means that the successive structures obtained are interdependent and any errors in the prediction are cumulative.

Although the choice of integration algorithm may permit longer time steps, it is usually not the limiting factor in the accuracy and speed to which one can conduct a calculation. The main limiting factor being the time taken to calculate the forces and the accuracy to which they are computed.

1.2.2 Calculating the forces

Accurate forces may be obtained by application of quantum mechanical methods. However these forces, though accurate are very time consuming to calculate. Semi-empirical methods such as PM6[6], which we use to compare performance of our method in chapter 3, make assumptions about the integrals that need to be calculated based on data to speed up the calculations.

We choose to compare to PM6 as Semi-empirical methods represent a competitor to the new class of machine learned force fields, running in a faster time than methods such as density functional theory whilst still retaining a good level of accuracy.

More simpler to compute still are forcefield methods, which are often classical mechanical in nature. For example consider the general amber forcefield (GAFF)[7], which we run a simulation of aspirin with in order to compare to our simulations in chapter 4, which can be given by equation 1.7.

$$E_{pair} = \sum_{bonds} (K_r(r - r_{eq})^2) + \sum_{angles} (K_\theta(\theta - \theta_{eq})^2) + \sum_{dih} \left(\frac{V_n}{2}(1 + cos(n\phi - \gamma)) \right) + \sum_{i<j} \left(\frac{A_{ij}}{R_{ij}^{12}} - \frac{B_{ij}}{R_{ij}^{6}} + \frac{q_i q_j}{\epsilon R_{ij}} \right) \tag{1.7}$$

Where r_{eq} and θ_{eq} correspond to the equilibrium positions of a bond and angle respectively, K_r,K_θ and V_n are force constants, n is multiplicitly (how many minima one observes) and γ is the phase angle.

Force fields such as GAFF are then parametrized using a mixture of experimental data and computational calculations, for a range of atoms in different chemical environments.

A new class of methods have been developed in recent years which use machine learning in order to make predictions of forces. These methods all take datasets of structures and fit a function to make predictions of energies or forces. Machine learning methods for predicting energies or forces will be discussed in more detail in section 1.3.

1.2.3 Analysing the results

The collection of structures one obtains from a molecular dynamics simulation is called a trajectory. A trajectory is useful for analysing the time dependent behaviour of a system - a notable advantage over other simulation techniques such as the Metropolis Monte Carlo approach.

A trajectory is also useful for determining which conformers a molecule is likely to adopt and the free energy cost of a transition between them.

1.2.4 Adding a solvent

Many of the systems of interest do not exist in a vacuum but are surrounded by the presence of a solvent such as water. The effects of solvent can be accounted for using either implicit or explicit methods. In implicit solvent models the solvent is modelled as a continuum and adds a perturbation to the gas phase behaviour of a system.[8]

Alternatively solvent may be modelled explicitly where the behaviour of the solvent molecules and their effects on the solute are modelled via molecular dynamics. As interactions with the solvent molecules in explicit solvent are modelled as charge-charge interactions and van der Waals interactions. We attempt to predict charges in chapter 5 based on the conformation of the solute which may be useful in more accurately modelling the explicit interactions with solvent.

1.3 Machine learning and its applications to chemistry

Machine learning is a subclass of artificial intelligence in which the computer acquires new behaviours or knowledge based on the results of experience or data. The applications of machine learning are extremely diverse and in many areas machine learning has begun to equal or surpass the abilities of humans, for example in the image classification test set ImageNet[9] where human classification is only 11% better on average.

There are many different approaches to machine learning but one of the most successful in recent years for a variety of applications is the artificial neural network, especially with the possibility to construct deep learning models.

1.3.1 Artificial neural networks

Artificial neural networks are a machine learning technique loosely based on the workings of the brain. They consist of a series of 'artificial neurons' which are connected to each other. Each neuron applies a function, called an activation function to a weighted sum of its inputs (figure 1.1).

There are many activation functions to choose from but two common ones are the sigmoid function (equation 1.8) or the rectified linear unit (relu) function[10] (equation 1.9)

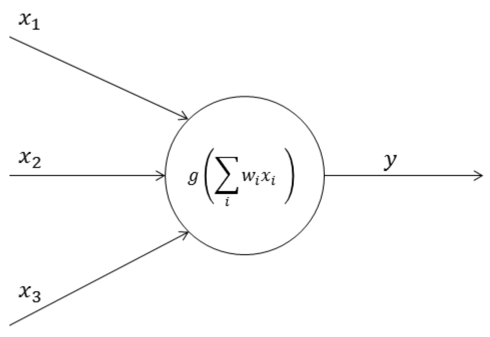


Figure 1.1: An example of an artificial neurone with 3 inputs x_1 x_2 and x_3 which applies a function g of their weighted sum and returns y

$$y = \frac{1}{1 + \exp(-\sum(w_i x_i))} \tag{1.8}$$

$$y = \max(0, \sum w_i x_i) \tag{1.9}$$

The choice of activation function can be important depending on network architecture as we will see in our discussion of network training.

From the activation functions listed it should be easy to see that a single neuron is limited in the functions it can fit. As such multiple neurons are employed in a variety of architectures. The simplest consists of neurons that form a single 'hidden layer' followed by an output layer but often multiple hidden layers are used (see Figure 1.2). Hidden layers are called as such as they lie between the inputs and the outputs and consist of any network layer that does not predict the final output.

It has been shown that a neural network consisting of a single hidden layer of infinite size may fit any arbitrary continuous function. This is referred to as universal approximation theory and was proved first for sigmoid functions[11], then later for any bounded nonconstant function[12]. Although universal approximation theorem offers some reassurances of the capabilities of a neural network, it does not offer any insights as to how to acquire a set of weights that fit a

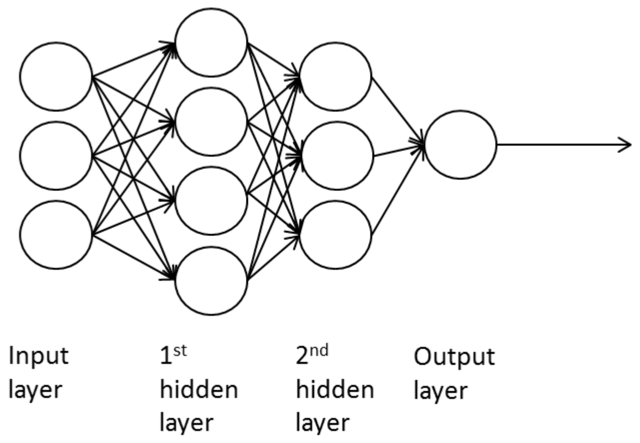


Figure 1.2: The connectivity of a feedforward neural network. The input layer serves to introduce inputs into the network and does not actually apply an activation function. Layers between the input layer and output layer are called hidden layers.

desired function.

The values of the weights of a neural network need to be optimized in order to make good predictions of a function. This optimization procedure is referred to as training. There are multiple strategies for training a neural network such as reinforcement learning[13] which is useful for learning behaviours in the context of game playing for example or unsupervised learning[14] which is useful for learning classes or patterns in unlabelled data (such as a picture of a cat without the label 'cat'). However the type of learning that is most applicable to our interest (the prediction of forces) is called supervised learning. Supervised learning involves using labelled data and attempting to learn the appropriate labels for the data, in our context molecular structures and the forces on each of the atoms.

For most fixed architectures, the number of neurons, how they are connected and the activation functions employed are all predetermined. This means the weights are the only adjustable parameter to better produce the desired output for a given input. Though there are multiple strategies for the selection of initial weights with some potentially offering the ability to reach better fits faster[15], there is always a need to conduct an optimization. This optimization procedure involves assigning 'blame' to hidden neurons and making adjustments of their weights accordingly.

For example, consider the backpropagation algorithm which adjusts the weights according to equation 1.10:

$$\Delta w\left(n\right) = \alpha \delta\left(n\right) x\left(n\right) + \lambda \Delta w\left(n-1\right) \tag{1.10}$$

where $\Delta w\left(n\right)$ is the change in a weight for the n^{th} time; α is a constant referred to as the learning rate; λ is known as the momentum constant. δ has two different expressions depending on if the neuron is in an output layer or a hidden layer given by equations 1.11 and 1.12 respectively:

$$\delta_o = f'(a) \times (t - y) \tag{1.11}$$

$$\delta_h = f'(a) \times \sum_{j \in I_h} \delta_j w_{jh} \tag{1.12}$$

where δ_o and δ_h are for output and hidden neurons; $f'(a)$ is the derivative of the activation function; the sum $j \in I_h$ is over all neurons that take an input from the output of the neuron of interest; δ_j is the function δ for the neuron receiving an input from the neuron of interests output and w_{jh} is the weight of this connection. The values of t and y correspond to the target output value and the outputted value respectively.[16]

From this scheme it should be possible to see that the weight adjustments of the earlier layers in a neural network are dependent on the gradients of the later layers they connect to, multiplied together. This is problematic in deeper networks containing multiple hidden layers as the gradient function δ_h may become very small and these weights will not change through successive training cycles. This is referred to as the vanishing gradient problem[17] and is an issue in networks which employ neurons such as sigmoid functions which may have shallow gradients in some regions they are sensitive to. The vanishing gradient problem may be overcome by the use of activation functions such as the relu activation function which has the same gradient in all regions of sensitivity.

Consider for example the ANI-1 model[18], a neural network with Gaussian activation functions in its hidden layers which learned to make predictions of energies based on structures. The authors noted that having 3 or 4 hidden layers offers the best performance which is likely a direct consequence of the vanishing gradient problem as any more layers would mean the earlier layers will not train.

Although optimization algorithms which do not depend on the gradients have been developed such as the use of particle swarm optimization[19], it is still most common to use algorithms which use the gradients, with algorithms such as the Adam optimizer[20] employed routinely in the optimization of neural networks and is included within the tensorflow library[21].

In our work we only use relu functions in the hidden layers of networks which are deep (consisting of more than two or three hidden layers) and have a fixed architecture to overcome this issue and indeed we are able to obtain well performing deep architectures as a consequence.

Another potential issue one could encounter when training a neural network is overfitting. This occurs when the training data has error associated with it (for example experimental data) but instead of fitting a good general trend, the network instead fits through every point exactly. This is countered by the use of 'cross validation', which involves a set of inputs that the network's performance is evaluated on, but not trained on. If during training the error of the cross validation set increases, even if the training error is decreasing, this is indicative of overfitting and training should be terminated early.

In our work we exclusively fit computational data calculated in a deterministic manner meaning there is no error at any of the training points (within floating point precision) and ideally we should fit each point exactly. As such in our work we do not employ cross validation to terminate training early.

So far we have only considered what are called 'feed forward networks' with a fixed defined architecture which take a fixed number of inputs. However there are many different types of neural network which we will discuss now.

Architecture types

A disadvantage of using a fixed architecture is that one is required to define a suitable number of hidden layers, the number of neurons in each and connectivity. Failing to do so appropriately could significantly impact the results of the machine learning experiment. The simplest extension to allow some variation in network architecture involve the use of pruning algorithms[22] which cut connections between neurons during the course of training, simplifying the architecture as training progresses with the hopes of attaining a better general fit.

Other approaches such as cascade-correlation neural networks[23] or Neural Evolution of Augmenting Topologies (NEAT)[24], take a more extreme approach to changing architecture as the network is trained. In both cascade correlation and NEAT the network begins with only a single output neuron.

At the start of cascade-correlation first the output neuron(s) (connected to the network inputs) are optimized. Then a series of trial neurons are trained independently receiving the network inputs as inputs, to maximise the correlation between their output and the error over the training data. The neuron with the highest correlation after training is then selected and inserted into the network. It is connected to the network inputs with its output connected to the output neuron(s). The output neuron(s) are then re-optimized. Then another set of neurons is selected, with input connections to the network inputs but also the previously added neuron and the process is repeated. Every neuron that is added is connected to the network inputs and all previous neurons. Cascade correlation produces deep neural networks with every new neuron being connected to all previous ones. Although implementations of cascade correlation exist, it would not have been easy to implement in our choice of neural network library, Keras (see Section 1.7), as such we opted not to use it.

NEAT uses genetic algorithms in order to optimize the weights,connectivity and architecture simultaneously ensuring sensible crossovers by tracking innovations in the structure. An extension of NEAT is HyperNEAT[25] which uses NEAT to construct a network known as the complex pattern producing network (CPPN) which is then used to define the weights between neurons and inputs based on their positions in space. This is very interesting as it means the resulting connection weights are a function of the geometry of the system. An example of how the CPPN may be queried in order to determine a connection between two nodes is presented in figure 1.3.

Although the use of an adapted HyperNEAT was considered because of its interesting feature of being a function of the positions of the various nodes, this was ultimately discarded as an option because of the lack of good implementations of the code at the time.

Another very important development in network architectures is the convolutional neural network (CNN). These networks attempt to make use of the geometry of the problem by learning features of the inputs invariant to translations implicitly. Although the features learned are not implicitly invariant to rotations it is possible to learn rotationally invariant features by presenting the network with the same input patterns but rotated to some degree of success. For example, in the work conducted by Dieleman, Willett and Dambre,[26] where CNNs were used for the prediction of galaxy type based on images of galaxies, they found that by including specific rotations of the input data it yielded a model which appeared to show invariance to rotations in its predictions. CNNs have also been successfully applied to chemical problems such as the work con-

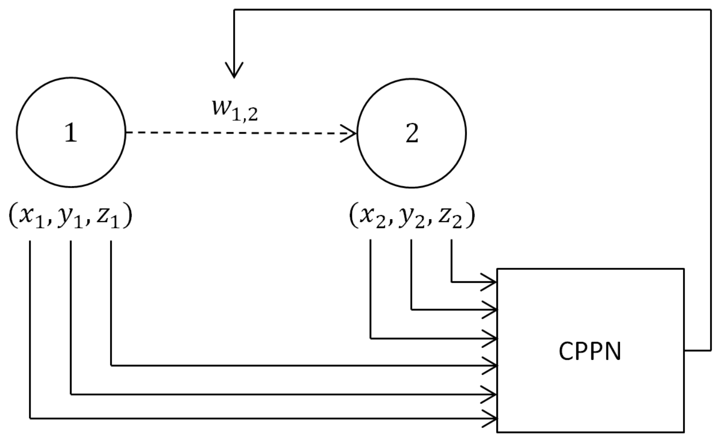


Figure 1.3: An example of determining the weight between two neurons by querying the CPPN by passing the coordinates in space of each one. Note sometimes other information is also passed such as the distance between them.

ducted by Ragoza et al[27], who conducted protein-ligand docking calculations based on the results of training a CNN which took 3D 'pictures' of molecular structures as input and they outperformed both other machine learning techniques and the AutoDock Vina scoring function.

For simplicity sake we shall consider a convolutional that takes a two dimensional input such as an image. In a convolutional neural network neurons in the first hidden layer can be thought of as being arranged in independent sheets. The neurons in each sheet have a defined 'receptive field', a region of input space to which they are sensitive. In order to encode translational invariance of the detected features every neuron in the sheet will have an identical set of weights for the inputs in its receptive field. For example in Figure 1.4 for both a) and b) the weight of the connection of the top left neuron in their receptive fields (coloured in black) will be identical.[28]

The sharing of weight values yields far fewer weights to be optimised in a CNN as compared to its fully connected counterparts. This reduction in parameters can improve generalisation.[26]

After the first hidden layer it is common to employ a 'pooling layer'. These layers have a receptive field to the previous layer. They do not have a trainable weight set but rather serve to combine multiple features together using a well defined rule. The idea of this is to reduce the dimensionality of the problem and eliminate noise sensitivity.[29]

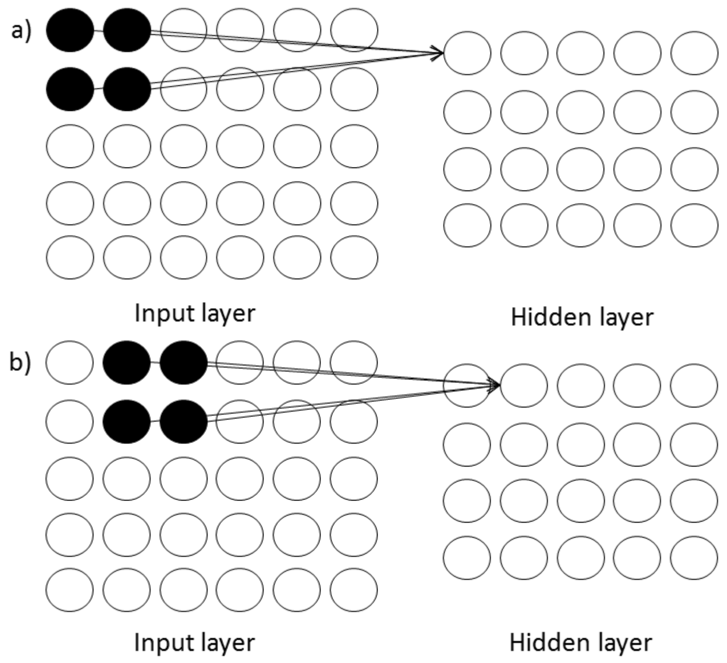


Figure 1.4: The connections for two neurons in the hidden layer of a convolutional neural network. In this example each neuron has a 2×2 receptive field which is coloured black.

The exact number of layers and their arrangement are subject to ongoing research. Some of the more recent networks have had a very large number of layers, one of the largest (at the time of writing) having 152 layers.[30]

Deep convolutional neural networks have come to dominate the field of image classification in recent times.[31] A class of convolutional nets which use a shortcut connection to add the input to the output of a convolutional layer (see Figure 1.5) have proven to be the most powerful tools for image classification to date based on the results of the "ImageNet Large Scale Visual Recognition Competition" (ILSVRC) image classification challenge. It is argued the success of such networks comes about as a result of the layers attempting to learn a residual or slight change to the input rather than hoping individual layers learn appropriate features.[30]. For very deep networks the use of these shortcut connections allows for better performance than similar networks of equal depth that do not have them. They consist of 1 or more convolutional layers and then a combinational layer. The combinational layer takes the output after the series of convolutions as well as the input to them and combines them using methods such as addition or a fully connected neural network layer.[32]

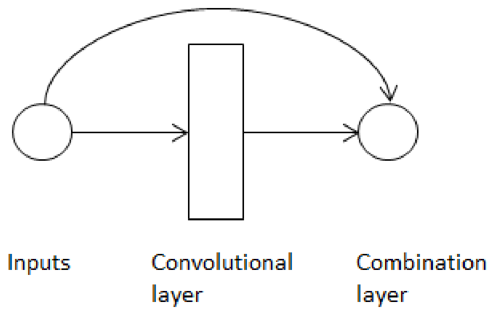


Figure 1.5: An example of a residual unit containing a shortcut connection bypassing the convolutional layer. In the combinational layer various methods may be employed ranging from a simple addition to a fully connected neural network layer.

Given their successes, convolutional networks have had in various fields but more specifically in chemical applications such as the prediction of docking scores but also in the case of the SchNet convolutional network's[33] ability to predict molecular properties we decided to use convolutional networks as our model.

For our input we initially used a grid based system which created a 3D image of a molecule to be used as input based on equation 1.13:

$$y = \sum_{i}^{N} \begin{cases} \left(\frac{m_i}{\sqrt{2 \times \pi \times \sigma_i^2}}\right) \exp\left(\frac{-d^2}{2 \times \sigma_i^2}\right) & \text{if } d < 2.0 \\ 0 & \text{if } d \geq 2.0 \end{cases} \tag{1.13}$$

where y is the value at a particular grid point; N is the number of atoms; σ_i is half the covalent radius of atom i; m_i is the mass of atom i and d is the distance between the grid point and the atom.

To yield 3D grids such as the ones shown in figure 1.6.

However this idea was later abandoned due to concerns about rotational invariance. The appropriate selection of rotations/schemes to orientate a molecule and the large computational overhead related to constructing and storing the grids. The networks we fit in Section 3.2 are not convolutional in nature as this would not have been appropriate for the input used but do contain features such as shortcut connections, which were shown to improve performance in CNNs and also build the architecture as the network trains to exploit the advantages of this.

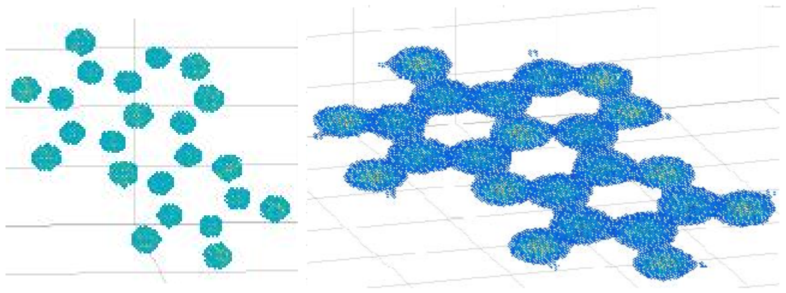


Figure 1.6: A $200 \times 200 \times 200$ grid populated according to equation 1.13. The (x,y,z) positions are plotted with the colour determining the value at each point. A threshold is used to remove points below a certain value. The left image has a higher threshold value than the right. The large number of grid points is to aid visualisation

1.3.2 Other machine learning methods

Although the focus of our work was the use of neural networks we will now discuss some of the other machine learning methods that exist.

One of the oldest forms of machine learning is the use of rule based systems. These have a series of rules of the format of IF THEN statements. Additionally Rules may also be acquired from data such as in the case of the Dendral project.

The Dendral project was an influential project concerned with the identification of unknown organic compounds based on the results of mass spectrometry and used rules in order to conduct its analysis. More interestingly another program called Meta-Denral was constructed which analysed the rules of Dendral based on training data consisting of mass spectra and organic compounds they were derived from. Not only was Meta-Dendral able to rediscover the known rules of mass spectroscopy, it was able to contribute to the discovery of new rules and make good predictions for mass spectra outside the original training data.

Although due to lack of marketability the Dendral project was ultimately discontinued, the various modules which made up the program have been adapted to other purposes such as chemical database management.[34]

Another commonly used method is radial basis function or a similar method called kriging. Radial basis function interpolation is discussed further in section.3.3

When conducting Kriging one first constructs and fits a variogram (see 1.9.3), this is then used to calculate weights for training structures, for a given new structure. These weights are then used to make a prediction.

1.3.3 Input design

The design of an appropriate input scheme is key in determining the performance of any machine learning experiment. Three important factors must be considered namely, logical adequacy (completeness), heuristic power and notional convenience. The first is how well can the input scheme distinguish every unique input, or in our context can the input scheme adequately distinguish different conformations of a molecule. The second is how well does the input relate to the output. An input that has a perfect linear correlation with the output will be very easy to fit but more complicated relationships may be less well fitted. The final relates to how large the input scheme is and how easy it is to compute, using more than 3N-6, where N is the number of atoms, numbers to represent a chemical structure is sub optimal from a notional convenience point of view, but doing so may offer more heuristic power.

A variety of techniques exist for encoding chemical structures for use by a machine learning method each with advantages and disadvantages.

One example is the symmetry functions conceived by Behler and Parrinello.[35] These serve to encode the enviroment around a single atom and were used in conjunction with a neural network to predict energies of atoms. They defined two types of symmetry functions: the first is a radial symmetry function (G) defined by the equation 1.14:

$$G_i = \sum_{j \neq i}^{N} e^{-\eta (R_{ij} - R_s)^2} f_c\left(R_{ij}\right) \tag{1.14}$$

where R_c is the cut off distance; η and R_s are parameters for the Gaussian function; R_{ij} is the interatomic distance, N is the total number of atoms and f_c defined in equation 1.15, is a cut off function;.

$$f_c(R_{ij}) = \begin{cases} 0.5 \times \left[\cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1\right] & \text{if } R_{ij} \leq R_c \\ 0 & \text{if } R_{ij} > R_c \end{cases} \tag{1.15}$$

The second type of symmetry function is an angular symmetry function given by the equation:

$$\begin{aligned} G_i = 2^{1-\zeta} \sum_{j,k\neq i}^{N} \left(1 + \lambda \cos\theta_{ijk}\right)^{\zeta} \\ \times e^{-\eta\left(R_{ij}^2 + R_{ik}^2 + R_{jk}^2\right)} f_c\left(R_{ij}\right) f_c\left(R_{ik}\right) f_c\left(R_{jk}\right) \end{aligned} \tag{1.16}$$

where the value of ζ determines the angular resolution of the symmetry function with higher values yielding a narrower range of none zero values for the symmetry function and λ has values between zero and one and serves to shift the the maxima of the cosine function.

A set of multiple symmetry functions are used for a single atom in order to describe its environment, in Behler and Parrinello's experiment which consisted of modelling the energies of silicon atoms, a total of 48 different symmetry functions were used.

These symmetry functions were extended by Smith, Isayev and Roitberg to account for different chemical environments[18] which was shown to improve performance. As most neural networks require a fixed number of inputs, symmetry functions are a potentially good choice of input for experiments where multiple molecules of different sizes should be fitted simultaneously. However they are not necessarily a complete input and they can be computationally expensive to compute and the constants difficult to define.

In our early work we attempted to use a modified version of symmetry functions to describe the environment around a pair of atoms. We chose to describe the environment around a pair rather than a single atom as our work consists of making pairwise predictions such as those presented in chapter 2. We developed functions which were sensitive to angles, dihedrals and distances (with all combinations to assure invariance to permutations). We found that generating the symmetry functions required significant computational time, that some had large values whereas others were barely influenced and that the neural network fit was poor. Given the difficulty associated with optimising this input scheme as well as optimizing the network architecture employed we instead opted for an input in our work not too dissimilar from coulomb matrices.

Coulomb matrices are another popular input choice and are $N \times N$ matrices (where N is the number of atoms) that encode the distance between atoms and the product of their charges as a single number according to the equation:

$$M_{ij} = \begin{cases} 0.5Z_i^{2.4} & \text{if } i = j \\ \frac{Z_i Z_j}{|\mathbf{R}_i - \mathbf{R}_j|} & \text{if } i \neq j \end{cases} \tag{1.17}$$

where $\mathbf{R}_i$ and $\mathbf{R}_j$ are the Cartesian coordinates of atoms i and j.[36] In this representation the size of the matrix will change as a function of the number of atoms. This may make them unsuitable for neural networks for the aforementioned reason of neural networks generally requiring a fixed input and output size.

In our work we primarily use the lower triangle of the nuclear repulsion force matrix as our input. This is appropriate as it is complete, forms part of the total force between a pair of atoms and is relatively easy to compute.

1.4 Delaunay triangulation and natural neighbour interpolation

Delaunay triangulation is a technique for joining points in space in order to form simplices (triangles in 2D, tetrahedrons in 3D etc). The condition that must be satisfied is that for any given simplex, the circumcircle in 2D, circumsphere in 3D, or circum-hypersphere in higher dimensions, joining the points of a simplex must not contain any other points.

An example of a Delaunay triangulation is shown in figure 1.7.[i]

In Chapter 3 we considered the use of Delaunay triangulation to reduce the number of pairwise interactions between atoms. However we found this introduced a situation whereby the number of outputs was not necessarily constant and which atoms still had connections could vary from structure to structure making it nigh on impossible to fit.

In Chapter 4 we considered the use of Delaunay triangulation to identify the convex hull defined by our training data to identify frames in a simulation where the forces were extrapolated but found this too computationally expensive to compute as the data is 3N-6 dimensional.

The Delaunay triangulation is the dual graph of the Voronoi diagram. The Voronoi diagram is constructed by drawing lines in positions where multiple

[i]Delaunay triangulation Image (figure 1.7) by By N es derivative work: Matthias Kopsch - This file was derived from: Delaunay circumcircles centers.png:, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=18929097

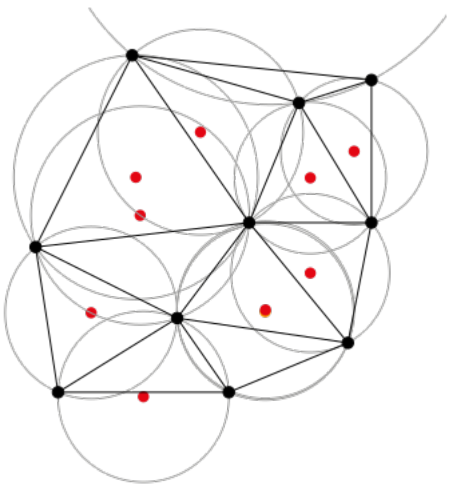


Figure 1.7: The Delaunay triangulation of 2D points (black dots) with circumcircles shown with centers(red dots)

points in the data are equidistant. These diagrams are observed in nature for example in dragonfly wings (figure 1.8)[ii] or giraffe spots.

The area of a Voronoi polygon can give us insight into how well sampled the area around the centre point. A large Voronoi polygon means there are no nearby points and that the region is sparsely sampled, whereas a small area means there are nearby points and the area is well sampled. The Voronoi diagram may be a useful tool in dataset construction because of this. However we opt to use available datasets instead (see section 1.8).

The Voronoi diagram can also be used to conduct an interpolation method called natural neighbour interpolation[37]. Natural neighbour interpolation works by first constructing the Voronoi diagram for all the training data. Then a point to be interpolated is inserted into the diagram, the subsequent change in area of the polygons when the diagram is recalculated with the point to be interpolated added in is calculated. These are used to generate weights for each training point (w) which are calculated by equation 1.18;

$$w_i = \frac{\Delta A_i}{\Delta A_{total}} \tag{1.18}$$

where w_i is the weight for training point i, ΔA_i is the change in area for the polygon corresponding to point i and ΔA_{total} is the total change in area of all

[ii]Dragonfly wing photo by Joi Ito on flickr.com

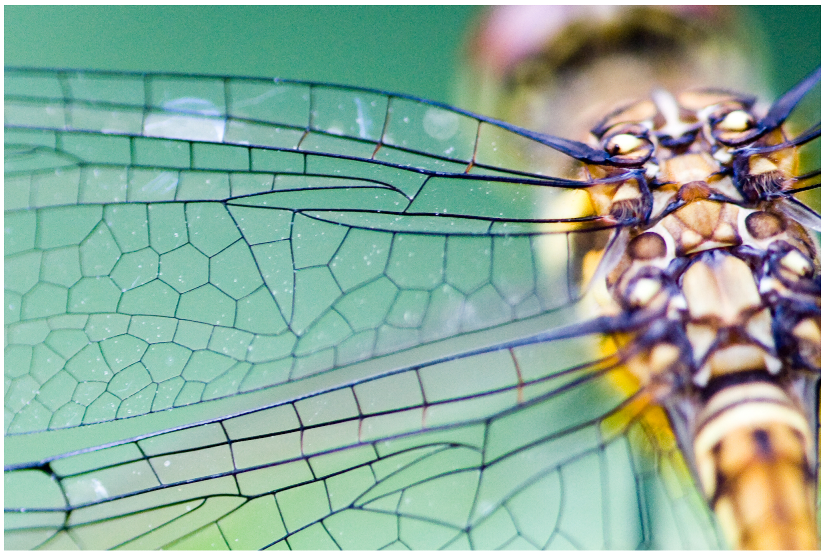


Figure 1.8: Wing of a dragonfly, note the pattern of polygons, these can be modelled by a Voronoi diagram.

polygons. The interpolated value is calculated using equation 1.19;

$$y(x) = \sum_{i=1}^{n} w_i f(x_i) \tag{1.19}$$

where $y(x)$ are the value(s) to predict for interpolated point x, w_i is the weight for training point i and $f(x_i)$ are the value(s) at position x_i.

In our early work we used a special kind of neural network called an autoencoder, these networks have an output identical to their input. However they have a layer that consists of only a few neurons (in our case two neurons). Then the output of those two neurons was used to define the now 2D positions of all the training points. With this lower dimensionality it became possible to construct the Voronoi diagram and conduct natural neighbour interpolation. We noticed very poor performance though likely due to a poor correlation between these 2 values and the force to predict and the local nature of the technique employed - natural neighbour interpolation only considers the points nearest to the one to be interpolated with no consideration of global trends.

1.5 Fit forces or energies?

Although one requires forces to update the positions of the atoms in the system during the course of an MD simulation, predictions of the forces may be obtained from energies as the force is simply the negative of the first derivative of energy, shown in equation 1.20.

$$\mathbf{F} = -\frac{\delta E}{\delta \mathbf{R}} \tag{1.20}$$

where $\mathbf{R}$ is the position of a particle.

Energies represent an easier value to predict as they are scalar values rather than forces which are vectors. Schemes for fitting energies usually involve allowing the machine learning method to decompose the energies internally and then summing them to give a total energy.[18 35]

A scheme which uses this approach is the ANI-1 neural network[18]. This network consisted of 4 layers of neurons with a Gaussian activation function which made per atom predictions of energies. The network used modified versions of the symmetry functions introduced in section 1.3.3 and was fitted using 80% of a very large dataset consisting of 58,000 small molecules sampled to give nearly 17.2 million conformers.

The deep tensor neural network (DTNN) scheme[38] is similar to ANI-1 except that instead of using symmetry functions to model the chemical environment, they instead employ an iterative method which serves to pass 'messages' from one atom to another, which over multiple iterations serves to capture information about the environment.

The SchNet architecture[33] also makes predictions of arbitrarily decomposed atomic energies. It used a convolutional neural network to make these predictions but it also incorporated forces into its error function by analytically calculating the first derivative of the model with respect to molecular positions. They demonstrate that by including this force information in their error function they are able to obtain better predictions compared to simply making their error function a function of the total energies only. The SchNet architecture employs a convolutional neural network with radially sensitive convolutional filters and short cut connections as well.

The creators of the SpookyNet method[39] argue it is important to include information not just pertaining to nuclear positions, such as symmetry functions or atoms sending messages, but also to incorporate electronic information as part of the input. As such as well using the Cartesian coordinates as inputs, they also include the spin state of the system and the number of electrons in the system. By including these inputs they demonstrated better performance than SchNet on the QM7 dataset.

An alternative approach to arbitrarily decomposing the energy within a machine learning method is to calculate energies for individual atoms and then fit these independently. This was done by Fletcher and Popelier who calculated per atom energies using a technique called interacting quantum atoms (IQA)[40] in the development of the FFLUX force field[41]. An issue with this approach is the large computational cost of running IQA calculations to generate the training data, which ultimately serves to limit the usefulness of such a technique to smaller systems.[42]

An advantage of fitting forces is there is more information per structure to work with. Whereas in most examples which fit energies there is only one global energy which then has to be decomposed by the machine learning method, there are 3N-6 independent forces. Methods such as GDML[43] fit forces directly because of this benefit and guarantee energy conservation in their fit. GDML used a distance metric between coulomb matrices as its input, it was later extended in sGDML[44] to account for molecular symmetry by attempting to find permutation matrices that transformed one input into another.

1.6 Notation used in this thesis

1.6.1 Neural network diagrams

Multiple neural network diagrams will be presented to illustrate the various architectures employed in this work. In order to simplify these diagrams and not show explicitly every neuron and every connection, a simpler approached is used.

A layer of neurons or a series of layers will be represented by a solid, labelled rectangle. Multiple rectangles with the same label may be present in one diagram which means the architecture is the same. However the weights are different unless otherwise stated.

A circle labelled with a function represents a layer that applies that function and has no optimizable weights.

A solid arrow represents fully connected when connecting between rectangles (every neuron in one rectangle's output connects to every neuron in the other rectangle's input).

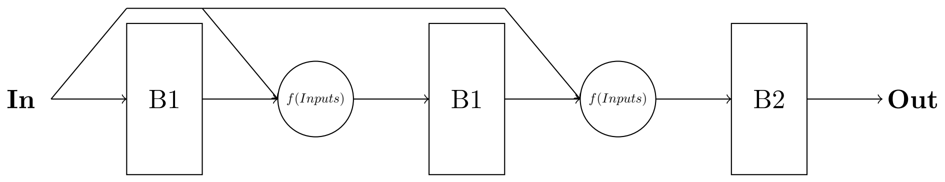


Figure 1.9: An example of a neural network diagram, showing two identical series of fully connected layers (B1) and a third block which is different (B2) as well as two instances of some defined function f

1.7 Software versions

Although it is possible to implement a neural network from scratch, doing so will be time consuming and likely yield worse performance compared to various libraries that exist. As there are multitude of libraries one can use to implement neural networks careful consideration must be given as to the best choice for the problem to be solved. The library should be simple to use, allow for the implementation of a wide range of architectures and optimize to a solution quickly.

We believed using the Keras frontend to the Tensorflow library represented the best choice for this. The Tensorflow library allows for the implementation of diverse architectures including custom functions and the Keras frontend allows this to be achieved in a only a few lines of code. This makes the implementation of network architectures easy. The Tensorflow library also implements some of the most (at the time of writing) up to date optimizers. Tensorflow is also extremely well optimized and can seamlessly use GPUs to speed up training, handling all the details of doing so internally.

A well maintained, powerful library Tensorflow represented one of the best choices for a neural network library to implement the architectures we designed, even if we could not easily implement algorithms such as cascade correlation.

All neural networks were implemented and run in keras version 2.2.4 with a tensorflow backend version 1.14.0.

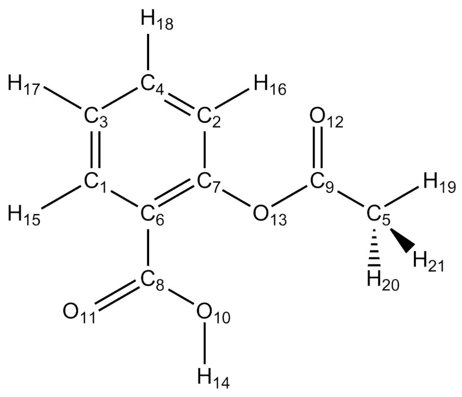
1.8 Datasets

Machine learning can require large datasets to generate good results. For the purposes of this project we use the MD17 dataset[43], using the positions and forces for aspirin, toluene and malonaldehyde. Throughout this project we assumed that the forces provided within this dataset are noise free, however the version used was before the corrections issued to the dataset in 2020 (https://archive.materialscloud.org/record/2020.82). Whereas this may explain the poor performance of our radial basis function fits presented in section 3.3, neural networks are excellent at dealing with noisy data which is why they are often employed in fields such as computer vision, voice recognition or many other applications where data may contain significant noise.

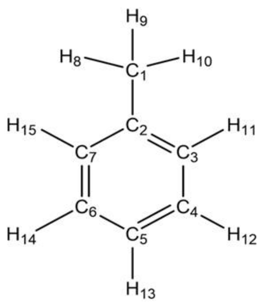
Our reasoning for not using cross validation in our network fits presented in section 3.2 is not valid. However given the similarity between the training error and the test error for our fits using this dataset, it is reasonable to conclude that over-fitting did not occur to a significant degree.

1.8.1 Atom numbering

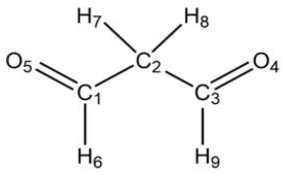
In order to aid discussion we number the atoms in aspirin, toluene and malonaldehyde according to figure 1.10



(a) Aspirin numbers



(b) toluene numbers



(c) malonaldehyde numbers

Figure 1.10: Numeric labelling scheme for aspirin toluene and malonaldehyde

1.8.2 Training and test set creation

The data were split into training and test sets by first ordering each structure by energy then evenly sampling aiming for 50k structures but never more and placing these data into the training set. The remaining structures formed the test set.

The dataset was created using molecular dynamics meaning minima and points around them will be sampled more than higher energy structures. An advantage of this is a molecular dynamics simulation will sample these points more meaning it is important the machine learning method learns to fit these the best as errors are cumulative in an MD simulation. However poor sampling of other structures means there is a higher probability that when making predictions for them, the network will be extrapolating rather than interpolating.

1.9 Analysis techniques

To analyse the performance of our machine learning models we employ multiple techniques which are discussed here.

1.9.1 S curves

When considering performance an average error such as the RMSE does not yield the complete picture. A small average error with a large range of errors may ultimately result in poor performance in the context of a molecular dynamics simulation.

S curves present a simple method for seeing the percentage of points that lie beneath a certain error. They are constructed by calculating the cumulative histogram of errors presented as a percentage of the total number of points within each bin versus the error given on a logarithmic scale. At any value of error on the x axis the percentage of points that have that error or lower may be read off the y axis.

Consider the example S curve presented in figure 1.11

Using figure 1.11 we can see that the percentage of points that have an error below 1 kcal mol^{-1} Å^{-1} is approximately 35%.

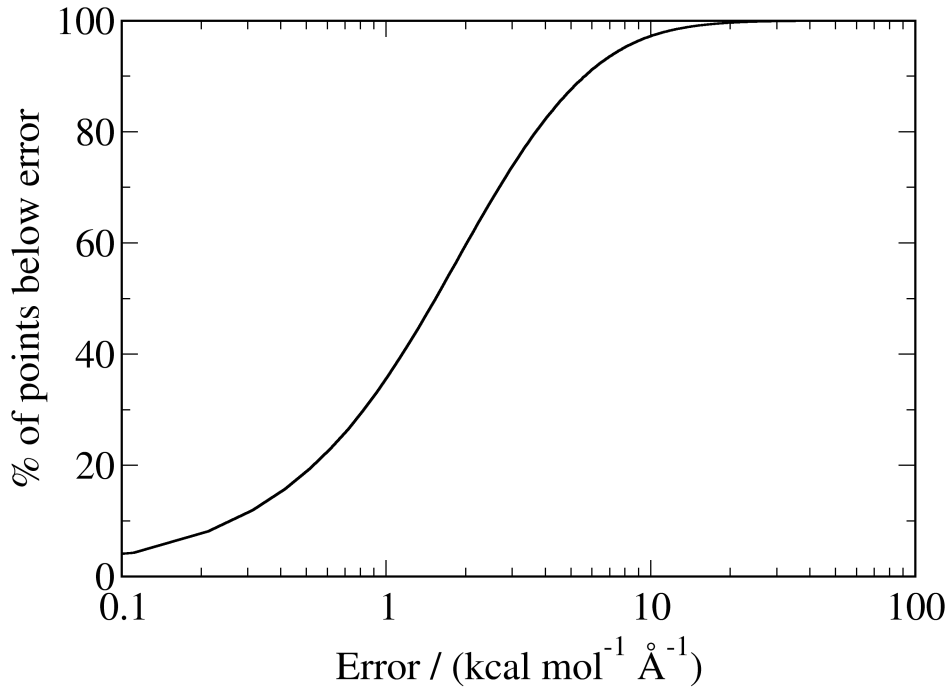


Figure 1.11: An example of an S curve, the x axis is a logarithmic scale

1.9.2 L1 plots

After creating a machine learning model to fit pairwise forces in chapter 3, it is of interest to see which interactions were fitted better than others. As there can be a large number of pairwise interactions in the system (as many as 210 in the case of aspirin), the construction of individual S curves for each pairwise interaction would not be a sensible way of achieving this, instead we plot ”L1 plots”.

An L1 plot consists of the interaction number on the x axis and the percentage of points in the test data that have an error of less than 1 kcal mol^{-1} Å^{-1} for that interaction on the y axis. This plot can be quickly used to indicate performance for the prediction of particular interactions and spot patterns in performance for certain classes of interaction. For example consider figure 1.12, we can that for interaction one's predictions, approximately 75% of them have

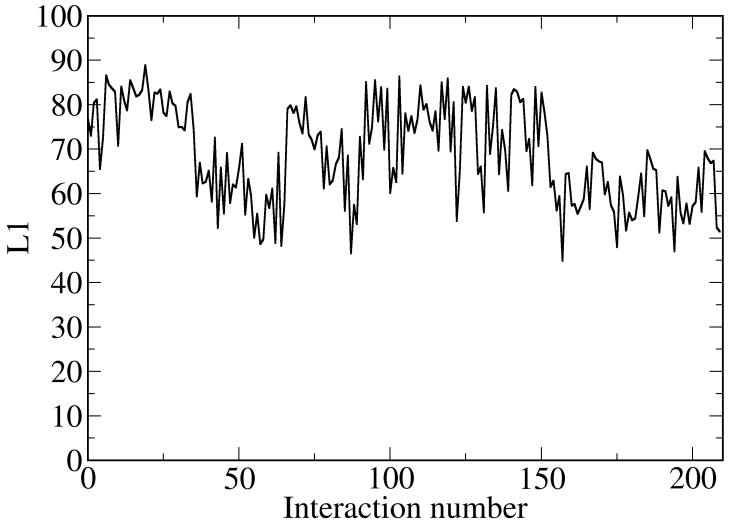


Figure 1.12: An example of an L1 plot

an error of less than 1 kcal mol^{-1} Å^{-1}.

1.9.3 Variograms

A variogram or semivariogram (the two terms are often used interchangeably) is a description of spacial continuity of data and gives an indication of how well the change in the value at a point in space correlates with the change in the distance between points.

The variogram plot is $\gamma(r)$ vs r where r is the distance between points and $\gamma(r)$ is given by equation 1.21

$$\gamma(r) = \frac{1}{2N} \sum_{i} \sum_{j}^{j<i} (Z_i - Z_j) \tag{1.21}$$

where Z_i and Z_j are the values at the two points separated by distance r and N is the number of pairs of points which have a separation of this distance.

As data do not usually lie on a regular grid, $\gamma(r)$ is calculated for bins of distance to yield meaningful information on the plot. An example of a simple variogram plot is given in figure 1.13.

Generally as the distance increases, so does the variability of values between points separated by that distance ($\gamma(r)$), up until a value of r where no or little correlation is observed between the value of $\gamma(r)$ and r. The value of $\gamma(r)$ this is observed at is called the sill and the value of r at which the curve reaches the sill gives us the range. At the range there there is no correlation between the values of Z_i and Z_j, meaning a distance metric can only be a useful predictor up until the value of the range.

A variogram fit should implicitly go through the origin of the plot as points separated by a distance of zero should have a value of $\gamma(r)$ of zero. However it is possible for a discontinuity to occur which would suggest that the plot does not go through the origin rather some other value on the y axis. This is called the "Nugget effect". The origin of the term comes from gold mining where samples taken at small distances from each other may show very different concentrations of gold because of the presence of gold nuggets causing a spike in the value of $\gamma(r)$ as the values of Z_i and Z_j will be very different over a short distance. The nugget effect could also be caused by experimental error, which over an entire dataset serves to cause a spike in $\gamma(r)$ at small distances.

As our data consists entirely of deterministically computationally calculated values, we will not consider the existence of a nugget effect in any of our analyses.

Figure 1.13 presents an idealistic view of what a variogram should look like and in reality, the experimental variogram shows much greater fluctuation. In

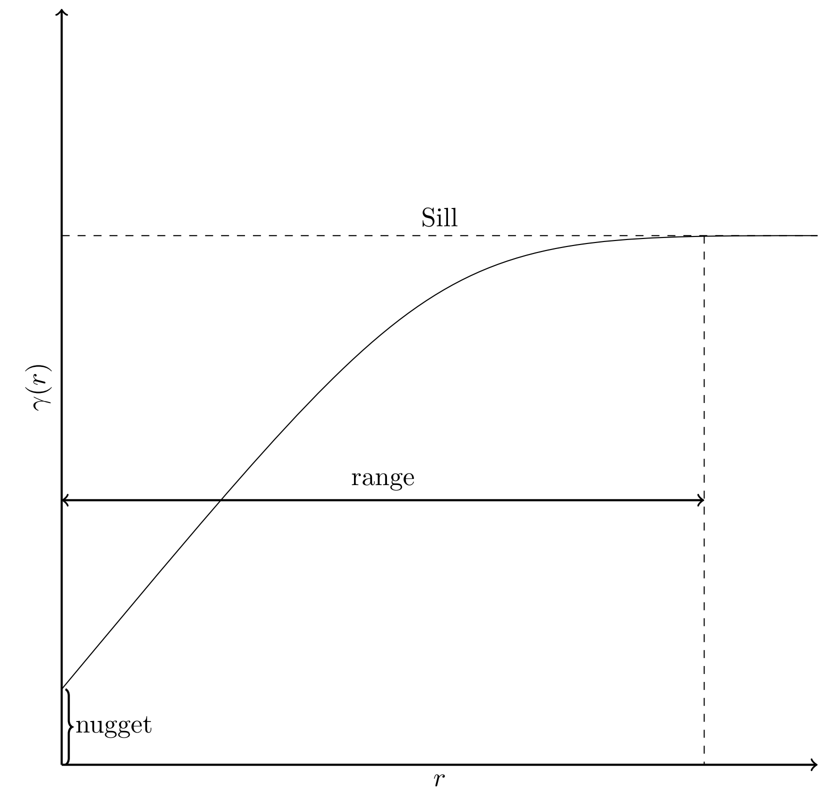


Figure 1.13: A simple example of a variogram plot with key features labelled

order to determine values for the sill, range and nugget the variogram is fitted using standard functions, such as cubic, gaussian or a spherical model and these key features are determined from the fit. As some of these models approach the value of the sill asymptotically the 'effective range' is often quoted which corresponds to the value of r at which 95% of the sill value has been observed.

When considering data on a 2D grid (for example mining data), it is possible that the direction also influences the value obtained for the effective range. As such it is common to plot variograms which consider points at a certain displacement from one another. As our data is very high dimensional, significantly limiting the number of points separated by a vector, when we plot variograms we do not consider the directional dependence of the variogram.

Chapter 2

Pairwise force decomposition schemes

2.1 Introduction

We wish to improve the performance of molecular dynamics simulations by creating a new force field generated by machine learning that predicts the forces close to quantum accuracy but at a fraction of the time cost of existing methods. Such a method would allow for simulations to be conducted in a similar time frame as simulations run using force fields. However the simulations will have a greater accuracy in the forces yielding an overall more accurate simulation. An appropriate output must be constructed for the prediction of forces in order to achieve this. We present a series of schemes here that all revolve around treating forces as the sum of pairwise interactions and discuss the benefits of them.

An extra difficulty in force prediction, as opposed to predictions of the molecular energy, is that the forces are vector quantities meaning they have a magnitude and direction. Additionally the forces in a system must obey Newtons third law - they must be equal and opposite (equations 2.1 and 2.2).

$$\epsilon_{translational} = \sum_{A=1}^{N} \mathbf{F}_A = \mathbf{0} \tag{2.1}$$

$$\epsilon_{rotational} = \sum_{A=1}^{N} (\mathbf{r}_A \times \mathbf{F}_A) = \mathbf{0} \tag{2.2}$$

where $\mathbf{r}_A = \mathbf{R}_A - \mathbf{R}_o$, $\mathbf{R}_A$ is the Cartesian coordinates of atom A, $\mathbf{F}_A$ is the Cartesian force on atom A, N is the number of atoms in the system and $\mathbf{R}_0$ is the rotational centre given by equation 2.3

$$\mathbf{R}_o = \frac{1}{N}\sum_{A=1}^{N}\mathbf{R}_A \tag{2.3}$$

In order to reduce the challenge of predicting forces, a machine learning output was proposed such that a machine learning model will only predict the magnitude and that obedience to Newtons third law is accomplished implicitly in the output. This is done by making predictions of the magnitude of the force between pairs of atoms with the direction defined as lying along the vector that connects them.

Obedience to Newton's third law is accomplished by having one atom take the positive of this predicted quantity and the other the negative, which can be illustrated by the case where we have two atoms A and B as shown in figure 2.1.

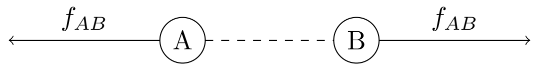


Figure 2.1: Two atoms A and B interacting with force $\mathrm{f_{AB}}$ which the machine learning model would predict the magnitude of.

Another significant advantage of predicting the Cartesian forces via predictions of pairwise forces as opposed to direct predictions of the x, y and z components, is the invariance of the value to predict to rotations of the system. It must also be noted that if every pairwise interaction is to be predicted, $\binom{N}{2}$ predictions are necessary as opposed to $3N-6$ Cartesian forces.

It is possible to construct a machine learning method that makes predictions of these pairwise forces, combines them to obtain the Cartesian forces and then evaluates the error in those to use in its optimization. This will be described in section 2.6; however we believe the direct calculation of some pairwise forces and fitting these conveys numerous advantages.

One advantage of calculating the pairwise forces as opposed to allowing a machine learning method to predict them internally is that it allows us to potentially construct different independent models for each pairwise interaction or class of pairwise interaction. Another advantage is it allows us to run sanity checks to see if the pairwise forces show trends we would expect to see (bond stretching/squeezing, dihedral angle rotations) rather than simply hoping a machine learning model learns such trends. Additionally the pairwise forces could be calculated in such a manner that distant interactions have a force close to zero and therefore can be ignored, helping to deal with the issue of having to make $\binom{N}{2}$ predictions.

2.2 The vanilla force decomposition scheme

What we call the vanilla scheme is a simple method for obtaining pairwise forces from Cartesian forces. Let us consider a system containing 3 atoms shown in figure 2.2.

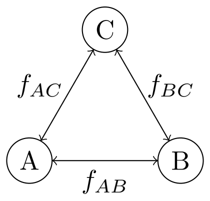


Figure 2.2: 3 atoms A, B and C with their pairwise forces f_{AB}, f_{BC} and f_{AC}

In the absence of external influences, the Cartesian force on each atom may be given as a sum of the projection of all the pairwise forces connected to it into Cartesian space by the following equations:

$$\begin{aligned}
\mathbf{F}_A &= \mathbf{e}_{AB} f_{AB} + \mathbf{e}_{AC} f_{AC} \\
\mathbf{F}_B &= -\mathbf{e}_{AB} f_{AB} + \mathbf{e}_{BC} f_{BC} \\
\mathbf{F}_C &= -\mathbf{e}_{AC} f_{AC} - \mathbf{e}_{BC} f_{BC}
\end{aligned}$$

where $\mathbf{F}_A$ $\mathbf{F}_B$ and $\mathbf{F}_C$ are the Cartesian forces on atom A,B and C respectively and $\mathbf{e}$ is a normalized projection into Cartesian space which may be given by equation 2.4:

$$\mathbf{e}_{AB} = \frac{\mathbf{R}_B - \mathbf{R}_A}{\|\mathbf{R}_B - \mathbf{R}_A\|} \tag{2.4}$$

We now have a set of simultaneous equations which in matrix format may be given by equation 2.5.

$$\mathbf{F} = \mathbf{e} \cdot \mathbf{f} \tag{2.5}$$

where $\mathbf{F}$ is a vector of length $3N$ of Cartesian forces, $\mathbf{e}$ is a matrix of dimensions $3N$ by $\binom{N}{2}$ of projections and $\mathbf{f}$ is a vector of length $\binom{N}{2}$ of pairwise interactions.

For the three atom system presented in figure 2.2 the matrix $\mathbf{e}$ is:

$$\mathbf{e} = \begin{bmatrix} e^x_{AB} & e^x_{AC} & 0 \\ e^y_{AB} & e^y_{AC} & 0 \\ e^z_{AB} & e^z_{AC} & 0 \\ -e^x_{AB} & 0 & e^x_{BC} \\ -e^y_{AB} & 0 & e^y_{BC} \\ -e^z_{AB} & 0 & e^z_{BC} \\ 0 & -e^x_{AC} & -e^x_{BC} \\ 0 & -e^y_{AC} & -e^y_{BC} \\ 0 & -e^z_{AC} & -e^z_{BC} \end{bmatrix}$$

where the term e^x_{AB} for example may be given by:

$$e^x_{AB} = \frac{R^x_A - R^x_B}{\|\mathbf{R}_A - \mathbf{R}_B\|}$$

where R^x_A is the x coordinate of atom A. For our three atom system the set of simultaneous equations is an overdetermined system. However because of the constraints on the forces arising from equations 2.1 and 2.2 there are only 3 unique forces that are linearly independent thus a unique solution exists.

The equations are solved using equation 2.6, as the matrix $\mathbf{e}$ is not square the Moore-Penrose pseudo inverse is used to find values for $\mathbf{e}^{-1}$.

$$\mathbf{f} = \mathbf{e}^{-1}\mathbf{F} \tag{2.6}$$

For systems which have 5 atoms or more the system of equations is underdetermined and has an infinite number of solutions; however by using the Moore-Penrose pseudo inverse to find the inverse of $\mathbf{e}$ when applying equation 2.6 we pick the solution that gives the smallest norm.

In the general case the vector $\mathbf{f}$ will consist of the lower triangle of an N by N matrix of all pairwise interactions and will be $\binom{N}{2}$ in length, $\mathbf{e}$ will be $3N$ by $\binom{N}{2}$, the columns will correspond to the pairwise interactions in the vector $\mathbf{f}$ whilst each of the N atoms will have three rows corresponding to the x, y and z components of the force. The matrix $\mathbf{e}$ will have zeros in columns where the atom on the corresponding row is not involved in the pairwise interaction, it will be positive if the row number is the lower atom index for the two atoms interacting and negative if the row number is the higher of the two indices. The Moore-Penrose pseudo inverse will be applied to the matrix **e** which will be dotted with a vector created by flattening the matrix **F** into a vector of length $3N$.

2.2.1 Analysis of the vanilla force decomposition scheme

The vanilla scheme does not take into consideration atomic masses or distances, which are two important factors in determining the magnitude of an interaction but is still able to produce rather intuitive pairwise forces. For the purposes of this analysis the pairwise nuclear repulsion force (NRF) is used which is given by equation 2.7 where q_A and q_B are the nuclear charges on atoms A and B respectively.

$$NRF_{AB} = \frac{q_A q_B}{r_{AB}^2} \tag{2.7}$$

Let us begin by looking at the application of the vanilla scheme to the aspirin training data presented in section 1.8. For this dataset we plot the pairwise vanilla force against the pairwise nuclear repulsion force (Figure 2.3).

From figure 2.3 we are able to see multiple 'bands'. These correspond to different interactions and we decompose the plot to investigate some of these separately in figure 2.4.

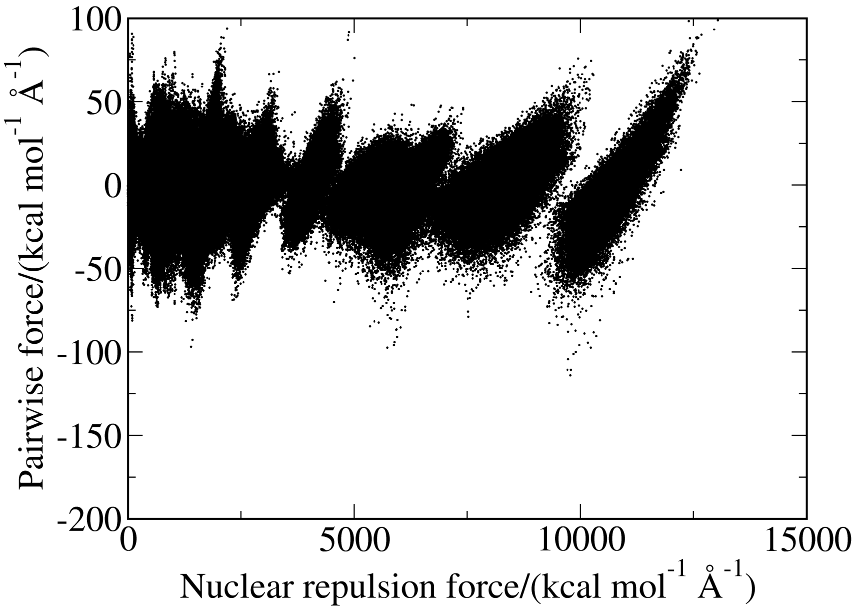
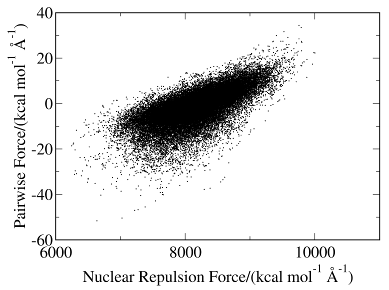


Figure 2.3: The pairwise vanilla force vs the pairwise nuclear repulsion force for aspirin.

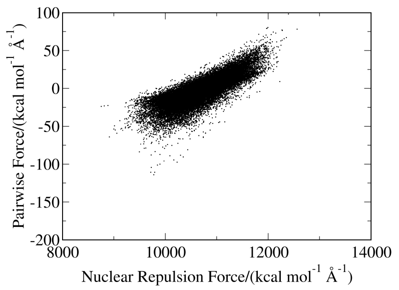
From figure 2.4 we can see that the correlation between the decomposed force and the nuclear repulsion force is greatest for the closer interactions with double and single bonds showing good correlation whereas more distant ones correlate less.

As for the 1-4 interaction shown in figure 2.4d it appears to have two separate bands associated with it. Plotting the NRF versus the dihedral angle (figure 2.6) shows that there are two conformers. Though these conformers may be distinguished by the NRF the relationship between NRF and the dihedral angle is weak in this dataset. We repeat the numeric labelling scheme for aspirin in figure 2.5 to aid identification of the different interactions being investigated.

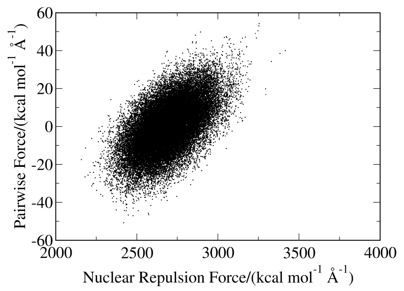
A plot of the vanilla decomposed pairwise force versus the dihedral angle yields no relationship 2.7.



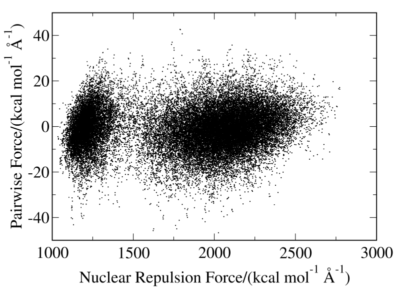
(a) Carbon oxygen single bond (atoms 7 and 13)



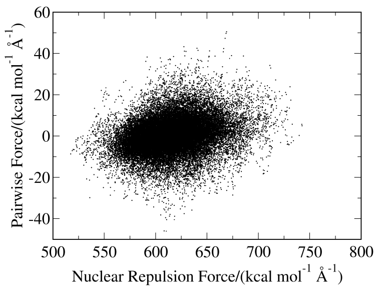
(b) Carbon oxygen double bond (atoms 8 and 11)



(c) Carbon oxygen 1 3 interaction (atoms 5 and 12)



(d) Carbon oxygen 1 4 interaction (atoms 1 and 10)



(e) Carbon oxygen non-bonding interaction (atoms 4 and 11)

Figure 2.4: Pairwise force calculated using the vanilla decomposition method vs Nuclear repulsion force for different carbon oxygen interactions in the aspirin training data.

We also see a similar band structure for toluene and malonaldehyde shown in figure 2.8 showing that the observed band structures are not unique to aspirin.

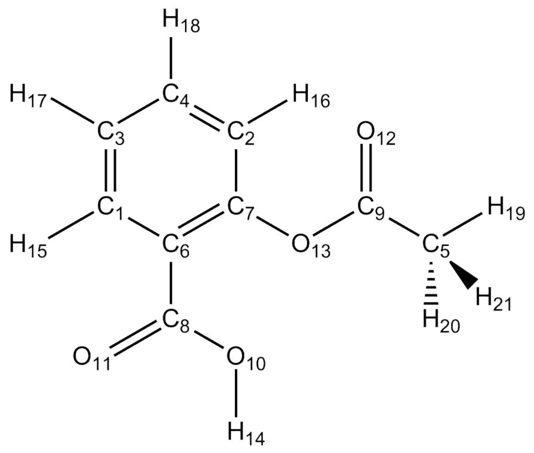


Figure 2.5: Aspirin numerical labelling scheme

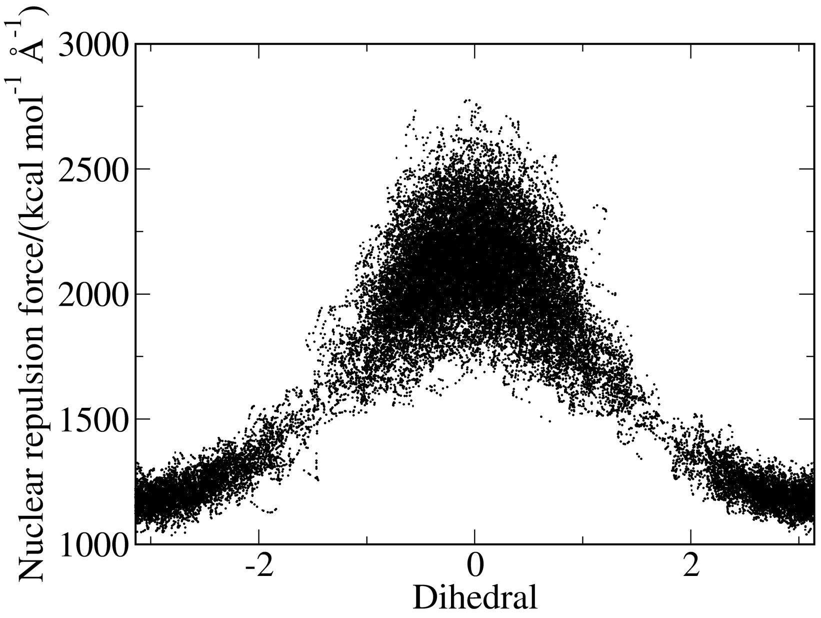


Figure 2.6: Nuclear repulsion force versus dihedral angle in radians for 1-4 interaction (atoms 1 and 10) in aspirin.

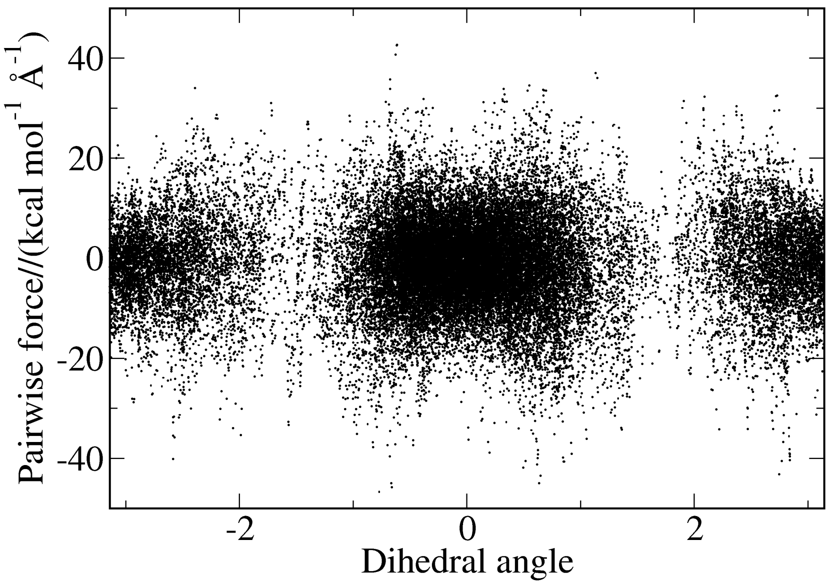
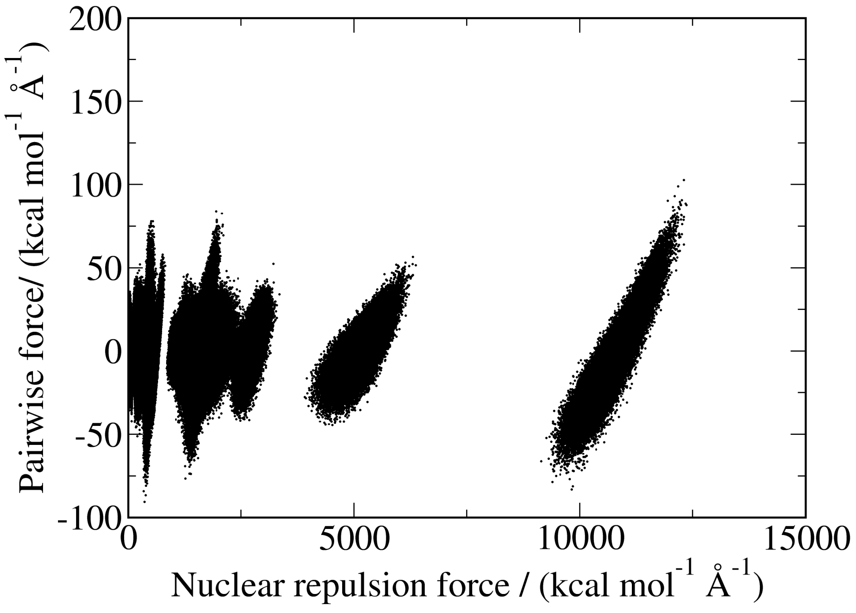
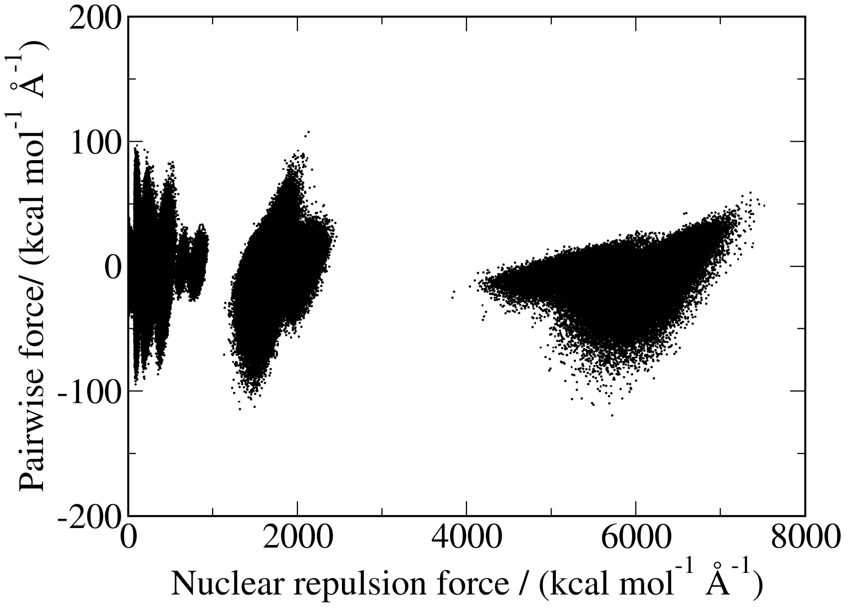


Figure 2.7: Vanilla decomposed force versus dihedral angle in radians for 1-4 interaction in aspirin (atoms 1 and 10)



(a) malonaldehyde



(b) toluene

Figure 2.8: Pairwise force decomposed by the vanilla method versus the nuclear repulsion force for toluene and malonaldehyde.

For malonaldehyde shown in figure 2.8a the large band furthest to the right at approximately 10000 kcal mol^{-1} Å^{-1} corresponds to the interactions between O4 and C3 as well as the interaction between O5 and C1, which is expected as these are double bonds between two heavy atoms and should give the highest values for the nuclear repulsion force. The next band at approximately 5000 kcal mol^{-1} Å^{-1} corresponds to interactions between C1 and C2 as well as C3 and C2 - bonding interactions between carbon atoms.

As we move more to the left in figure 2.8a the peaks correspond to longer range interactions between the heavier atoms, then interactions with the hydrogen atoms.

For toluene shown in figure 2.8b the band furthest to the right is far less well defined owing to the numerous interactions that contribute to it, these interactions are all the bonded interactions between the carbons in the aromatic ring. The next band to the left of this one which partially merges with the first is the interaction between C2 and C1. The remaining bands as with malonaldehyde correspond to longer range interactions and interactions with hydrogen.

2.3 Biasing the decomposition scheme

2.3.1 Introduction

The vanilla decomposition provides a set of pairwise forces from the infinite number of possible solutions to the set of simultaneous equations given by equation 2.6. The solution calculated does not consider the distance between atoms or even atom type.

Ideally the solution obtained would have larger forces for closer, heavier atoms and smaller forces for distant, lighter atoms, generally as it is chemically intuitive that closer heavier atoms interact more than distant light ones.

This would also allow us to define a sensible cut off that allows the number of pairwise interactions to be reduced from $\binom{N}{2}$. Additionally it is desirable to have larger forces on pairs of heavier atoms and smaller forces on pairs of lighter atoms, as in an MD simulation, errors in the force prediction of light atoms yield greater errors in the acceleration than for heavier atoms.

In order to achieve this we must select a different solution to the set of simultaneous equations, which can be accomplished by changing the values of $\mathbf{e}$ in equation 2.5.

2.3.2 Method

We note in section 2.2.1 that the nuclear repulsion force is able to make reasonable distinctions between distance and atom type and has large values for closer heavier atoms and smaller values for lighter closer atoms. As such we use this as our biasing function.

In order to bias the decomposition we multiply the values of **e** by the nuclear repulsion force for that interaction. We then obtain pairwise forces using this adjusted matrix, using the pseudo inverse as in the vanilla scheme. These obtained forces are then multiplied by the nuclear repulsion force in order to obtain a set of forces which give the correct total force when recombined using the equation 2.5 with an unbiased value of **e**.

This is done for the aspirin, toluene and malonaldehyde training sets.

2.4 Results and discussion

The NRF-biased pairwise forces follow a similar pattern as the vanilla scheme for aspirin; however the magnitude of the forces is larger (figure 2.9). We see similar patterns for the various interactions on the graph.

Our goal was to reduce the force magnitudes of interactions to lighter atoms, as such in figure 2.10 we replot figure 2.3 and figure 2.9 taking into consideration only interactions to hydrogen atoms. It can be seen in the plot that whereas some interactions at smaller values of the nuclear repulsion force have a smaller range of force values, others at higher values of nuclear repulsion force have a larger range.

Although the total Cartesian force on each hydrogen atom is the same in both the vanilla and the biased methods, it is hoped that if the pairwise interactions between two hydrogen atoms are minimized, the fitting error for these interactions will be smaller. This is desirable as an error in a pairwise prediction for a carbon-hydrogen interaction will contribute to an error in the Cartesian forces for a carbon atom and a hydrogen atom, whereas an error in the pairwise force prediction for two hydrogen atoms will contribute to the total error in the Cartesian forces for two hydrogen atoms, therefore having twice the impact on the total error in the Cartesian forces of hydrogen atoms.

Looking at the biased decomposition for toluene and malonaldehyde we see a similar pattern, a larger range of forces but still with visible band structures (figure 2.11).

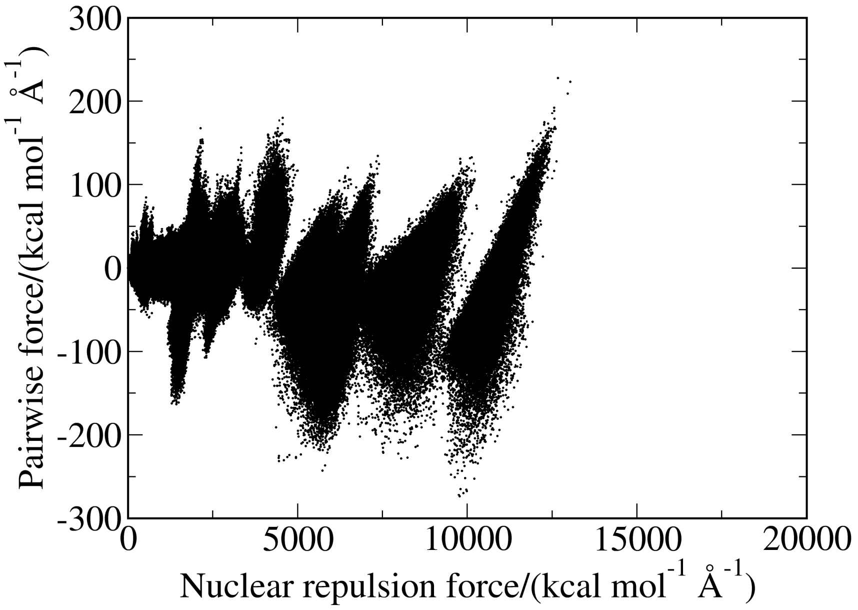
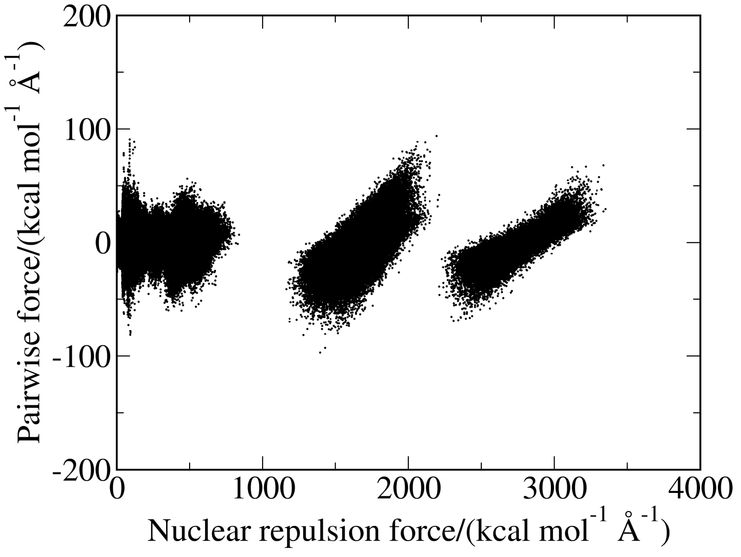
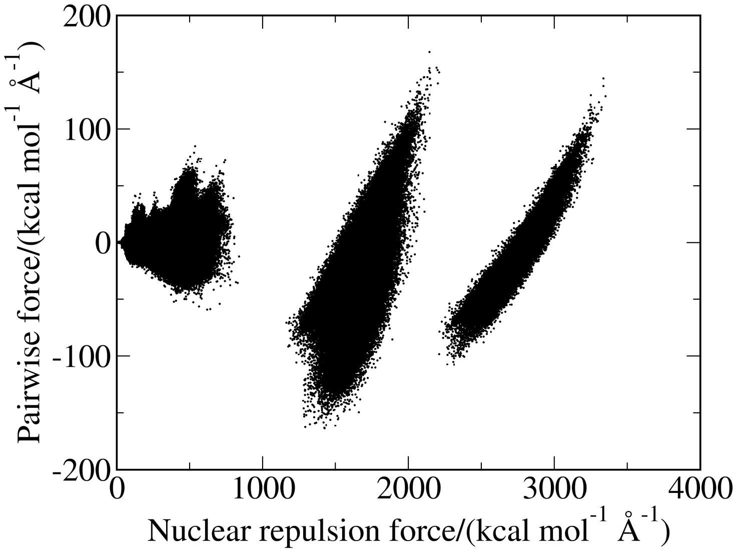


Figure 2.9: Nuclear repulsion force versus the pairwise forces for aspirin decomposed using the biased decomposition scheme.

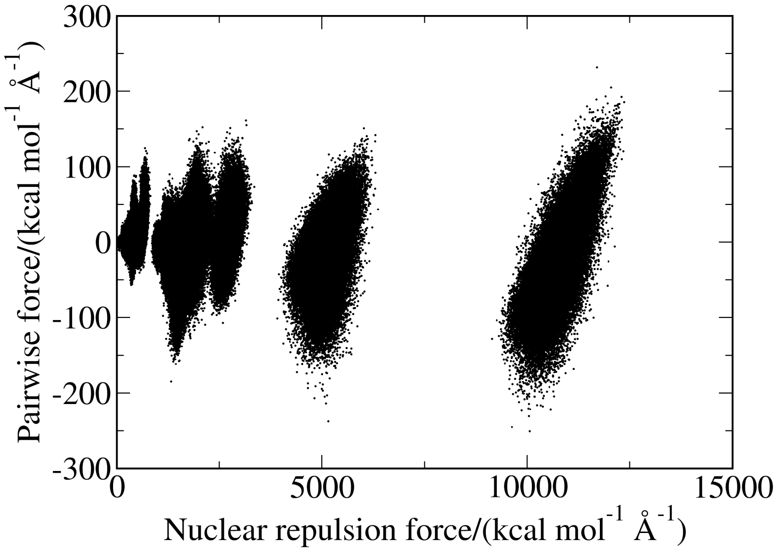


(a) Vanilla decomposition

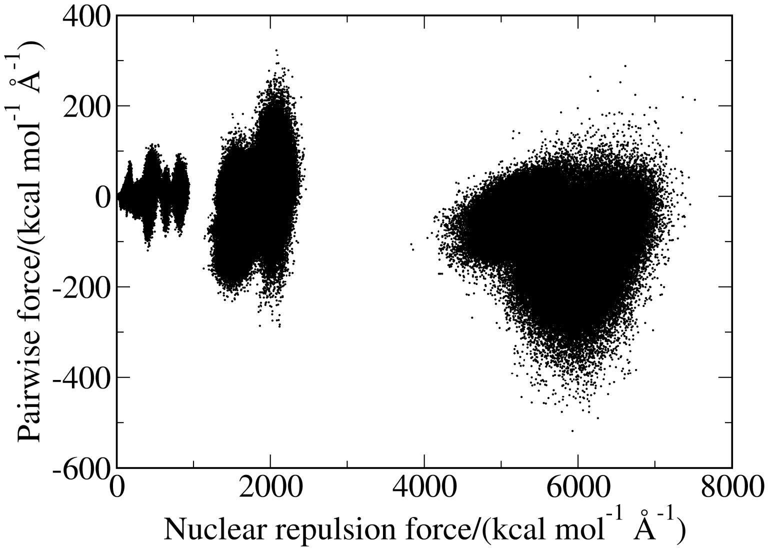


(b) biased decomposition

Figure 2.10: pairwise force for interactions to hydrogen in aspirin decomposed by the vanilla (a) and biased (b) method versus the nuclear repulsion force



(a) Malonaldehyde



(b) Toluene

Figure 2.11: pairwise forces decomposed using the biased decomposition scheme for a) malonaldehyde and b) toluene versus the nuclear repulsion force

2.5 Potential energy surface scans using DFT

2.5.1 Introduction

To further assess the continuity and behaviour of the decomposed forces we use the Gaussian software package.[45] We are able to stretch/compress bonds or rotate angles whilst minimizing the rest of the structure and calculate the forces [i]. This will allow us to see the effects of changing one internal coordinate on the pairwise forces that are generated across multiple structures (a scan).

2.5.2 Method

Different scans were performed using Gaussian version 9 are summarised in table 2.1. The forces were then extracted from the output files, converted into

Table 2.1: Gaussian scans performed for aspirin

scan type	Atoms involved	basis set	functional	# structures
C-C-C-O dihedral	1,6,8,10	6-31G(d)	B3LYP	12
C-O bond	7,13	6-31G(d)	UB3LYP	10
C=O bond	8,11	6-31G(d)	UB3LYP	8

units of kcal mol^{-1} Å^{-1} then decomposed using either the vanilla decomposition method or the biased vanilla decomposition method.

2.5.3 Results and discussion

Biasing the decomposition using the nuclear repulsion forces causes the pairwise interactions to have large values. This can be seen in the case for a single and double bond as well as the dihedral scan (figure 2.13) and is consistent with the observations made when looking at the decomposed forces on the aspirin dataset. We show the numerical labelling scheme for aspirin again in figure 2.12 to aid easy identification of the different interactions.

Although based on this observation one may quick to conclude the barrier to rotation of the dihedral or the stiffness of the bond changes depending on the biasing used, one must remember that the Cartesian forces are the same for both the biased and the vanilla method even if the pairwise interactions are different.

[i]Gaussian scans conducted by Dr N. Burton.

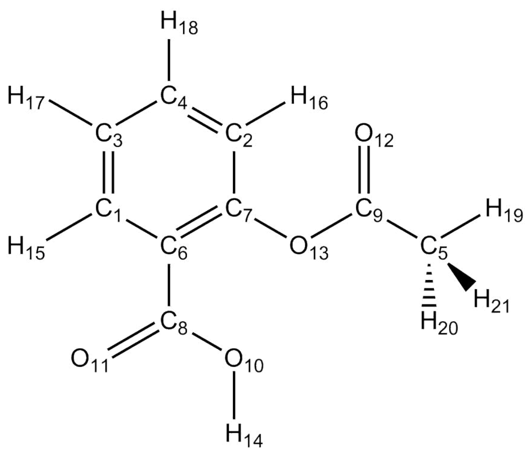
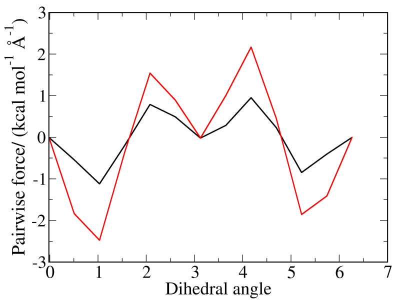


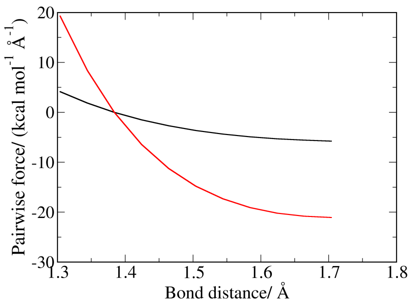
Figure 2.12: Aspirin numerical labelling scheme

These scans show that chemically sensible patterns in the pairwise forces may be seen when dihedrals are rotated and bonds are squeezed and stretched and that they are continuous between structures. We even observe that in both the biased and the vanilla decomposition cases the sensitivity to a change in bond distance for a double bond is greater than that for a single bond.

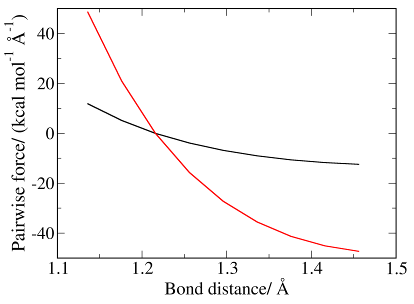
The continuous nature in the forces likely arises from the fact that the same criteria for a solution (the minimum norm) is employed for any given structure. This means that for a small change in the forces one would expect a correspondingly small change in the pairwise force.



(a) Scan of the C1,C6,C8,O10 dihedral



(b) C7 O13 CO single bond



(c) C8 O11 CO double bond

Figure 2.13: Pairwise forces decomposed using the vanilla method (black) and NRF biased (red) for the Gaussian scans plotted against a) the dihedral angle C1,C6,C8,O10 in radians b) the bond length C7 O13 and c) C8 O11 bond length for aspirin.

2.6 Network decomposed pairwise forces

2.6.1 Introduction

When the forces are decomposed using either the vanilla or biased decomposition method a specific solution is used from the infinite number of solutions to the set of simultaneous equations involving the pairwise forces. A solution is determined for each structure individually with no consideration for other similar conformers. Furthermore we are relying on a sensible choice of biasing in order to yield forces which are easy to fit across multiple structures by a neural network.

An alternative approach is to allow a neural network to decompose the forces from Cartesian to pairwise forces internally. This may be achieved by creating a neural network that makes predictions of pairwise forces for a structure, evaluates the error based on the Cartesian forces and then uses it to adjust predictions of pairwise forces.

The Keras library allows the use of 'lambda layers': these are layers containing a custom function using Keras defined mathematical routines, such as addition, subtraction, square root, et cetera. When using lambda layers, Keras is able to find the first derivative of the function and back-propagate the error in the same way it would do for a layer of neurons.

Unlike the vanilla or biased decomposition method which would allow for pairwise forces to be predicted independently, this approach requires that all pairwise interactions are fitted simultaneously.

The lambda layer was used to achieve our goal of predicting pairwise forces by fitting Cartesian forces. We created a lambda layer that calculated the dot product of the predicted pairwise forces and the matrix $\mathbf{e}$ in equation 2.5.

2.6.2 Method

A neural network with an architecture defined by figure 2.14 was constructed, where **NRF** is the lower triangle of the nuclear repulsion force matrix, NN1 is a series of fully connected layers ending with a layer of $\binom{N}{2}$ neurons. $\mathbf{y}$ is the network output from that layer which are the pairwise force predictions $\mathbf{f}$.

In the evaluation of a training performance we use the mean squared error (MSE) of the predictions of the Cartesian forces which is given by equation 2.8, where t is the target value y is the predicted value and n is the number of

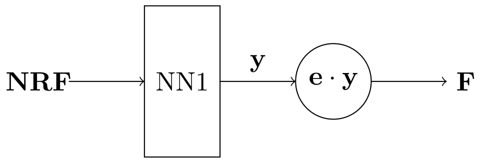


Figure 2.14: Scheme for generating network decomposed pairwise forces

samples, which for our tests will be the number of training samples multiplied by the number of atoms multiplied by three (three Cartesian forces on each atom over all the training data).

$$MSE = \frac{1}{n}\sum_{i}^{n}(t-y)^2 \tag{2.8}$$

For NN1 we used a hidden layer of 1000 neurons before the $\binom{N}{2}$ neurons, which were created using the Keras 'dense' function and had an activation of relu and linear respectively. The network was trained for a total of 33 977 epochs over a period of 14 days on the aspirin training data. A termination criterion of 500 epochs without improvement was implemented. However after 14 days it was observed that over the last 500 epochs the MSE had only improved by 0.0469 kcal mol^{-1} Å^{-1} and training was terminated. The final training RMSE was 0.619 kcal mol^{-1} Å^{-1} and the test RMSE was 0.649 kcal mol^{-1} Å^{-1}. The network was trained using the Adam optimizer with the default parameters for the version of keras employed.

We used the Keras library to obtain the output from the hidden layer that predicted the pairwise forces and saved the network predictions for the aspirin training data.

The trained network was then run on the structures from the Gaussian scans introduced in section 2.5 by using the structures as inputs.

2.6.3 Results and discussion

The network decomposed forces for aspirin look very different to the vanilla and biased vanilla decomposed forces (figure 2.15) showing a much clearer band structure and a greater range of pairwise forces. This is possibly due to the fact that unlike the vanilla and biased vanilla methods which are determined using the pseudo inverse giving the solution with the minimum norm, the network may pick any solution and the choice is informed by the other structures in the training data.

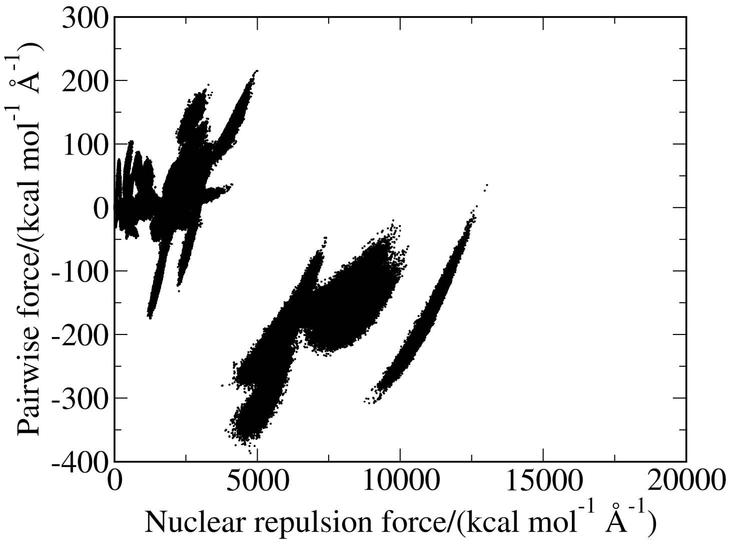


Figure 2.15: Nuclear repulsion force versus the network pairwise forces for aspirin.

We also see more of a relationship between the dihedral angle and the pairwise force compared with the other decomposition methods shown in figure 2.16.

The structures from the Gaussian scans show significantly larger errors for the network predictions compared to the training data, these are summarised in table 2.2. The error for the structures from the dihedral scan is much smaller than for the bond stretches as during the course of an MD simulation one would expect to sample this rotation quite well whereas large strains on bonding interactions will be less well sampled. This means the training data - constructed using MD will contain many examples of this rotation but very few (if any) of the bond stretches, which means the network's ability to fit those stretches well is likely poor, especially given the nature of the scans where one interaction is manipulated whilst the others minimized - something unlikely to be observed in the course of an MD simulation.

The predictions for the structures obtained from the Gaussian scans are quite different for the network decomposition (figure 2.18). First the magnitude

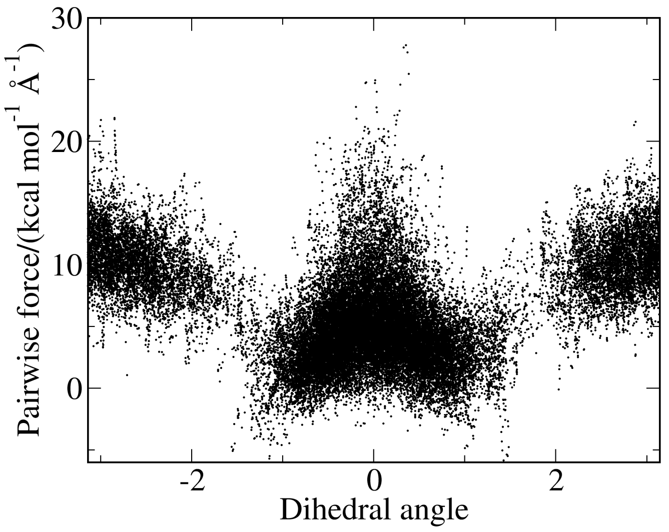


Figure 2.16: dihedral angle 1,6,8,10 in radians versus the network decomposed pairwise forces for aspirin.

Table 2.2: RMSE for network predictions of gaussian scans

Interaction	Error (RMSE mol^{-1} Å^{-1})
C8-O11 double bond	6.99
C7-O13 single bond	3.40
C1,C6,C8,O10 dihedral	0.249

of the forces are much larger for the single and double bonds; however this was expected given figure 2.15. We again show the numerical labelling scheme for aspirin to allow for easy identification of the different interactions in figure 2.17.

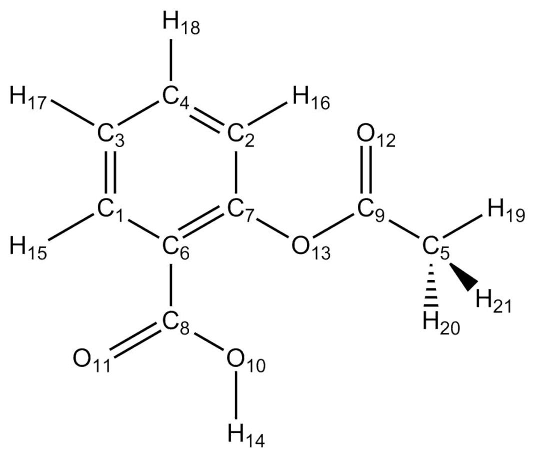
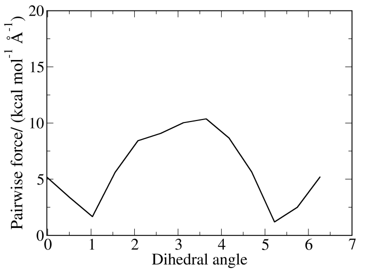
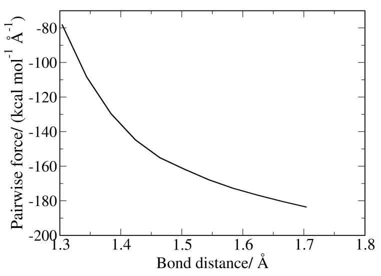


Figure 2.17: Aspirin numerical labelling scheme

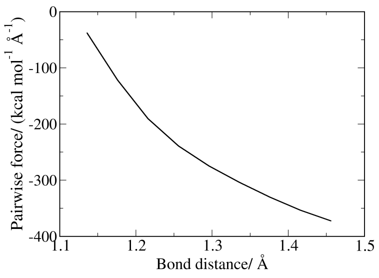
Unlike the vanilla and the biased methods which implicitly guarantee that when the forces acting on all the atoms are zero (equilibrium position), the decomposed forces are also zero, no such guarantee is offered by the neural network and we see in figure 2.18 that the pairwise forces are large even when the structure is at an equilibrium position (these forces will still cancel out to an overall small error in the prediction of the Cartesian forces).



(a) C1,C6,C8,O10 dihedral



(b) C7 O13 CO single bond



(c) C8 O11 CO double bond

Figure 2.18: Pairwise forces decomposed using the neural network for the Gaussian scan structures for a) the dihedral scan structures versus the dihedral angle in radians b) the C7 O13 single bond scan structures versus the bond length c) the C8 O11 CO double bond scan structures versus the bond length

2.7 Conclusions

As forces are a vector quantity and it is important that Newton's 3rd law is obeyed, we constructed a scheme which allows Cartesian forces to be simply broken down into pairwise interactions, allowing for predictions of forces which implicitly guarantee a rotational and translational sum of the forces to zero.

We showed that this scheme can be biased in order to increase the force on some interactions whilst decreasing it on others and we opted to bias it using the nuclear repulsion force, which will bias it dependent on atomic mass and distance. When this bias was applied we successfully showed that the magnitude of the hydrogen - hydrogen interactions was reduced.

Finally we showed that the decomposition can be incorporated into a machine learning routine which will make predictions of pairwise forces based on patterns it learns from the training data. We note that sensible trends such as having an overall force of zero does not implicitly guarantee that all the pairwise forces are also zero as it does for the vanilla scheme.

Further work could include investigating different biasing schemes and their effects on the calculated pairwise forces. Additionally work could be conducted to find a suitable scheme for reducing the $\binom{N}{2}$ interactions down to a smaller number.

We now have an appropriate output scheme for a machine learning model with good features as described and the next step is to attempt to fit the pairwise forces in our training sets and then use the models to evaluate the errors in the predictions of the test sets.

Chapter 3

Prediction of the pairwise forces

3.1 Introduction

In chapter 2 we introduced a method for creating an appropriate machine learning output for predicting Cartesian forces which implicitly guarantees obedience to Newton's third law. Now we attempt to fit these pairwise forces using machine learning so that the forces on new structures may be predicted which may be utilized in an MD simulation.

Two main considerations now are the input to the machine learning method, and the nature of the machine learning method. As for the input we choose to use the elements of the lower triangle of the N by N nuclear repulsion force matrix where each element is given by equation 2.7.

This is an appropriate choice of input as it is complete, meaning every possible structure is uniquely defined, it does not require too much computation to generate and has relatively low dimensionality. Also the nuclear repulsion force is a component of the total force and as we saw in chapter 2, it correlates quite well with some pairwise interactions.

Symmetry functions that described the environment around a pair of atoms were considered to generate inputs but multiple issues were encountered with this input scheme. First the number of symmetry functions to adequately describe the environment around a pair including angle and dihedral angle information proved quite large, making these a notionally inconvenient method.

The next issue is symmetry functions do not necessarily represent a complete input as there is not necessarily a one-to-one mapping between structure and set of symmetry functions. They are also potentially not a heuristically powerful method as there may not be a good correlation between the symmetry function values for an interaction and the force, therefore determining if a machine learning fit was performing poorly because of these factors or other ones is rather difficult to achieve. As such this input idea was abandoned and our focus instead was on improving the machine learning method to obtain the best fit possible.

For the machine learning method we first consider a deep neural network taking inspiration from more recent advances in image recognition namely the use of defined repeatable blocks and the use of shortcut connections.

3.2 Using neural networks to fit the pairwise forces

3.2.1 Introduction

We develop a series of neural networks for the prediction of pairwise forces, which we refer to as "pairwise force networks" (PFN).

The choice of network architecture is very important in determining the performance of a neural network. In recent years the addition of short cut connections between layers has enabled a significant increase in performance in tasks such as image recognition yielding a new class of neural networks called residual neural networks, called as such as each layer needs only to learn a small transformation of the inputs as any information not captured by a layer is still passed on to the next one, unlike standard neural networks where this information is lost.

Additionally in order to reduce the problem of architecture optimization and prevent tailoring of the architecture to a very specific problem, repeating blocks are often defined which are used to construct the overall architecture, we do the same using a standard block of 1000 fully connected neurons connected to a layer of $\binom{N}{2}$ neurons in all our networks which will henceforth be referred to as the standard block.

We also combine this idea of residual connections with the idea of building up the architecture during training drawing some inspiration from cascade correlation neural networks[23]. This partially removes the need to decide upon an architecture before beginning training. We start with a single standard block and optimize it until a termination criterion is met, then freeze the weights of this block, add another standard block as well as any shortcut connections, optimize the weights for that block before freezing them and repeating the process to build up an architecture of arbitrary size. We refer to this training scheme as "greedy blockwise training". Networks that utilize the standard block with this method of training are referred to as the PFN3 series, which consists of PFN3a,PFN3b etc.

In greedy blockwise training we begin with a single block training until a termination criterion is met (Figure 3.1)

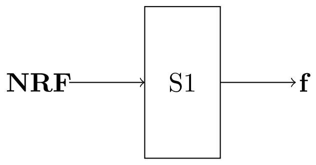


Figure 3.1: Step 1: train a single block until termination to predict the pairwise forces $\mathbf{f}$

After the termination criterion is met, the weights in the block 'S1' are frozen (are treated as constants and cannot be changed) to give the block 'S1*'. The output of this block is combined with the inputs by some function 'f' and used as the input to another S1 block, which is then trained until the termination criterion is met (Figure 3.2)

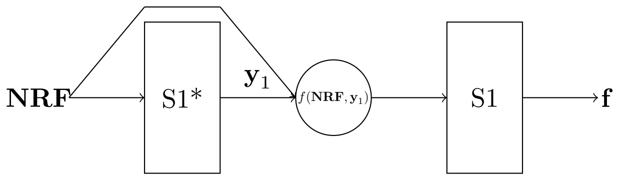


Figure 3.2: Step 2 combine the inputs with the output of the previously trained block ($\mathbf{y}_1$), add another block and train until termination

After the new S1 block has been trained the same process is repeated again, the weights are frozen to give S1*, the outputs combined with the inputs using the function f and another block is added which is trained until termination (Figure 3.3)

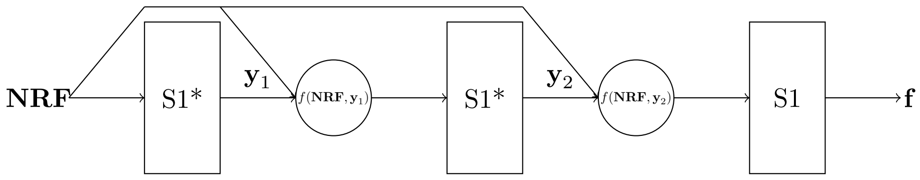

NRF
S1*
$\mathbf{y}_1$
$f(\mathbf{NRF}, \mathbf{y}_1)$
S1*
$\mathbf{y}_2$
$f(\mathbf{NRF}, \mathbf{y}_2)$
S1
$\mathbf{f}$

Figure 3.3: Step 3 combine the inputs with the output of the previously trained block $\mathbf{y}_2$, add another block and train until termination

We also train neural networks that use the standard block but have a fixed architecture, which will be referred to as standard training. Networks that are in this class will be referred to as the PFN2 series.

Finally we train networks with just a single block to act as a benchmark for comparison with the other networks and we call these the PFN1 series.

For most networks the pairwise force is used as an output, we also try fitting the pairwise force minus the pairwise nuclear repulsion force to see if this yields any improvement on the fit. We believed this may be the case as the nuclear repulsion force is a component in the total DFT force, and by fitting the pairwise forces minus the nuclear repulsion forces we will have a simpler problem to solve.

3.2.2 Method

As we are fitting the pairwise forces we use the mean squared error (equation 2.8) for predictions of the pairwise forces not the Cartesian forces as done in section 2.6. We use the Adam optimizer with the default parameters for the version of Keras used.

The different inputs and outputs for each architecture are summarised in table 3.1

Table 3.1: Different networks and their parameters

network ID	Input	Output	Training method	number of blocks
PFN3a	**NRF**	**f-NRF**	greedy block wise	1
PFN2a	**NRF**	**f-NRF**	fixed block number	2
PFN1a	**NRF**	**f**	fixed block number	1
PFN3b	**NRF**	**f**	greedy block wise	5
PFN1b	**NRF**	**f**	fixed block number	1
PFN2b	**NRF**	**f**	fixed block number	4
PFN2c	**NRF**	**f**	fixed block number	6

The training and architectural details of all the networks trained are as follows.

PFN3ao

PFN3ao used relu functions for the first layer of 1000 neurons in the standard block and linear functions in the second layer. It was optimized using a 500 epoch with no improvement termination criterion for each of the blocks added to the network. Training was terminated when additional blocks did not cause a reduction in the overall error. The combining function ($f(\mathbf{NRF}, \mathbf{y})$) used in the residual connections was addition. The inputs were the nuclear repulsion forces and the outputs the vanilla decomposed pairwise forces.

Training was terminated after the addition of 18 blocks as the rate of improvement in the MSE was too slow to ensure it was worthwhile continuing training.

PFN3a,PFN3b

These networks used sigmoid activations in both layers of the standard block (equation 3.1).

$$y = \frac{1}{1+e^{-x}} \tag{3.1}$$

As a consequence of this these networks can only predict an output between 0 and 1. PFN3b fitted the target values t of $t = \frac{\mathbf{f}}{2\times\max(|\mathbf{f}|)} + 0.5$ as its output where $\max(|\mathbf{f}|)$ is the largest pairwise force in the training data, meaning the outputs would range between 0 and 1 where 0 to 0.5 would represent negative numbers and 0.5 to 1 would represent positive numbers.

PFN3a fitted the aspiring training data where its target value t was defined by $t = \frac{(\mathbf{f}-\mathbf{NRF})-\max(\mathbf{f}-\mathbf{NRF})}{\max(|\mathbf{f}|)}$ meaning all predictions were for negative values only. This was done as the range went from 30.3 to -12,937.8 kcal mol^{-1} Å^{-1}, meaning having 0.5 to 1.0 represent positive numbers did not seem sensible.

The training algorithm for these networks may be summarised by algorithm 3.1.

PFN2a,PFN2b,PFN2c

As these networks used a fixed architecture consisting of multiple layers, the vanishing gradient problem becomes important. As such these networks use RELU functions in all the blocks except the final layer of the final block which

Algorithm 3.1 PFN3 series training algorithm

```
get $starttime$
load $coords, pairforces, atomtypes$
reshape $coords \leftarrow$ [# samples][# atoms][3]
reshape $pairforces \leftarrow$ [# samples][$\binom{\#\text{atoms}}{2}$]
declare $NRFinputs$ shape $\leftarrow$ [# samples][$\binom{\#\text{atoms}}{2}$]
$counter \leftarrow 0$
loop $i \leftarrow 1 : \#atoms$
    loop $j \leftarrow 0 : i$
        $NRFinputs[all][counter] \leftarrow$ calcNRF($coords[all][i], coords[all][j],$
                                          $atomtypes[i], atomtypes[j]$)
        $counter \leftarrow counter + 1$
    end loop
end loop
$NRFinputs \leftarrow NRFinputs/\max(NRFinputs)$
$pairforces \leftarrow$ Scale $pairforces$
$mc \leftarrow$ Keras checkpoint, save best to filename "best_model"
$es \leftarrow$ Keras early stopping, check loss, patience 500 epochs
$besterror \leftarrow 1000$
$maxdepth \leftarrow 6$
$network \leftarrow$ 1 input block
loop $i \leftarrow 0 : maxdepth$
    if $i > 0$ then
        add shortcut connections
    end if
    $network \leftarrow$ add block
    print network summary
    $network \leftarrow$ keras_train_net($maxepochs = 100k$, $callbacks = mc, es$)
    if evaluate($network$)$>$ $besterror$ then
        $besterror \leftarrow$ evaluate($network$)
        save $network$ filename "best_ever_model"
    end if
    if endsignal or $time - starttime$ >6 days then
        end program
    end if
    $network \leftarrow$ freezeweights($network$)
end loop
```

uses sigmoid functions, serving to limit the output between 0 and 1 as with the other networks.

PFN1a,PFN1b

PFN1a and PFN1b are the simplest networks constructed and consisted of a single block. Both networks predicted the decomposed force **f**, PFN1a used the standard block whereas PFN1b had 5000 neurons in the first layer instead of 1000.

Verification of the network architecture

The keras library provides a simple method for the analysis of the network architecture in the form of the *model.summary* method. This provides a table of network layers, their connections and the number of trainable and none trainable parameters. The number of weights was checked as were the connections using this method for the various networks. When training using the greedy blockwise method a summary was printed after the addition of each block.

PM6 calculations

To help assess the quality of the network fit we also calculate the PM6[6] forces for each structure in the aspirin training data using the Gaussian software package with a key word of "force" and then decompose these using the vanilla decomposition scheme.

3.2.3 Results and Discussion

Network training

A training curve shows how the error of network predictions of the training data changes as a function of the number of training epochs. For our plots we use the RMSE for this error. Generally this is a smooth curve which asymptotically approaches the final training error, for example the training curve for PFN2b (Figure 3.4) which shows an error of just under 2 kcal mol^{-1} Å^{-1} after 80,000 epochs.. Note although the error is technically still improving when training is terminated the rate of improvement is small enough to justify terminating training.

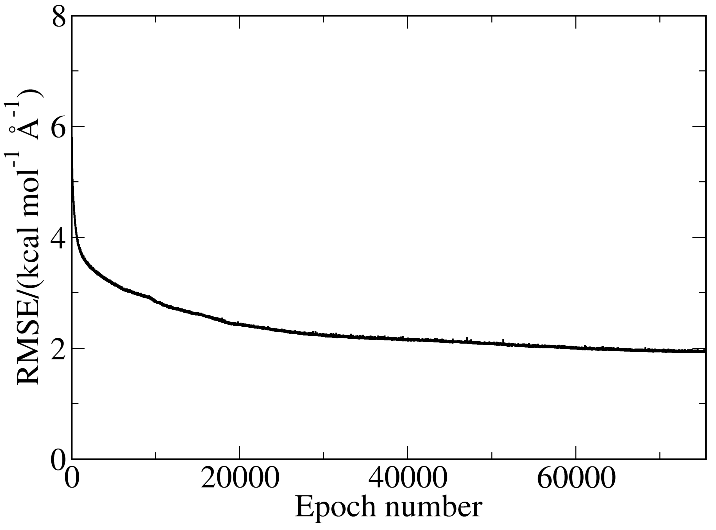


Figure 3.4: Training curve for network PFN2b

The training curve for the PFN3 series is rather different because of the greedy blockwise training regime. For example consider the training curve for network PFN3b fitting the aspirin dataset (Figure 3.5).

Multiple sharp spikes are observed which correspond to the addition of new blocks and are caused by the random nature of the weights which now determine the output of the network. Shortly after the addition of the block we notice a rapid decrease in the error. This likely occurs as the previous error may be attained by simply predicting the output of the previous block, which is used as an input, exactly.

The addition of a new block causes the error to not only match the previous error but improve upon it rapidly. This is most noticeable after the addition of the second block at around 1e+05 epochs and subsequent blocks give diminishing returns and overall we asymptotically approach the final error in the same way a fixed architecture would.

These observations lend credence to the idea that sequential blocks are capturing the information of the previous blocks and learning an improvement to the function constructed. Looking at the performance of the networks show that this method of learning produces superior results.

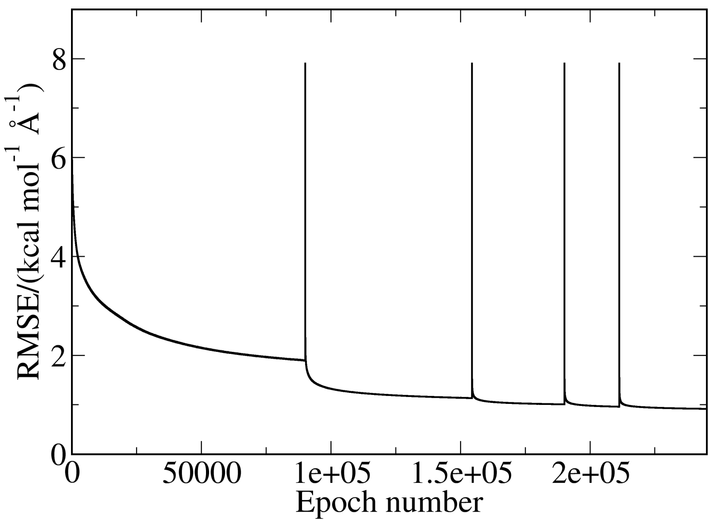


Figure 3.5: Training curve for network PFN3b for the aspirin training data.

Network performance

Our first network in the series, PFN3ao seemed to show poor performance for the aspirin dataset on average attaining an MSE of 5.70 kcal $\text{mol}^{-1}\text{Å}^{-1}$. The S curve for this network is shown in figure 3.6 and shows the predictions are of comparable accuracy to the PM6 predictions for the training data. Whereas the network predicts 45% of points in the test data to an RMSE below 1 kcal $\text{mol}^{-1}\text{Å}^{-1}$, there are a small number of points that have very high errors, the largest being over 2000 kcal $\text{mol}^{-1}\text{Å}^{-1}$, this serves to make the average error appear larger. Should a force be predicted with an error that high for even one frame in a molecular dynamics simulation the result would be catastrophic.

The ability to make such large predictions is of a consequence of the use of relu and linear functions which do not have a limit on the range of values they can predict. This in combination with the use of an addition function to combine a blocks output and shortcut connection allows for inputs to get larger from one layer to the next potentially exacerbating the issue. Consequently all future networks used concatenation where the block outputs and the net inputs are simply joined to give a new input of length $2 \times \binom{N}{2}$ as their method of

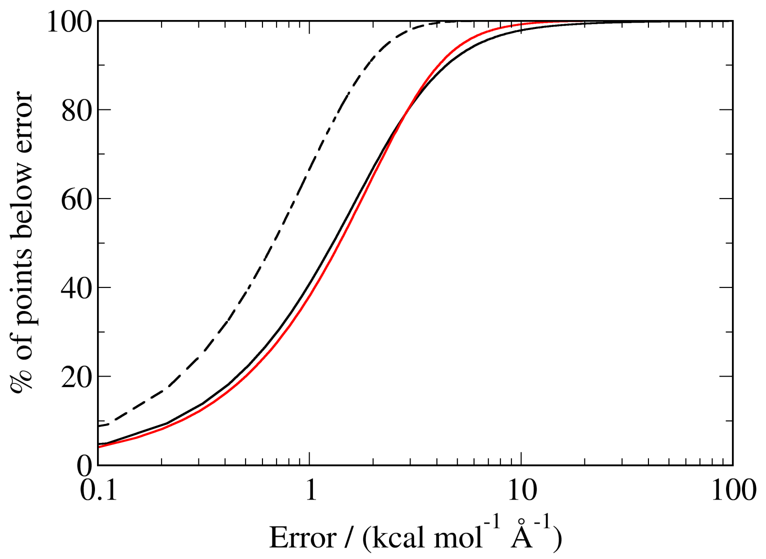


Figure 3.6: S curve for PFN3ao, showing the the error in the pairwise training predictions (dashed line), the test predictions (solid black) and the S curve for the PM6 training predictions (red) for the aspirin dataset.

combining the network outputs rather than addition. This ensures that there is no build up of value between layers.

As the data we fit was assumed to be inherently noise free being acquired from deterministic calculations, we reasoned that ideally every point should be fitted exactly. However unlike methods such as fitting radial basis functions we do not implicitly guarantee this with a neural network. Because of this assumed noise free nature of the data being fitted, we opted not to use any sort of cross validation to terminate training early. We do however note that as the training and test errors for our models are fairly similar over-fitting likely did not occur to a significant degree.

The performance of the networks in the series that followed PFN3ao for predictions of the Cartesian forces in the aspirin test set is summarised in table 3.2. The mean absolute error (MAE) and RMSE for the magnitude of the force is also given in this table by considering $||\mathbf{F}_p|| - ||\mathbf{F}_a||$, where $\mathbf{F}_p$ are the predicted Cartesian forces and $\mathbf{F}_a$ are the actual Cartesian forces of an atom to yield a vector of length equal to the number of test set samples multiplied by the number of atoms. The angular MAE is also presented which is calulated by considering $\arccos\left(\frac{\mathbf{F}_p}{||\mathbf{F}_p||} \cdot \frac{\mathbf{F}_a}{||\mathbf{F}_a||}\right)$ for each atom. These metrics are presented as they were included in the analysis of the GDML method[43].

Table 3.2: Accuracy of the different networks based on predictions of the Cartesian forces

Network name	Total MAE	Total RMSE	Magnitude MAE	Magnitude RMSE	Angle MAE	Angle RMSE
PFN3a	5.51	7.33	5.83	7.78	0.0898	0.125
PFN2a	4.83	6.50	5.28	7.17	0.0758	0.1070
PFN1a	1.91	2.59	1.97	2.68	0.0303	0.0425
PFN3b	1.72	2.39	1.96	2.80	0.0268	0.0402
PFN1b	1.85	2.52	1.92	2.60	0.0296	0.0421
PFN2b	2.98	4.02	3.19	4.33	0.0463	0.0662
PFN2c	3.41	4.61	3.64	4.97	0.0530	0.074

In the series of networks that followed after PFN3ao we note that PFN3a performs particularly poorly achieving an MAE of 5.51 kcal mol^{-1} Å^{-1}. Training terminated after only the addition of a single block and the network being too small was suspected as a reason for the poor performance. PFN2a, consisting of two blocks and a fixed architecture did show some improvement attaining an MAE of 4.83 kcal mol^{-1} Å^{-1}. However network PFN1a, which had only a single block but fit the value of $\mathbf{f}$ and not $\mathbf{f} - \mathbf{NRF}$ improved significantly on both of these networks attaining an MAE of 1.91 kcal mol^{-1} Å^{-1}. This would suggest the issue lies with the output scheme of using $\mathbf{f} - \mathbf{NRF}$ rather than the architecture as PFN1a was a much simpler network yet improved on the performance.

The assumption that the values of the NRF form a component of the total force value and therefore subtracting the pairwise NRF from the pairwise decomposed force would yield a better fit is incorrect. This is because of the nature of the pairwise forces, which are not necessarily physically meaningful. This can be illustrated by calculating the total Cartesian nuclear repulsion force on each atom which is the sum of the pairwise terms given by equation 3.2), then decomposing these using the vanilla scheme, then plotting this against the calculated pairwise forces given by equation 2.7 (Figure 3.7).

$$\mathbf{F}_A^{NRF} = \sum_{i,A\neq i}^{N} \frac{q_i q_A}{|\mathbf{R}_A - \mathbf{R}_i|^2} \tag{3.2}$$

From Figure 3.7 we are able to see a poor correlation between the decomposed NRF and the calculated NRF. This even holds true for the biased decomposition scheme, where even though a greater correlation is observed, the decomposition does not yield the exact pairwise terms Figure 3.8.

We conclude that subtracting out the values of the nuclear repulsion forces from the pairwise forces yields, represents an arbitrary change to the output and actually makes the problem more difficult to fit, not less. All the networks in the rest of the series therefore fitted the value of $\mathbf{f}$.

We repeated PFN3a but outputting the value of $\mathbf{f}$ instead and we note a significant improvement on the error obtained compared with PFN3a as well as an increase in the number of blocks added before training termination. The S curve for network PFN3b also shows significant improvement from PFN3a (Figure 3.9) predicting 68.8% of test points to an error below 1 kcal mol^{-1} Å^{-1}.

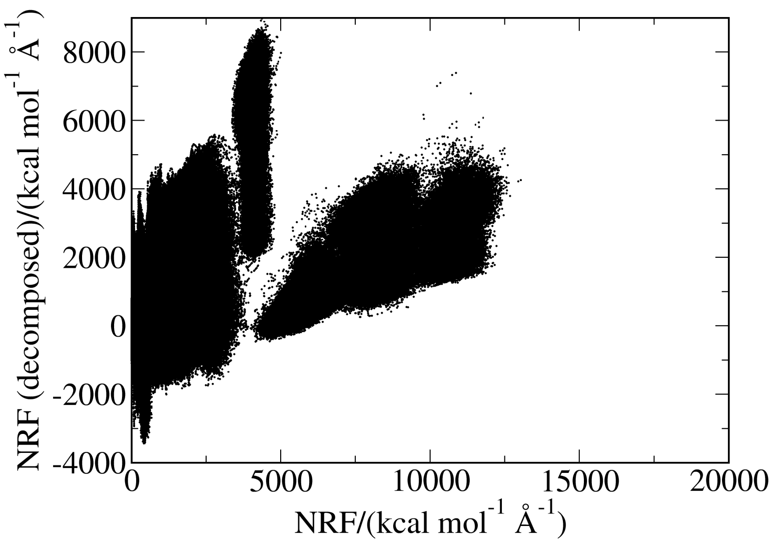


Figure 3.7: The decomposed total atomic NRF using equation 3.2 into pairwise forces using the vanilla scheme versus the calculated pairwise NRF using equation 2.7

Although the MAE for network PFN1a (1.91 kcal mol^{-1} Å^{-1}) and PFN3b (1.72 kcal mol^{-1} Å^{-1}) is comparable, plotting the S curves for the test error on the same graph shows that PFN3b had higher percentages of points below low errors (Figure 3.10) as PFN1a only predicted 48% of points to an accuracy of 1 kcal mol^{-1} Å^{-1}. This may be due to the 'herd effect' described by Fahlman and Lebiere[23].

The herd effect is a consequence of neurons training independently from one another in a neural network which may cause them to all be sensitive to certain regions of inputs at the expense of others. Essentially as the network is optimized the neurons will try to be more sensitive to regions of high error changing their weights accordingly, this will cause new regions of high error to appear which the neurons will then try be sensitive to and so on. Consequently whereas some regions will be fitted well, this will be done at the expense of others. With PFN3b the weights of previous layers are fixed meaning that those layers will never change the regions they are sensitive to and thus produce

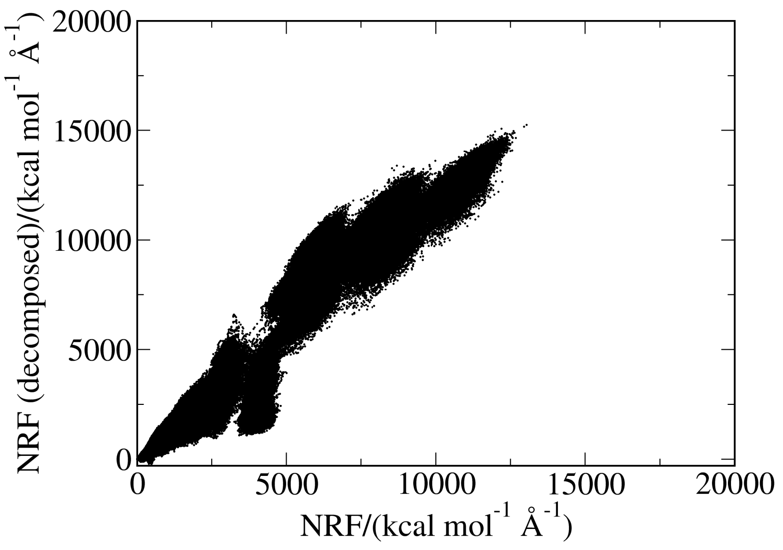


Figure 3.8: The biased decomposed pairwise NRF versus the calculated NRF

an overall better fit. By freezing weights of some neurons during training (as we do for PFN3b after a block has been trained) the herd effect can be effectively mitigated.

In order to compare our greedy block-wise scheme to networks with fixed architectures we constructed networks PFN2a and PFN2b. Whilst this isn't an exact comparison as the activation function is different in the hidden layers, without this change we would expect poor performance. PFN3b outperformed both of these networks showing our training scheme has merit.

We also constructed PFN1b to show that increasing the number of neurons in the standard block does not have a large impact on the performance we attain, comparing PFN1b to PFN1a shows this is the case.

The better performance of PFN3b compared with the other networks can be seen clearly in the values of L1 (Table 3.3) where the L1 value is calculated using equation 3.3

$$\frac{\#(\sqrt{(\mathbf{f}_p - \mathbf{f}_a)^2} \leq 1)}{\#\mathbf{f}} \times 100 \tag{3.3}$$

where the value of $\#\mathbf{f}$ is the number of pairwise interactions in a structure multiplied by the number of structures in the set. PFN3b attains an L1 value of

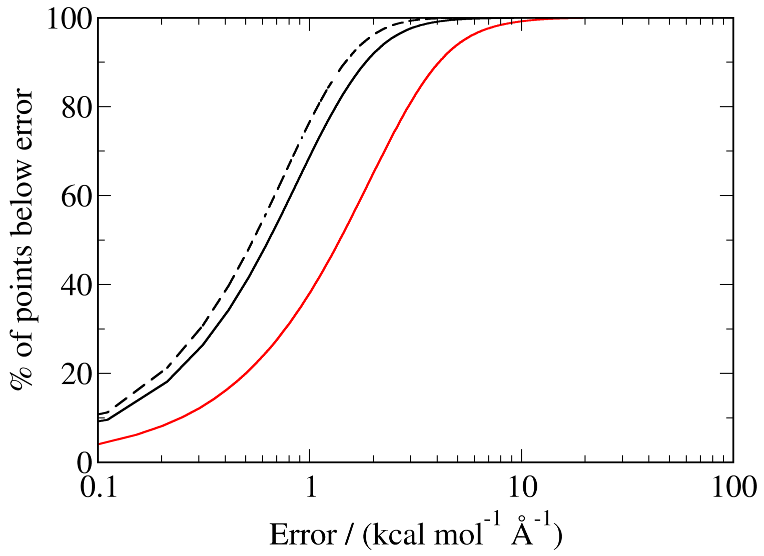


Figure 3.9: S curve for PFN3b, showing the the error in the pairwise training predictions (dashed line), the test predictions (solid black) and the S curve for the PM6 training predictions (red) for the aspirin dataset.

68.8% significantly higher than PFN1a and PFN1b which achieved an L1 value of 48.0% and 49.0% respectively, this is still an improvement on PFN3a which only achieved a value of 25.8%. The increased performance of PFN3b over the

Table 3.3: The values of L1 for the different networks

network name	L1 test (L1 train)
PFN3a	25.8 (26.0)
PFN2a	31.0 (32.0)
PFN1a	48.0 (49.5)
PFN3b	68.8 (76.7)
PFN1b	49.0 (51.2)
PFN2b	45.2 (46.3)
PFN2c	37.6 (38.2)

fixed architectures in the PFN2 series is possibly due to PFN3b mitigating the herd effect.

Given the better performance of PFN3b compared with the other networks we use this network to fit both malonaldehyde and toluene, the results for the 3

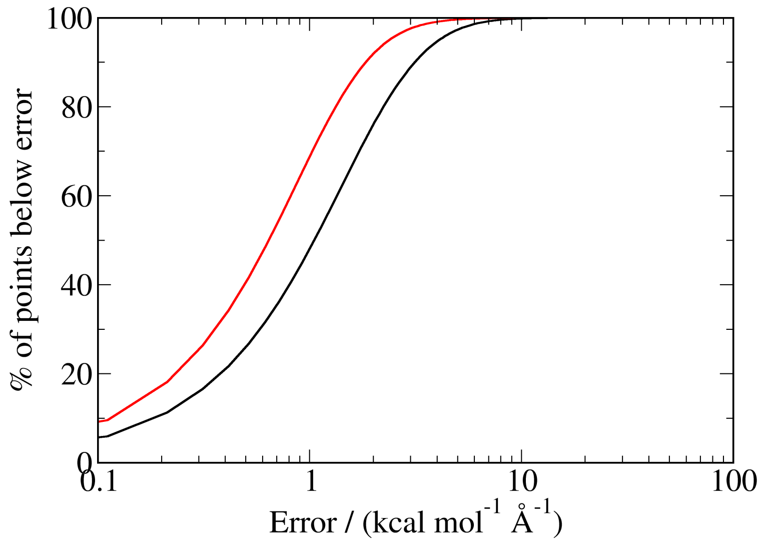
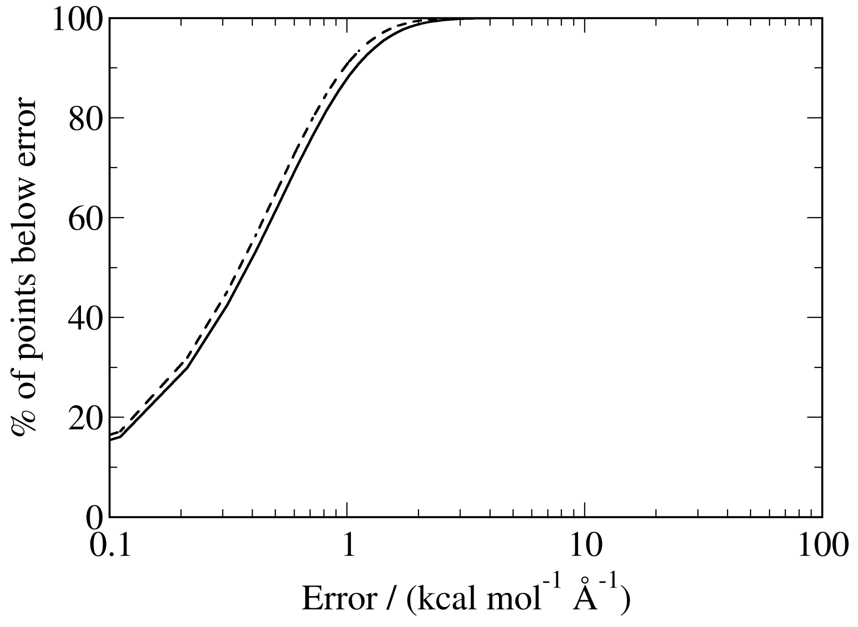
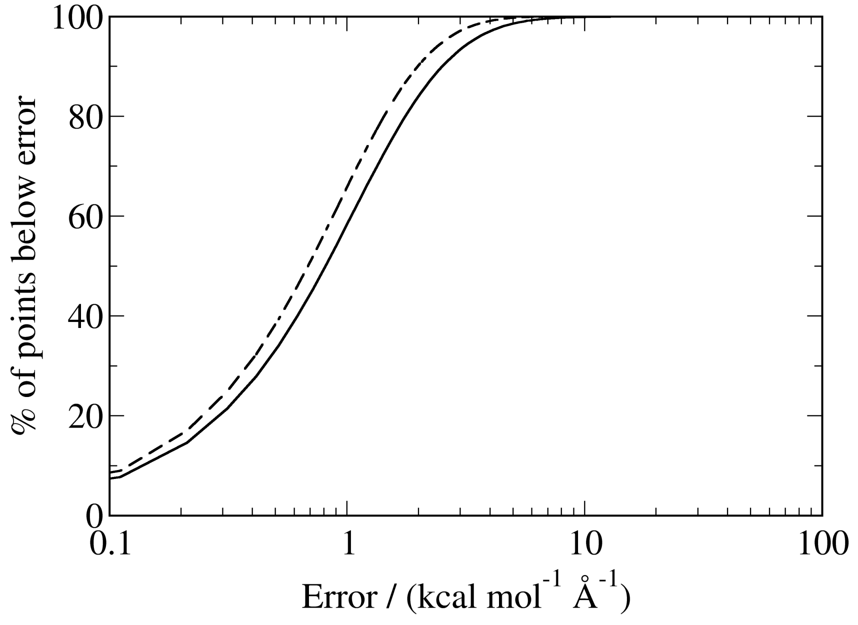


Figure 3.10: S curve for test error of PFN1a (black) and PFN3b (red) for predictions of the aspirin test set.

molecules are summarised in table 3.4. The S curves also show a similar profile Figure 3.11.



(a) malonaldehyde



(b) toluene

Figure 3.11: S curves for the pairwise predictions of the test set (solid line) and training predictions (dashed line) by network PFN3b for malonaldehyde (a) and toluene (b).

The s curve for the training predictions and the test predictions are very similar for these two datasets, especially for malonaldehyde. This indicates that the training data and the test data contain very similar structures which is not unexpected given toluene has little flexibility and malonaldehyde has only a few degrees of freedom when compared with aspirin.

Table 3.4: The performance of network PFN3b fitting the aspirin, toluene and malonaldehyde datasets.

Molecule	aspirin	malonaldehyde	toluene
Total MAE	1.72	0.54	1.25
Toal RMSE	2.39	0.72	1.71
Magnitude MAE	1.96	0.58	1.44
Magnitude RMSE	2.8	0.77	1.97
Angle MAE	0.0268	0.0099	0.0200
Angle RMSE	0.0402	0.0147	0.0300
L1 test (L1 train)	68.8 (76.7)	87.9 (90.8)	58.4 (65.8)

We next consider which pairwise interactions are predicted well and which are poorly for aspirin. This is important as it can serve to highlight types of interactions our method performs poorly on. As mentioned in chapter 2 it is more important to ensure that pairwise interactions involving light atoms such as hydrogen atoms are predicted to a higher level of accuracy as errors in these will have a larger impact on the simulation. We calculate the L1 values for each pairwise interaction independently then plot them for network PFN3b (Figure 3.12).

We note there are several areas of poorer performance which appear to be common to the other networks for example the PFN2 series (Figure 3.13). We combine the L1 graphs for these networks into a single graph to better illustrate the similarity in the profile in figure 3.14.

One region of higher error runs from approximately interaction number 150 until 209, which corresponds to interactions between H18 and H17 and H18 with H16 as well as interactions to the methyl hydrogen atoms in aspirin. As for those interactions with H18 no adequate explanation can be offered, an investigation did note no correlation between the NRF of these interactions and the decomposed force.

For the interactions with the methyl hydrogens the poor performance may be attributed to the input scheme not capturing permutational invariance. This means if the methyl group was rotated to a chemically equivalent position the

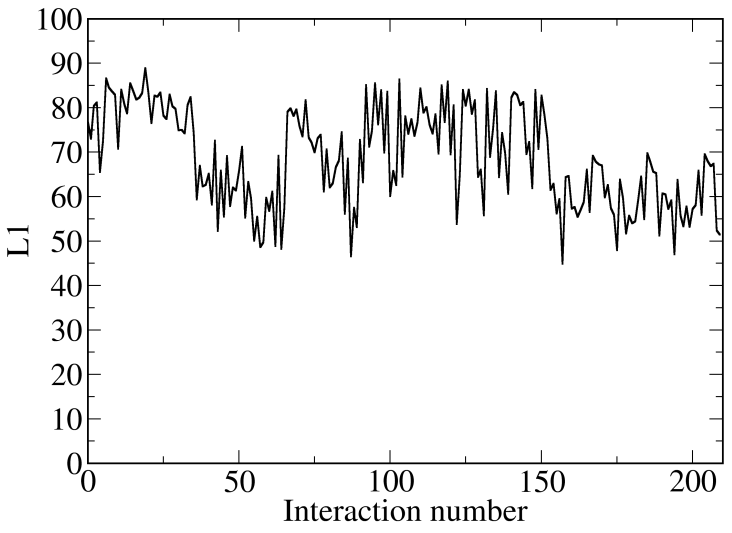
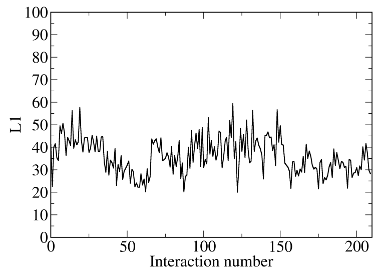
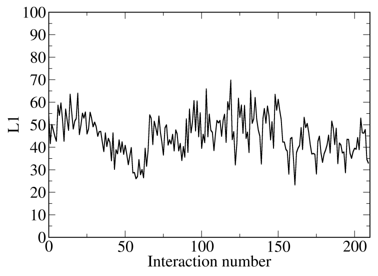


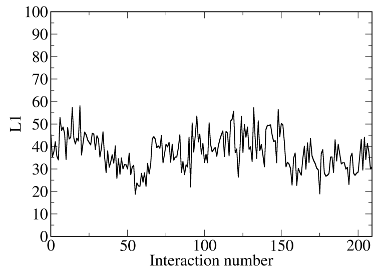
Figure 3.12: The L1 value for the different interactions in aspirin predicted by PFN3b. Interactions 0 to 35 correspond to carbon-carbon interactions. 36 to 77 correspond to oxygen-oxygen and oxygen-carbon interactions and 78-209 correspond to all interactions to hydrogen atoms.



(a) PFN2a



(b) PFN2b



(c) PFN2c

Figure 3.13: L1 plots for the PFN2 series for (a) PFN2a, (b) PFN2b and (c) PFN2c predicting the aspirin test data.Interactions 0 to 35 correspond to carbon-carbon interactions. 36 to 77 correspond to oxygen-oxygen and oxygen-carbon interactions and 78-209 correspond to all interactions to hydrogen atoms.

labels on each hydrogen atom will not update accordingly. This, combined with the flexibility on the methyl group presents an increased challenge to fitting. To ensure a good fit the dataset would have to include training points along a full 360° rotation of this group, instead of 109.5° if the labels were updated to account for permutational invariance.

Another common region of poor performance is interactions 36 to 65 which correspond to interactions with the oxygen atoms 10,11 and 12. Perhaps these are modelled less accurately as the behaviour of oxygen atoms is more complicated due to the presence of lone pairs compared with carbon and hydrogen atoms.

We also note in the L1 plot that symmetrical interactions show repeating patterns in the L1 plot showing equal difficulty in predicting these interactions. For example consider interactions to the methyl hydrogens in aspirin, the accu-

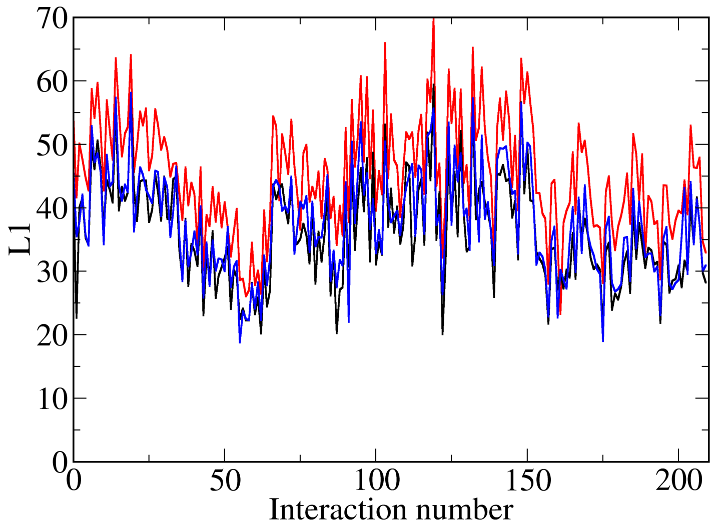


Figure 3.14: The L1 value for the different interactions in aspirin predicted by PFN2a (black), PFN2b (red) and PFN2c (blue). Interactions 0 to 35 correspond to carbon-carbon interactions. 36 to 77 correspond to oxygen-oxygen and oxygen-carbon interactions and 78-209 correspond to all interactions to hydrogen atoms.

racy of the predictions for each of these shows a very similar pattern which can be seen in figure 3.15.

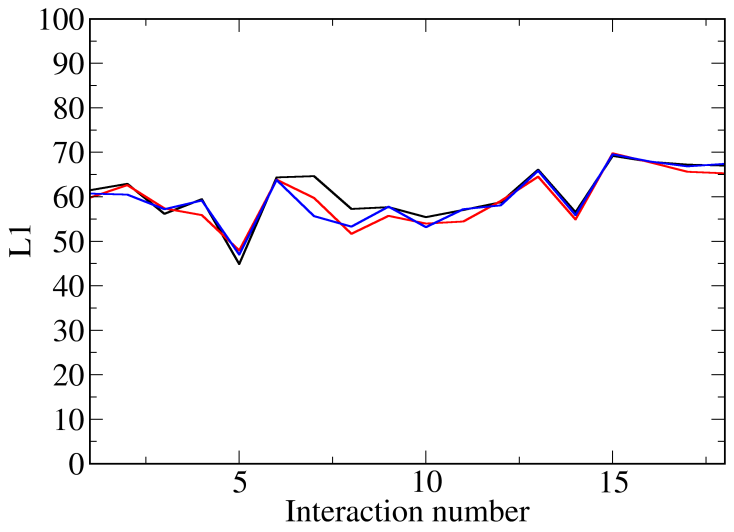


Figure 3.15: L1 values for H19 (black), H20 (red) and H21 (blue) interacting with the other 18 atoms in the aspirin test set as predicted by PFN3b (interactions between these hydrogens are excluded). The interaction number corresponds to the atom number minus 1, for example interaction 0 corresponds to the interaction to C1.

Figure 3.15 could indicate that some environments of the methyl hydrogens are harder to fit than others for certain interactions, for example the interactions with C7 show significant difference between the hydrogens. Figure 3.16 which shows that the methyl hydrogens are sampled with different frequencies for the dihedral angle (methyl hydrogen, C5 C9 O13), meaning they are in different environments with different frequencies, this supports the theory that some environments are harder to fit than others, had the sampling been even we would have expected similar performance.

Applying PFN3b to the biased decomposed forces for aspirin we obtain a rather different result. The S curve is significantly shallower suggesting a greater range in errors, likely caused by the extra complexity of fitting certain interactions figure 3.17.

The L1 plot (Figure 3.18) looks very different to that for pairwise forces

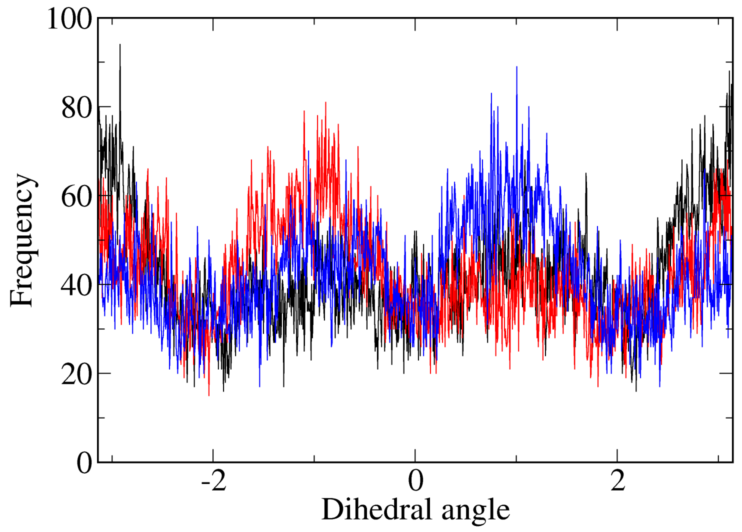


Figure 3.16: histogram frequency of values of dihedral angle (HX C5 C9 O13) where HX is H19 (black), H20 (red) and H21 (blue) in the aspirin training data

generated using the vanilla method (Figure 3.9) and confirms the previous conclusion that some interactions have been made easier to fit whereas others have become considerably harder as a consequence of biasing the force decomposition scheme.

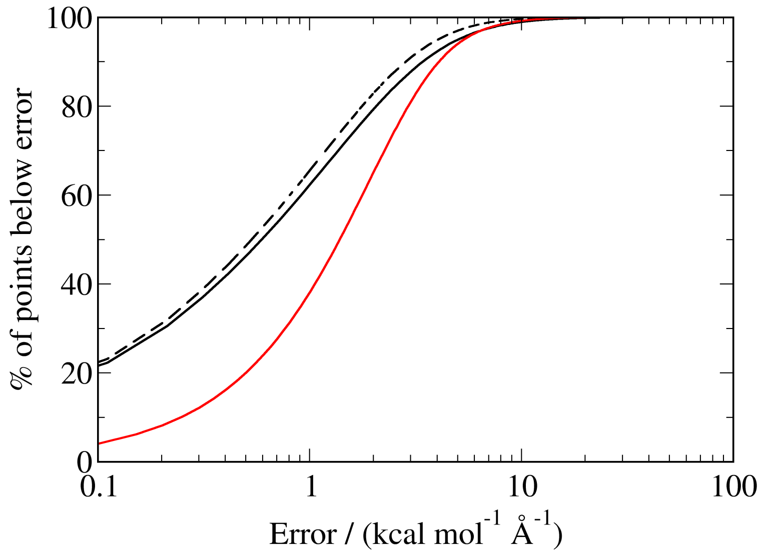


Figure 3.17: S curve for errors in the predictions of pairwise forces for the biased decomposed forces predicted for the aspirin test set by PFN3b (solid black line) the training set (dashed black line) and PM6 predictions decomposed using the vanilla method compared with aspirin training set vanilla decomposed forces (red line)

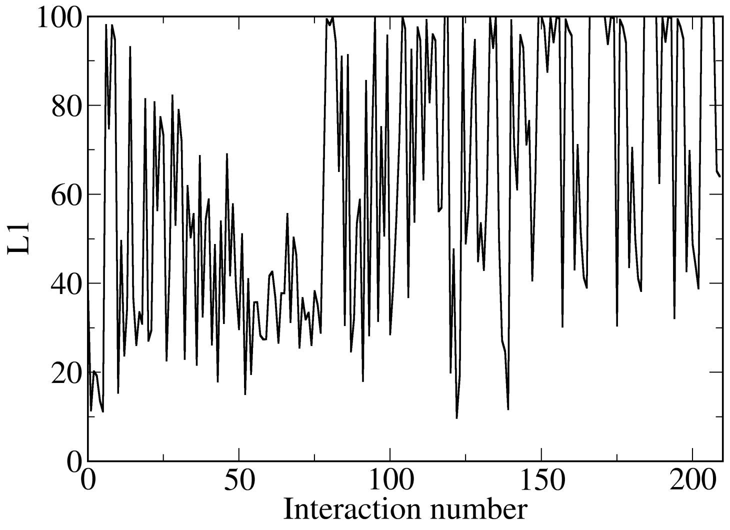


Figure 3.18: L1 plot for predictions of pairwise forces decomposed using the biased method for aspirin fitted by PFN3b

We compare PFN3b for aspirin with the network decomposed forces presented in section 2.6 for predictions of the test set. As there are no reference decomposed forces to compare to for the network decomposed forces we instead construct an S curve for the total Cartesian force (Figure 3.19).

We notice a very similar performance between the PFN3b fitting the biased decomposed forces and PFN3b fitting the vanilla decomposed forces when the error is evaluated for the total force. This suggests that biasing the decomposition scheme serves only to impact the total error minimally (Table 3.5). The total MAE for PFN3b fitting the vanilla forces was 1.72 kcal mol^{-1} Å^{-1}

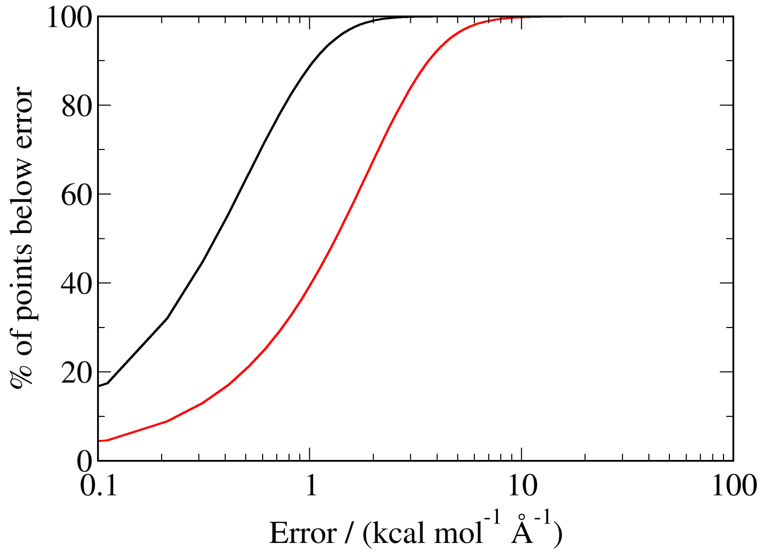


Figure 3.19: S curve for total forces for the network decomposed forces (black) and PFN3b (red)

It can be seen in figure 3.19 that the network decomposed forces appear to significantly outperform PFN3b, this is likely due to the flexibility of the former to select the pairwise forces allowing it to produce a better fit. We also compare the performance of the two networks in table 3.5.

Although one could conclude from this that decomposing the forces internally within the network yields the best performance, it is important to consider how well the simulation runs, which will be discussed in chapter 4.

In order to prove the reproducibility of our fits we reran PFN3b for aspirin and obtained an RMSE of 2.46 kcal mol^{-1} Å^{-1} for the test data which is very similar to the RMSE obtained for the first run of 2.39 kcal mol^{-1} Å^{-1}. Whereas it may be initially surprising that a neural network with millions of parameters to optimize fitting high dimensional data fitted using a stochastic process yields such similar results. This is consistent with the conclusions of the work conducted by Choromanska et al. who showed that for neural networks with large numbers of parameters the error surface consists of many local minima with similar errors, meaning we would expect the network to optimize to similar errors.[46]

Table 3.5: The performance of network PFN3b, the network decomposed forces and PFN3b fitting the biased decomposed forces for aspirin.

Method	Total MAE	Total RMSE	Magnitude MAE	Magnitude RMSE	Angle MAE	Angle RMSE
PFN3b (vanilla decomposed forces)	1.72	2.39	1.96	2.80	0.0268	0.0402
network decomposed forces	0.478	0.649	0.539	0.729	0.00732	0.0116
PFN3b (biased decomposed forces)	1.82	2.44	2.09	2.81	0.00281	0.0414

3.3 Radial basis function fits

3.3.1 Introduction

In this section we attempt to use radial basis function (RBF) interpolation to fit the decomposed forces. We construct a different model for each of the pairwise forces and predict them independently of one another. We use the same input as the neural networks, the pairwise nuclear repulsion forces.

Radial basis function interpolation is conducted by first constructing an Npts by Npts matrix where Npts is the number of training points and each element is a function g of the distance r between the training points. In our example the distance between training points is the euclidean distance between their respective nuclear repulsion forces input.

This matrix is dotted by a vector of weights $\mathbf{w}$ of length Npts to give a vector of predictions of $\mathbf{f}$ of equal length, with 1 value per training point. Consider the case of the pairwise interaction between atoms A and B, we would construct equation 3.4

$$\begin{bmatrix} 0 & g^{AB}(r_{1,2}) & g^{AB}(r_{1,3}) & \cdots & g^{AB}(r_{1,N}) \\ g^{AB}(r_{2,1}) & 0 & g^{AB}(r_{2,3}) & \cdots & g^{AB}(r_{2,N}) \\ g^{AB}(r_{3,1}) & g^{AB}(r_{3,2}) & 0 & \cdots & g^{AB}(r_{3,N}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g^{AB}(r_{N,1}) & g^{AB}(r_{N,2}) & g^{AB}(r_{N,3}) & \cdots & 0 \end{bmatrix} \begin{bmatrix} w_1^{AB} \\ w_2^{AB} \\ w_3^{AB} \\ \vdots \\ w_N^{AB} \end{bmatrix} = \begin{bmatrix} f_1^{AB} \\ f_2^{AB} \\ f_3^{AB} \\ \vdots \\ f_N^{AB} \end{bmatrix} \tag{3.4}$$

and solve for $\mathbf{w}^{AB}$.

As the matrix of functions of distance is square, the solution to the set of simultaneous equations presented in 3.4 can be found using the inverse of this matrix to obtain a vector of weights $\mathbf{w}^{AB}$. Once this is achieved this vector of weights may be applied to a new structure to yield predictions for that structure.

This is achieved by taking a new structure (m) and finding its distances to each training structure, applying the function of this distance g and taking the dot product with the weights, (see equation 3.5).

$$f^{AB}_{prediction} = \begin{bmatrix} g^{AB}(r_{1,m}) \\ g^{AB}(r_{2,m}) \\ g^{AB}(r_{3,m}) \\ \vdots \\ g^{AB}(r_{N,m}) \end{bmatrix} \begin{bmatrix} w^{AB}_1 \\ w^{AB}_2 \\ w^{AB}_3 \\ \vdots \\ w^{AB}_N \end{bmatrix} \tag{3.5}$$

Unlike neural networks, RBF interpolation implicitly guarantees that the function fits every training point exactly, assuming that for any training point the measurement is exact, this would have been a desirable quality if the data we are fitting was actually noise free. It is also a deterministic method, meaning the same solution will be obtained every time it is run and the end point of the optimization is well defined.

Neural networks are a function of the entire input. RBF interpolation is a function of some distance metric between the different input structures. This is problematic as input design becomes more important as the distance metric must correlate well with the output in order to obtain useful predictions away from training points. RBF interpolation guarantees a continuous fit if the function g is continuous, neural networks can offer the same guarantee in the case where all neurons use a continuous activation function (such as a sigmoid function).

For our approach, the distance between structures is calculated as the Euclidean distance between the nuclear repulsion force inputs given by equation 3.6.

$$g(r_{ij}) = ||\mathbf{NRF}_i - \mathbf{NRF}_j|| \tag{3.6}$$

For the function we begin by using the cube of this distance (method RBF1). This was selected for multiple reasons, first one does not need to worry about selecting appropriate parameters for the width as you would with a Gaussian function. Secondly as one moves away from the training points the output of the function becomes larger which may serve to move the simulation towards these points and help prevent extrapolation.

We also applied a Gaussian function as the radial basis function to the previously mentioned distance metric, called RBF2. In order to avoid the ‘bed of nails’ interpolant where there is a small region of sensitivity near training points and values of zero everywhere else, it was important to ensure there was sufficient overlap between Gaussian functions.

3.3.2 Method

We constructed a radial basis function interpolation program using python's numpy libraries numpy.solve method to calculate the weights. The script was run in parallel for every pairwise interaction. The nuclear repulsion force inputs were identical for each run so were calculated prior to fitting each model. The resulting weights were then used to make predictions for the test sets.

The algorithm used for conducting an RBF fit for a particular interaction is given by algorithm 3.2

Algorithm 3.2 RBF fitting

```
    load $NRFinputs$ shape $\leftarrow$ [# samples][$\binom{\#\text{atoms}}{2}$]
    get $jobnum$ from terminal input
    load $pairforces[jobnum]$ shape $\leftarrow$ [# samples]
    Declare $distmat$ shape $\leftarrow$ [# samples][# samples]
 5: loop $i \leftarrow 0 : \#samples$
        loop $j \leftarrow 0 : \#samples$
            $distmat[i][j] \leftarrow \mathrm{f}(||NRFinputs[i][all] - NRFinputs[j][all]||)$
        end loop
    end loop
10: $constants \leftarrow \text{numpy.solve}(distmat, pairforces)$
    save $constants$
```

After the constants are obtained by algorithm 3.2 predictions on a test set can be made using algorithm 3.3

Algorithm 3.3 RBF run on test

```
load $testNRF$ shape $\leftarrow$ [# testsamples][$\binom{\#\text{atoms}}{2}$]
get $jobnum$ from terminal input
load $constants$ shape $\leftarrow$ [# samples]
load $NRFinputs$ shape $\leftarrow$ [# samples][$\binom{\#\text{atoms}}{2}$]
Declare $testresult$ shape $\leftarrow$ [# testsamples]
Declare $distmat$ shape $\leftarrow$ [# samples]
loop $i \leftarrow 0 : \#testsamples$
    loop $j \leftarrow 0 : \#samples$
        $distmat[j] \leftarrow \mathrm{f}(||testNRF[i][all] - NRFinputs[j][all]||)$
    end loop
    $testresult[i] \leftarrow distmat \cdot constants$
end loop
save $constants$
```

Variogram plots

In order to assess the performance of the RBF method we construct variograms for each interaction which are fitted by writing python code using the scikit gstat[47] library. From this fit we obtain the effective range of the different interactions.

RBF1

RBF1 fitted the distance r between each NRF input matrix (equation 3.6) cubed.

RBF2

The general equation for a Gaussian function Ga is given by equation 3.7:

$$Ga = a \exp\left(-\frac{(r-b)^2}{2c^2}\right) \tag{3.7}$$

where a b and c are constants. The constant b determines the location of the Gaussian function, c is the standard deviation or how wide it is and a controls the height of the peak. The values of b and c are determined manually then the RBF fitting method is used to determine values of a.

The values of r employed in equation 3.4 can be given by equation 3.8

$$r_{i,j} = \exp\left(-C_i \times ||\mathbf{NRF}_i - \mathbf{R}_j||^2\right) \tag{3.8}$$

where C_i is dependent on the euclidean distance to the furthest point in the nearest 1% of points $P_{1\%}$ may be given by equation 3.9.

$$C_i = \frac{1}{\left(\frac{2P_{1\%}}{3}\right)^2} \tag{3.9}$$

We divide $2P_{1\%}$ by 3 as we want the position corresponding to $P_{1\%}$ to be overlapped by three times the standard deviation.

The value of $P_{1\%}$ for an atom was found by ordering its corresponding row in the distance matrix from lowest to highest. The structure that was at the index floor $\left(\frac{\#trainig}{100}\right)$ was selected and the euclidean distance to this structure calculated to give $P_{1\%}$.

3.3.3 Results and discussion

The RBF1 fits were completed successfully for both aspirin and toluene; however we note that there was a significant error in the fit for malonaldehyde for all interactions. This was evident as the error for predictions of the training set should be zero, this was not the case for malonaldehyde. As no coding error was found upon inspection, and the same code used for all runs, we attribute this to either issues with the numpy solve routine or problems within the dataset. Attempting to invert the matrix without the use of the numpy solve routine proved too memory intensive and we omit the analysis of malonaldehyde here.

We apply the same analysis as we did to the neural networks to the RBF fits. We begin by plotting the S curve for RBF1 for the aspirin test set and compare this to the PM6 data calculated earlier (Figure 3.20).

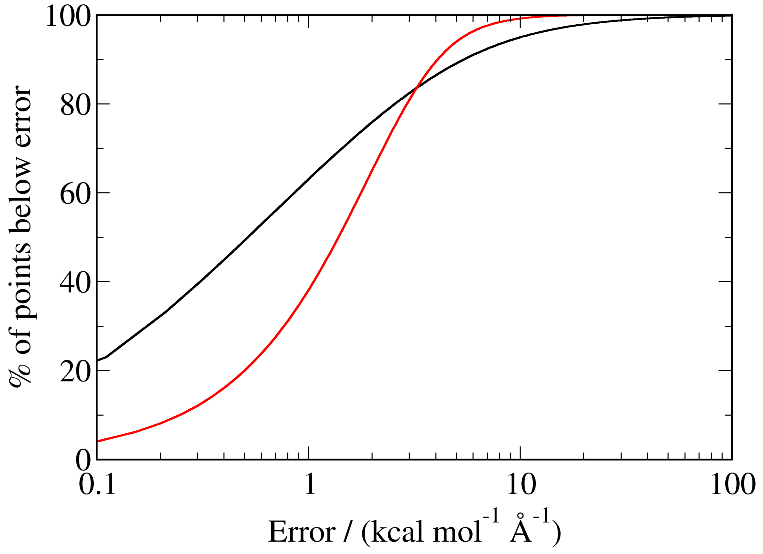


Figure 3.20: S curve for RBF1 (black) and PM6 (red)

We note that the gradient of this curve is significantly smaller than the gradients observed for the neural networks predicting the vanilla decomposed forces and PM6. This can be explained by the large range of accuracies for the predictions of different pairwise interactions, which can be seen from the L1 plot (Figure 3.21).

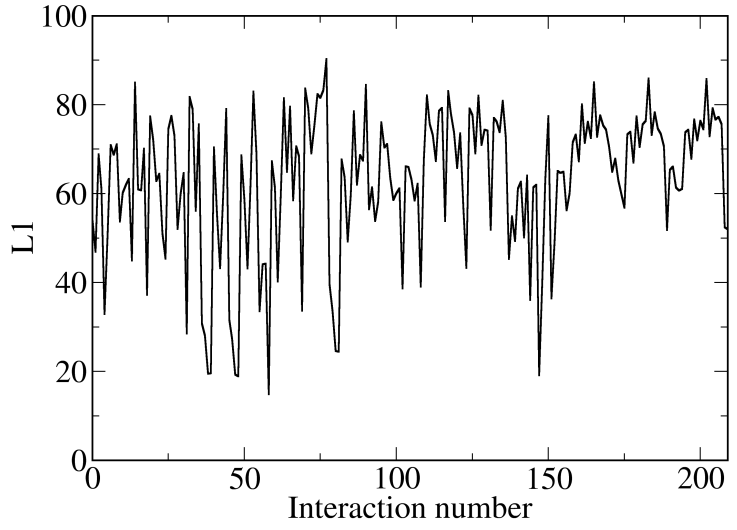


Figure 3.21: l1 plot for RBF1. Interactions 0 to 35 correspond to carbon-carbon interactions. 36 to 77 correspond to oxygen-oxygen and oxygen-carbon interactions and 78-209 correspond to all interactions to hydrogen atoms.

The range of the L1 values in the plot is significantly larger than the previously presented L1 plots ranging from 14.9% to 90.3% for different interactions. It is also noted that the same observations about the best fitted pairwise interactions obtained from the L1 plots for neural network fits to vanilla method decomposed forces, cannot be observed for the RBF fit, with methyl hydrogen interactions being among some of the best fitted interactions.

Plotting the pairwise interactions involving methyl hydrogens we see a repeating pattern as we did with PFN3b (Figure 3.22).

We note the same patterns repeated from PFN3b where for example in the case of the interaction with C7 the same hydrogens show better/worse performance relative to each other. We also note very similar L1 values for these interactions when looking at PFN3b and RBF1.

We similarly note a shallower gradient in the S curve for toluene compared with PFN3b and that it shows worse performance than PFN3b (Figure 3.23).

We attempted to explain which interactions would be fitted better by plotting the L1 value of an interaction against the effective range calculated from

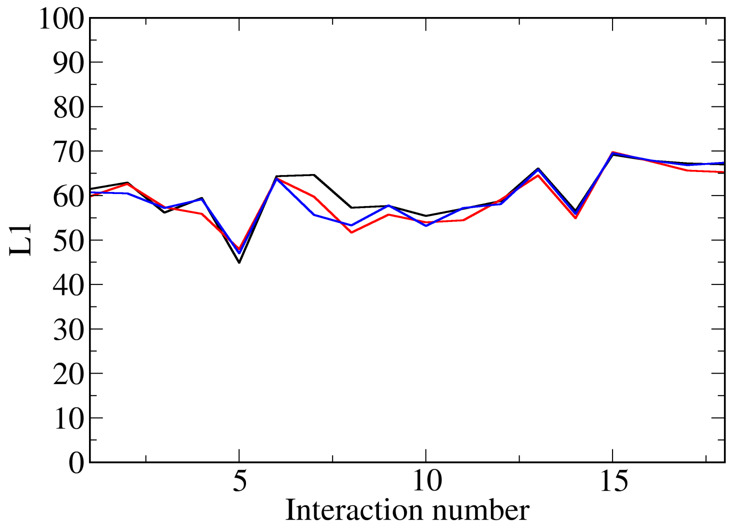


Figure 3.22: L1 values for H19 (black), H20 (red) and H21 (blue) interacting with the other 18 atoms in the aspirin test set as predicted by RBF1 (interactions between these hydrogens are excluded).

the variogram for that interaction, for both aspirin (Figure 3.24) and toluene (Figure 3.25) however, surprisingly we observe no correlation in either case.

RBF2 shows a worse result for fitting aspirin than RBF1 based on the s curve shown in figure 3.26 which shows a significantly lower L1 value. This could be due to the added difficulty in determining appropriate values of the exponents (something Fornberg et al.[48] show is very important) and good further work would be an exploration of this or perhaps a cubic function is better suited to fitting the relationship between distance and the value of the force.

We summarise the performance of RBF1 and RBF2 in table 3.6. This clearly shows RBF1 fitting aspirin significantly outperformed the toluene fit and the RBF2 fit on aspirin.

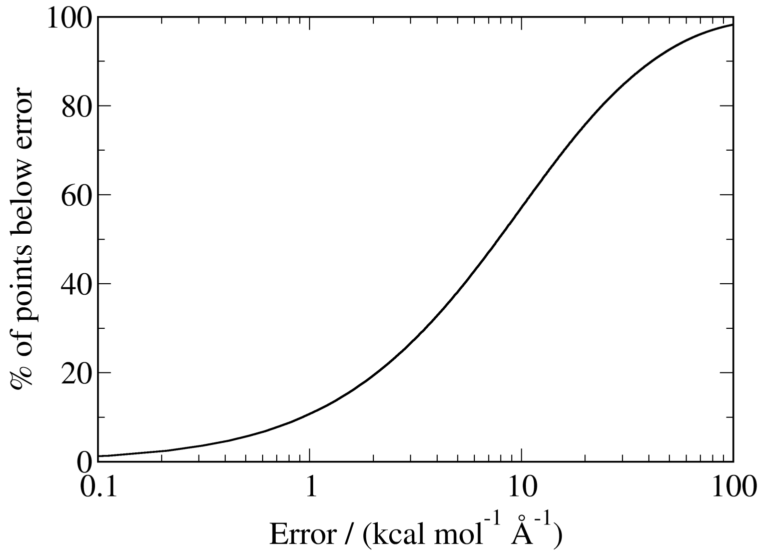


Figure 3.23: S curve for RBF1 making predictions on the toluene test set

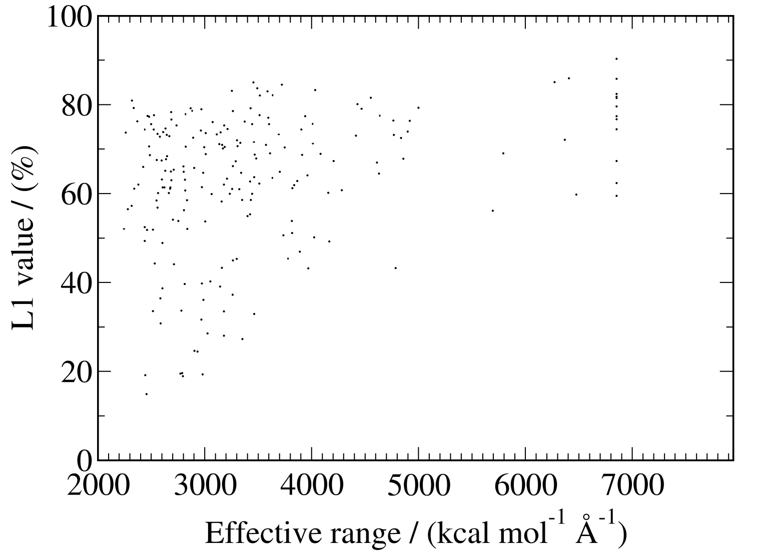


Figure 3.24: L1 values versus effective range for RBF1 making predictions on the aspirin test data

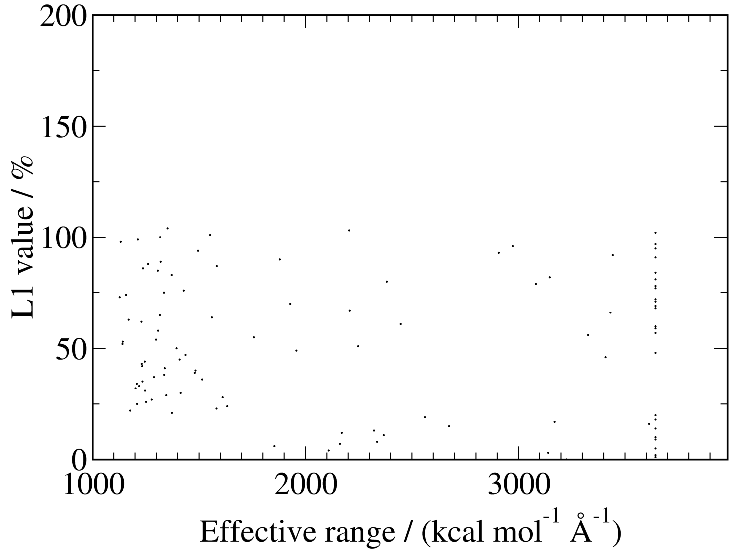


Figure 3.25: L1 values versus effective range for RBF1 making predictions on the toluene test data

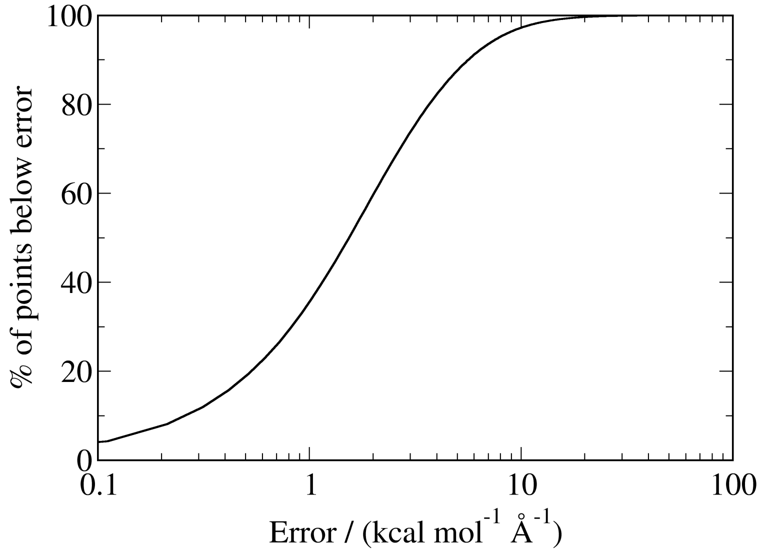


Figure 3.26: S curve for pairwise predictions for aspirin made by RBF2

Table 3.6: The performance of RBF1 fitting aspirin and toluene and the performance of RBF2 fitting aspirin with PFN3b for comparison

Method	Total MAE	Total RMSE	Magnitude MAE	Magnitude RMSE	Angle MAE	Angle RMSE
RBF1 (aspirin)	2.63	7.66	2.59	8.64	0.0386	0.0859
RBF1 (toluene)	24.1	34.7	46.3	58.4	0.513	0.578
RBF2 (aspirin)	9.05	14.2	14.2	21.0	0.0986	0.156

3.4 Conclusions and further work

In this section we attempted to fit the pairwise forces calculated in chapter 2 by using neural networks and radial basis function fits.

We developed a novel neural network architecture called PFN3b which combined the advantages of shortcut connections proposed by resnet with the advantages of constructing an architecture during training observed in network types such as cascade correlation or the neural evolution through augmenting topologies.

We demonstrated that this network was able to significantly outperform fixed network architectures of comparable size. By constructing the network block by block we were able to use a sigmoid activation function without encountering issues due to the vanishing gradient problem.

We note that although this network was outperformed by the network decomposed forces network, the pairwise forces it learned are more chemically sensible as the data fitted implicitly guarantees features such as structures corresponding to maxima or minima have a set of pairwise forces of 0.

We also applied PFN3b to fitting the set of biased decomposed forces and noted only slightly worse performance in the total MAE which was 1.82 kcal mol^{-1} Å^{-1} compared with 1.72 kcal mol^{-1} Å^{-1} for the vanilla decomposed forces. However we noted that the interactions to lighter atoms appear to have been fitted significantly better which is important for the performance of an MD simulation.

Further work for the network fits would involve investigating changes to the optimization algorithm, changing the activation functions, changing the number of neurons or connections of the standard block and perhaps altering the training scheme to resemble cascade-correlation where multiple candidate blocks are trained with the best being added to the network.

We then attempted to use radial basis function interpolation, which unlike neural networks, implicitly guarantees that every training point will be predicted exactly - which would have been a desirable quality because of the nature of our training data being deterministically calculated as opposed to based on experimental measurement with an associated error but because of the presence of noise in the training data, this is less desirable. However the RBF fits performed worse, showing significant issues when fitting the malonaldehyde datasets, likely due to issues with the data.

We partially attributed this poorer performance to the need to define an appropriate distance metric that allows for easy interpolation of the pairwise force and further work in this area could involve identifying a more appropriate distance metric.

In the case of RBF2 which used a Gaussian function further work needs to be done to identify appropriate exponents which may yield better performance.

Comparing the RBF fitting with neural networks, neural networks seem to be the better approach. With the use of modern deep architectures one can more easily obtain a good model, which although does not guarantee implicitly each training point is fitted exactly it does attain good performance on a test set and is not sensitive to noise in the same way RBF interpolation is.

We now have multiple machine learned force field models capable of accurately predicting the forces in our test sets, the next step is to evaluate the performance of these models in the context of a molecular dynamics simulation.

Chapter 4

Simulations using the fitted forces

4.1 Introduction

In chapter 3.2 we produced machine learning models that fit the forces of our aspirin, toluene and malonaldehyde datasets to a high degree of accuracy. However the real test for any machine learned force field is how well it performs in simulations compared with force fields currently employed.

We implemented our machine learned force field into the Amber software package[49] such that it could make force predictions at every time step.[i] However due to difficulties in interfacing the python libraries within the FORTRAN code of Amber this proved very slow.

Therefore a simple molecular dynamics simulation code was written in python as a way to test the performance. This simulated the molecule in vacuum in an NVE ensemble using the Verlet equation at each time step. The initial velocities were simply set to 0.

Every simulation run for our three test molecules (aspirin, toluene and malonaldehyde) “exploded” - an increase in kinetic energy was observed to the extent that distances between atoms became completely unrealistic and the molecule dissociated. This was observed for both the neural network and RBF fits. However the time taken for the explosion to be observed was different for each method.

[i]Achieved as part of a collaboration with Linghan Kong

Even when using AMBER to conduct an NVT molecular dynamics simulation using PFN3b for malonaldehyde, run using a thermostat to help remove excess energy by keeping the temperature constant, we still observe the explosion.

There could be multiple reasons for the simulation to explode. The first and most obvious is the lack of energy conservation in our force predictions which could mean over time we are gaining energy as the simulation progresses.

Another potential issue is one of extrapolation. The simulation may not necessarily be staying in regions of phase space appropriately represented by our training data. If this occurs then the force predictions may be completely nonsensical inevitably affecting the quality of our simulation. To investigate if extrapolation is an issue we looked at every structure in the simulation and assessed if any of the distances between the atoms lie outside the range of distances for those atoms in the training set, to obtain a subset of extrapolated points. Ideally one would calculate the convex hull of the data and assess if the structures lay within or outside it. This would give us a complete set of extrapolated points. However given the dimensionality of the data, this proved computationally prohibitive when attempting to use Delaunay triangulation in order to do so.

The final issue which could contribute to the systems exploding is "deterministic noise" in the force predictions. Deterministic noise is when a flexible function (such as a neural network) is fitted to a dataset, where potentially the function passes through each point exactly. However between points the function is very complicated such that the values can vary considerably. This is illustrated in figure 4.1.

This is of notable concern as it means the network is potentially very sensitive to small changes in the structures, which if true, would necessitate very small time steps to compensate. This would also be an argument for the use of cross validation when training the networks, even though ideally we should fit every point exactly.

We investigate the sensitivity of the neural network to changes in the structure by generating the intermediary points between two normal timesteps in a simulation, then calculating the forces for these additional points. We also do the same for a simulation run with the GAFF force field[7] to compare.

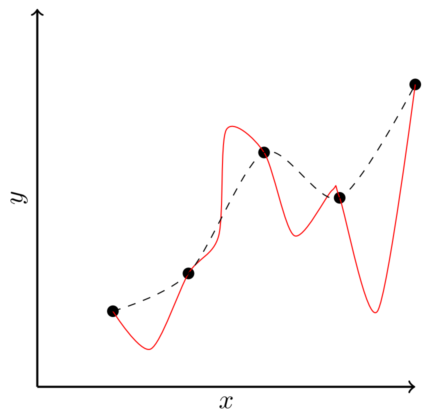


Figure 4.1: Illustration of deterministic noise. Note each of the black points is fitted by the red line (a hypothetical machine learned fit) exactly; however the function shows significant complexity between the points, which may differ considerably from the true function (dashed line).

4.2 Method

In order to investigate the observed trajectories of the simulation the temperature of the aspirin simulation with PFN3b and GAFF was calculated. This included the intermediary points (see section 4.2.4), which were used to calculate the kinetic energy, which was then used to calculate the temperature in Kelvin.

Additionally the temperatures for the trajectory obtained from the simulation using the network decomposed forces were calculated.

4.2.1 The simulation program

The algorithm for our simulation program is summarised in algorithm 4.1. We ran simulations using this code to test the performance of the various fits in the context of an MD simulation.

As the models predicted pairwise forces, these were simply rescaled when necessary for the models which made predictions between zero and one; and combined using equation 2.5 to give Cartesian forces.

For aspirin we ran simulations using PFN3b and the network decomposed forces and analysed them. For the other molecules these were simulated using their PFN3b fits using Amber.

Algorithm 4.1 Molecular dynamics simulation

```
    load $startstructure$
    load $predictionmodel$
    $timestep \leftarrow 0.5fs$
    Declare $\#timesteps$
5:  Declare $trajectory$ shape $\leftarrow [\#steps][\#atoms][3]$
    $trajectory[0] \leftarrow statstructure$
    loop $i \leftarrow 0 : \#timesteps$
        $forces \leftarrow predictionmodel$ get Cartesian forces for trajectory[i]
        if  $i == 0$ then
10:         VerletStep $(r(t - \Delta t) = trajectory[i], r(t) = trajectory[i], forces)$
        else
            VerletStep $(r(t - \Delta t) = trajectory[i-1], r(t) = trajectory[i], forces)$
        end if
        Save $trajectory$
15: end loop
```

4.2.2 Amber simulations

For NVT simulations or simulations where the GAFF forces were used with the default parameters for the version of AMBER employed and the simulations of malonaldehyde and toluene, AMBER was used to run them.

The NVT simulations were run using the Langevin thermostat with no pressure limit, a time step of 0.5 fs and a reference temperature of 300K. Different runs were conducted with a collision frequency of 5, 10, 15 or 20 and using these parameters we were able to obtain a simulation of length 23.3 ps for aspirin without issues of the system 'exploding' being present. Using a lower collision frequency of 2.0 only allowed for a simulation of length 2.5 ps before the system 'exploded'.

An NVT simulation was also run with a collision frequency of 20 but with a timestep of 1.0 fs and were able to obtain a 55ps simulation without the system exploding.

Due to difficulties in integrating the python code into AMBER's FORTRAN code, an alternative approach was required.

For every frame, the AMBER code was modified to write the atomic coordinates to a text file. AMBER then called a python program which read this text file then made predictions of the forces, saving these to a text file. AMBER then read this text file, and used the forces to predict the next frame. This was very slow but was a pragmatic initial way of integrating the two programs.

4.2.3 Exploring extrapolation

In investigating extrapolation in the aspirin dataset we first plotted the two dihedral angles ($\tau_{7,6,8,11}$) and ($\tau_{6,7,13,9}$). These indicate how the two functional groups attached to the benzene ring in aspirin move out of the plane of the benzene ring relative to each other and is indicative of the conformers observed in the dataset for every structure in the aspirin training data. The dihedral angles were calculated for the PFN3b trajectory in order to yield a visual indication of how dense the training sampling was along the followed trajectory.

We then constructed a mechanism to find a subset of the extrapolated points. We define "extrapolation" as points which lie outside the convex hull defined by the training data. Attempts were made to calculate the convex hull and assess if points lay within it by the use of Delaunay triangulation; however this was found to be computationally prohibitively expensive.

Instead we simply identified the points that had at least one nuclear repulsion force that lay outside the range of the values for that nuclear repulsion force in the training data.

4.2.4 Investigating deterministic noise

A step in a simulation is determined using the Verlet method (equation 4.1)

$$\mathbf{r}(t + \Delta t) = 2\mathbf{r}(t) - \mathbf{r}(t - \Delta t) + \frac{\mathbf{f}(t)}{m}\Delta t^2 \tag{4.1}$$

where Δt is the timestep $\mathbf{r}(t + \Delta t)$ are the positions after a timestep $\mathbf{r}(t - \Delta t)$ are the positions in the step before the current structure, $\mathbf{r}(t)$ are the current positions and $\mathbf{f}(t)$ are the forces for the current positions.

This equation is derived from the sum of equations 4.2 and 4.3.

$$\mathbf{r}(t + \Delta t) = \mathbf{r}(t) + \mathbf{v}(t)\Delta t + \frac{\mathbf{f}(t)}{2m}\Delta t^2 \tag{4.2}$$

$$\mathbf{r}(t - \Delta t) = \mathbf{r}(t) - \mathbf{v}(t)\Delta t + \frac{\mathbf{f}(t)}{2m}\Delta t^2 \tag{4.3}$$

In order to generate the intermediary points along a trajectory for a given point $\mathbf{r}(t)$ we need to make a step using a time step smaller than Δt, which we will call δt. We need to calculate values of $\mathbf{r}(t + \delta t)$. Simply substituting δt for Δt in equation 4.1 will not work as the values of $\mathbf{r}(t - \delta t)$ are not known.

First we rewrite equation 4.2 in terms of δt to give equation 4.4

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \mathbf{v}(t)\delta t + \frac{\mathbf{f}(t)}{2m}\delta t^2 \tag{4.4}$$

Rearranging equation 4.3 to make $\mathbf{v}(t)$ the subject for the time step Δt and then substituting into 4.4 gives us equation 4.5

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + [\mathbf{r}(t) - \mathbf{r}(t-\Delta t)]\,\frac{\delta t}{\Delta t} + \frac{\mathbf{f}(t)}{2m}(\delta t^2 + \Delta t \delta t) \tag{4.5}$$

This then tidies up to giving equation 4.6

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \left[\frac{\mathbf{r}(t) - \mathbf{r}(t-\Delta t)}{\Delta t} + \frac{\mathbf{f}(t)}{2m}\Delta t\right]\delta t + \frac{\mathbf{f}(t)}{2m}\delta t^2 \tag{4.6}$$

We use equation 4.6 to generate nine more points between the regular timesteps (Δt) in the simulation using PFN3b.

We generate nine intermediary structures between each pair of structures in the trajectory for PFN3b and GAFF simulating aspirin. For PFN3b we compute the forces for these additional points and recombine them to give the Cartesian forces.

In order to judge the change in force between points we calculate the average difference between the first point in a pair of structures and each of the intermediary points. This is done for both GAFF and the PFN3b simulation.

We then calculate the average rate of change of force between the smaller timesteps δt and the previous full timesteps Δt given by equation 4.7, where i is the structure in the trajectory with the nine intermediary points, N is the number of atoms and R_0 and F_0 are the positions and forces of the previous full timestep.

$$\frac{\delta F}{\delta R} = \frac{1}{3N}\sum \frac{\mathbf{F}_i - \mathbf{F}_0}{||\mathbf{R}_i - \mathbf{R}_0||} \tag{4.7}$$

4.3 Results and Discussion

The gain in kinetic energy as the simulation runs is evident from figure 4.2, which compares the temperature change of the simulation run with PFN3b and GAFF for NVE simulations. Whereas the GAFF temperature shows little overall change, the network run begins to gain temperature at an appreciable rate and eventually the temperature is so high the molecule falls apart.

The other fits show a similar pattern when used in a simulation, a period of

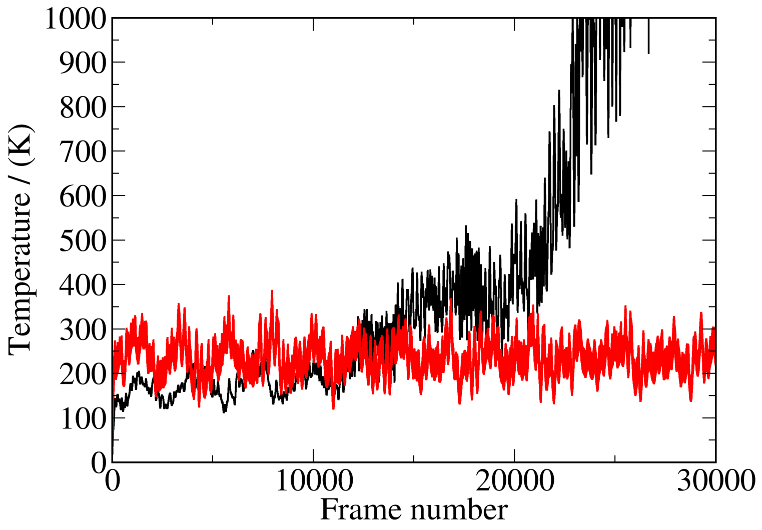


Figure 4.2: Change in temperature over the course of the simulation for aspirin simulated using PFN3b forces (black) and GAFF forces (red)

lower temperature where the simulation appears stable followed by an increase in temperature. Looking at the time taken to reach 1000K, the network decomposed forces show the best performance with 11.4 ns before this temperature is reached and PFN3b ran for 2.5 ns.

4.3.1 Investigating extrapolation

The plot of ($\tau_{6,7,13,9}$) vs ($\tau_{7,6,8,11}$) (Figure 4.3) for the aspirin training data shows multiple well sampled conformers. However the transitions between them are rather poorly sampled shown by the lack of data points, as expected as these data were generated using an MD simulation which will sample lower energy configurations more than higher energy configurations.

We superimpose the trajectory followed for PFN3b on figure 4.3 to give Figure 4.4. The start of the trajectory is within a well represented area in the cluster at approximately (0,-2). However as the trajectory progresses and a transition occurs from the cluster at (0,-2) to the one at (0,2) the path taken is only very sparsely represented in the dataset.

Working out a subset of the points that are being extrapolated (Figure 4.5)

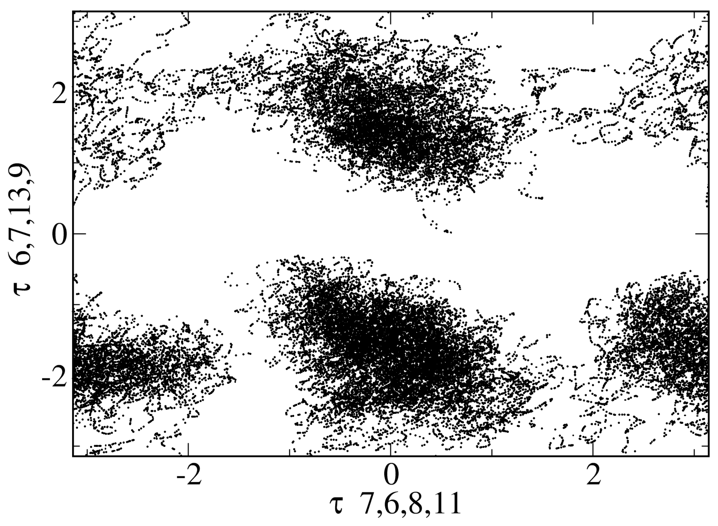


Figure 4.3: The aspirin training data plotted as dihedral ($\tau_{6,7,13,9}$) vs dihedral ($\tau_{7,6,8,11}$) in radians.

shows that even early in the simulation, certain frames are being extrapolated rather than interpolated and by the end of the simulation, before the "explosion" almost every point is extrapolated.

Although extrapolation likely plays a role in the simulation exploding, it is not the only factor. When a simulation was run in AMBER where every detected extrapolated point had its forces predicted using GAFF rather than PFN3b, the simulation still exploded having gained so much energy that even with GAFF conducting 100% of the predictions by the end, it was still not enough.

4.3.2 Investigating deterministic noise

It would appear from a comparison in figure 4.6 between the average change in forces between each timestep for GAFF and for network PFN3b that both are very similar for the majority of the simulation, only differing significantly towards the end of the simulation. This would indicate that deterministic noise is perhaps not a reason for energy gain in the NVE simulation until later on and this indicates for the majority of the simulation the network is not too sensitive

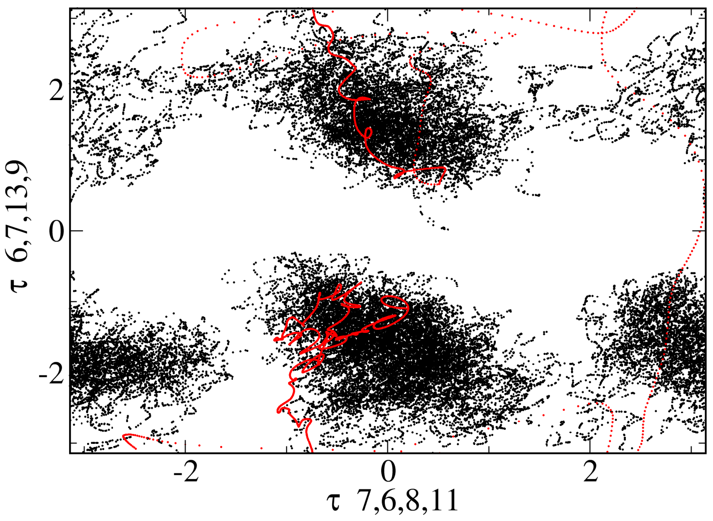


Figure 4.4: The aspirin training data (black) and for the PFN3b simulation trajectory (red) plotted as dihedral ($\tau_{6,7,13,9}$) vs dihedral ($\tau_{7,6,8,11}$) in radians.

to change and the time step of 0.5fs is appropriate.

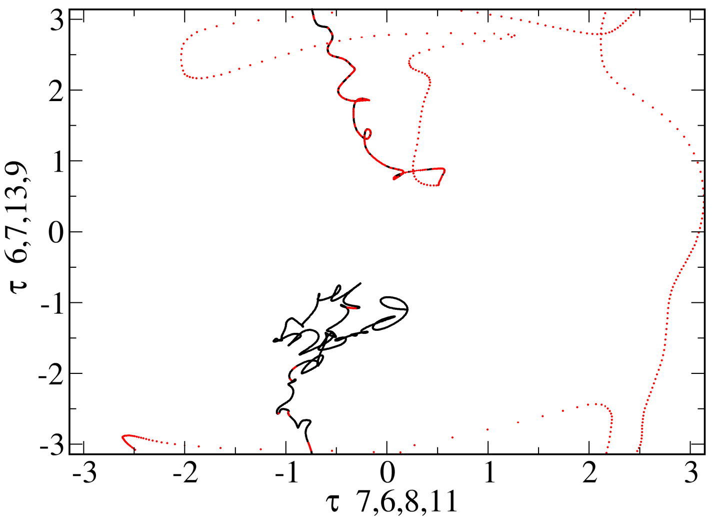


Figure 4.5: The aspirin simulation trajectory with PFN3b forces. The subset of extrapolated points are shown in red plotted as Dihedral ($\tau_{6,7,13,9}$) vs dihedral ($\tau_{7,6,8,11}$) in radians. Note even early in the simulation around the range of x=(0,-1) y=(-0.5,-2) we observe multiple instances of extrapolation.

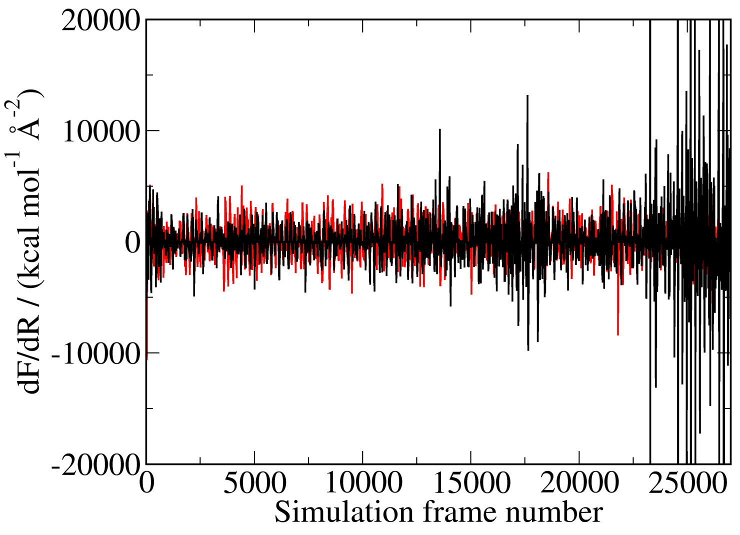


Figure 4.6: The rate of change of force ($\frac{\delta F}{\delta r}$) for intermediary steps compared to the previous full timestep for the PFN3b (black) and GAFF (red), NVE simulations of aspirin.

4.3.3 Investigating energy conservation

The final issue which may be causing energy gain is the problem of energy conservation. Essentially all of the atomic forces should be the first derivative of the energy. If they are, then energy change between two points is always equal regardless of the path taken and any arbitrary path that starts and finishes in the same place it started necessarily has an energy change of zero.

If the forces are not the first derivative of the energy, none of the previous necessarily holds true. Unfortunately it may be shown analytically that our forces are not energy conserving. This could help explain the observed energy gain.

The proof is as follows:

2 atom system, single neuron

Forces are energy conserving when they can be expressed as the negative of the first derivative of the energy (equation 4.8)

$$\mathbf{F} = -\nabla E \tag{4.8}$$

As $\nabla \times \nabla$ is always 0, to prove (4.8) we can show that the curl of the forces equals zero (equation 4.9):

$$\nabla \times \mathbf{F} = \mathbf{0} \tag{4.9}$$

Expanded, we must show that equation 4.10 holds true for the Cartesian force on an atom.

$$\begin{bmatrix} \frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z} \\ \frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x} \\ \frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \end{bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix} \tag{4.10}$$

To prove this is true for all values of the pairwise force let us consider trying to prove $\frac{\partial F_z}{\partial y} = \frac{\partial F_y}{\partial z}$.

As the equation for the Cartesian force can be given by equation (2.5), considering a single pairwise interaction between two atoms A and B we get the two equations (4.11) and (4.12)

$$\frac{\partial F_z}{\partial y} = \frac{\partial f}{\partial y} \times \frac{(z_A - z_B)}{r_{AB}} + f \times \frac{(y_A - y_B)(z_A - z_B)}{((x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2)^{\frac{3}{2}}} \tag{4.11}$$

$$\frac{\partial F_y}{\partial z} = \frac{\partial f}{\partial z} \times \frac{(y_A - y_B)}{r_{AB}} + f \times \frac{(y_A - y_B)(z_A - z_B)}{((x_A - x_B)^2 + (y_A - y_B)^2 + (z_A - z_B)^2)^{\frac{3}{2}}} \tag{4.12}$$

Simplifying, equations (4.11) and (4.12) will be equivalent when (4.13) is true.

$$\frac{\partial f}{\partial y} \times (z_A - z_B) = \frac{\partial f}{\partial z} \times (y_A - y_B) \tag{4.13}$$

Now consider our neural network with a single neuron. This can be given by equation (4.14) where w is the network weight.

$$f = \mathrm{f}(w r_{AB}) \tag{4.14}$$

Using the chain rule, which states the derivative of a composite function is the derivative of the outer function evaluated at the inner function times the derivative of the inner function, we can show

$$\frac{\partial f}{\partial y} = \frac{w(y_A - y_B)}{r_{AB}} \mathrm{f}'(r_{AB})$$

and

$$\frac{\partial f}{\partial z} = \frac{w(z_A - z_B)}{r_{AB}} \mathrm{f}'(r_{AB})$$

Substituting this back into equation 4.13 we get equation 4.15 which proves the force is conservative for any prediction of the value f.

$$\frac{w(y_A - y_B)}{r_{AB}} \mathrm{f}'(r_{AB})(z_A - z_B) = \frac{w(z_A - z_B)}{r_{AB}} \mathrm{f}'(r_{AB})(y_A - y_B) \tag{4.15}$$

4.3.4 3 atoms, 3 neurons

For a 3 atom system consisting of atoms A B and C, the total Cartesian force is given as the *sum* of projected pairwise interactions. Therefore equation 4.13 may be given for atom A by equation 4.16

$$\frac{\partial f_{AB}}{\partial y} \times (z_A - z_B) + \frac{\partial f_{AC}}{\partial y} \times (z_A - z_C) = \frac{\partial f_{AB}}{\partial z} \times (y_A - y_B) + \frac{\partial f_{AC}}{\partial z} \times (y_A - y_C) \tag{4.16}$$

For our predictions we predict f_{AB} and f_{AC} independently; therefore this may be treated as two separate functions which we will call f and g.

A neuron in a neural network is a function of the weighted sum of its inputs. Our inputs are a function of r but for simplicity let us assume they are r. We can express the derivative of the single neuron network as equations 4.17 and 4.18 for $\frac{\partial f_{AB}}{\partial z}$ and $\frac{f_{AC}}{\partial y}$ respectively. Similar equations can be constructed for $\frac{\partial f_{AB}}{\partial y}$ and $\frac{f_{AC}}{\partial z}$.

$$\frac{\partial f_{AB}}{\partial z} = \left(\frac{w_1(z_A - z_B)}{r_{AB}} + \frac{w_2(z_A - z_C)}{r_{AC}}\right) \mathrm{f}' \left(\sum(w_i r_i)\right) \tag{4.17}$$

$$\frac{\partial f_{AC}}{\partial y} = \left(\frac{k_1(z_A - z_B)}{r_{AB}} + \frac{k_2(z_A - z_C)}{r_{AC}}\right) \mathrm{g}' \left(\sum(k_i r_i)\right) \tag{4.18}$$

Substituting this back into equation 4.16 we must prove

$$\left(\frac{w_1(y_A - y_B)}{r_{AB}} + \frac{w_2(y_A - y_C)}{r_{AC}}\right) \mathrm{f}' \left(\sum(w_i r_i)\right) (z_A - z_B) +$$

$$\left(\frac{k_1(y_A - y_B)}{r_{AB}} + \frac{k_2(y_A - y_C)}{r_{AC}}\right) \mathrm{g}' \left(\sum(k_i r_i)\right) (z_A - z_C)$$

equals

$$\left(\frac{w_1(z_A - z_B)}{r_{AB}} + \frac{w_2(z_A - z_C)}{r_{AC}}\right) \mathrm{f}' \left(\sum(w_i r_i)\right) (y_A - y_B) +$$

$$\left(\frac{k_1(z_A - z_B) + k_2(z_A - z_C)}{r_{AB}}\right) \mathrm{g}' \left(\sum(k_i r_i)\right) (y_A - y_C)$$

which is true only when

$$\mathrm{f}' \left(\sum(w_i r_i)\right) = \mathrm{g}' \left(\sum(k_i r_i)\right)$$

and therefore the weights k equal the weights w. Therefore our forces are not conservative as we do not guarantee the above for any network predicting forces for systems with more than two atoms.

As a consequence of this lack of conservation, energy may be continually gained as the simulation progresses as it does not necessarily follow that a path consisting of a closed loop will result in an energy change of zero.

One solution would be to fit a simple pairwise potential which would guarantee energy conservation; however this is unlikely to represent an improvement on existing forcefield or machine learned methods. Alternatively one could fit energies rather than forces and then either analytically find the first derivative of the model or do so numerically, which other groups have done.

4.4 Conclusions and further work

We have observed that simulations using our fitted forces observe a gain in kinetic energy to the point where the simulation becomes unstable. We attributed this to 3 potential factors, deterministic noise, extrapolation and the lack of energy conservation in our predictions.

We found that deterministic noise, though present significantly towards the end of the simulation where it likely serves to further increase the energy gain, is not a factor early on in the simulation.

Interestingly we observe that extrapolation is present even quite early in the simulation and likely serves to cause energy gain because of poor predictions of force. However it is likely not the only factor as code written to use GAFF when a subset of extrapolated points was detected did not serve to improve the issue significantly and we still saw kinetic energy gain.

Finally we show mathematically that our forces are not energy conserving. This means the forces cannot be written as the first derivative of a potential energy and therefore the energy change between two points may be entirely dependent on the path taken, or that energy gain/loss may be observed when a path consisting of a loop is taken.

None of these observations necessarily explains why we observe an overall gain in energy on average as they could all equally be responsible for a loss in energy and no reasonable explanation has been found for this.

Further work in this area to combat the issue of energy conservation may be to implement a network that implicitly guarantees that the predicted pairwise forces are energy conserving. This would mean they are no longer predicted independently of one another, this work has shown fairly conclusively that predicting them independently without any other 'fix' yields a poor simulation in an NVE simulation.

Another solution may lie in the post processing of forces, potentially looking at predictions for similar structures and making adjustments such that the forces are energy conserving; however this will yield a poorer result than the previous solution as the corrections will never be perfect.

Finally identifying and implementing a method that allows for the detection of all extrapolated points (those that lie outside the convex hull defined by the data points) quickly and efficiently will potentially help in the implementation of any checks on forces and prevent poor predictions from being used in the simulation. Chemical space is very large and high dimensional and the chances of covering it appropriately in a training set, such that no points are extrapolated during the course of a simulation, are slim.

Chapter 5

Charge decomposition

5.1 Introduction

In chapter 2 we introduced a method for decomposing total Cartesian forces into pairwise internal forces which were fitted by machine learning models to create a forcefield.

This pairwise decomposition method also has a wide range of other applications to situations where predictions must be made and some total value kept constant. One example of this are the partial atomic charges of molecules in a system. In the absence of any external influence, the total charge of a system is constant during an MD simulation. However it is possible for the charge on each atom to change.

The ability to predict changing charges may be useful for simulations where there are multiple components interacting, where decomposing pairwise forces for all these different components is not feasible. The charge on an atom will be influenced by the structure and conformation of the molecule,as well as by the other molecules in the system being studied. Currently it is common to use none polarizable force fields such as GAFF where the charges are fixed, when simulating systems in solution. We hope to improve on this by predicting charges for molecules in a system which are dependent on the conformation of that molecule. This will not capture the effects of multiple molecules influencing each other; however it represents a step forward from fixed charges.

In other work charges are updated by using quantum methods; however just like with forces, these incur a large computational overhead which could be reduced using machine learned fits. It has been shown that updating charges can have a significant impact on the observations made during a molecular dynamics simulation, for example on protein folding[50] but also on much simpler systems such as erythritol in water[51].

Existing methods for charge prediction using machine learning such as the work conducted by Mills and Popelier[52], the work conducted by Kato et al.[53] or the work conducted by Wang and Gao[54] do not implicitly guarantee the charge will sum to zero in their predictions, something that is a noteworthy advantage of our approach.

5.2 Vanilla charge decomposition scheme

For every atom in a system we have a single value of charge associated with it, unlike forces which are a vector quantity with 3 values. Consequently the vanilla scheme for charges is dependent only on the distances between atoms, and not displacements. Consider the 3 atom system presented in figure 5.1

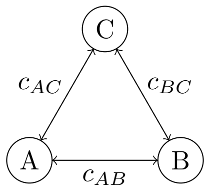


Figure 5.1: 3 atoms A B and C with their pairwise charges c_{AB},c_{BC} and c_{AC}

In the vanilla charge decomposition scheme the charge on each atom (q_A, q_B and q_C) can be given by the sum of fictious pairwise charges (C_{AB}, C_{BC} and C_{AC}), given by the following equations for our three atom system:

$$q_A = \frac{C_{AB}}{r_{AB}} + \frac{C_{AC}}{r_{AC}}$$
$$q_B = -\frac{C_{AB}}{r_{AB}} + \frac{C_{BC}}{r_{BC}}$$
$$q_C = -\frac{C_{AC}}{r_{AC}} - \frac{C_{BC}}{r_{BC}}$$

Note that the value of **e** (equation 2.4) used in the force decomposition scheme, in the charge decomposition becomes

$$e_{AB} = \frac{1}{r_{AB}}$$

for atoms A and B.

The inverse of the distance is used as there is no direction to the charge unlike the forces. Alternatively one could employ values of **e** which are simply zeros and ones; however it was not believed that was this a sensible approach.

As there is only 1 value per atom rather than 3 when we decompose forces our matrix for **e** has dimensions of N by $\binom{N}{2}$ rather than 3N by $\binom{N}{2}$.

The difference in matrix size and the different method for calculating **e** represent the only differences between this scheme and the vanilla method for force decomposition and the two methods are solved in the same way.

Care should be taken when attempting to ascribe physical meaningfulness to the pairwise charges as with the pairwise forces as the solution is selected from an infinite set of possible solutions somewhat arbitrarily. It could be considered that the pairwise charge is simply a measure of how much one atom polarizes another.

We note that charge predictions made for pairwise charges are implicitly charge conserving, always summing to zero whilst allowing for the charges on individual atoms to vary.

Although the charges implicitly summing to zero presents a limitation of the method in applications to systems with an overall charge, this issue is potentially surmountable by inclusion of charged 'counter ions' in the calculation.

Another limitation is the assumption that the charges are a function of the conformation of the solute alone as only the solute can be reasonably included in the decomposition scheme hence fitted model, as the number of pairwise interactions scales according to $\binom{N}{2}$. Whereas this is a poor assumption, this technique does still represent a step forward from assuming constant charge.

5.3 Application to the aspirin dataset

5.3.1 Introduction

We apply the vanilla charge decomposition scheme to the aspirin dataset as a proof of principle. The charges were generated using antechamber[49] for every training structure and every test structure. This was chosen as the method for charge calculation as the GAFF forcefield is parametrized to use these charges and an analysis of how these charges change is indicative of the usefulness of this method for aspirin.

5.3.2 Method

The structures for the aspirin training and test set were converted into xyz format using a python script, and then from xyz to mol2 format using openbabel.

Another python script was created to run antechamber on small batches of structures, as attempting to run it for all structures in one job caused it to exceed the maximum allowed time on the available computing resource.

The AM1-BCC model was used for calculating the charges.[55]

5.3.3 Results and discussion

We begin by looking at the range of charges in the aspirin training data noting that the different atoms show a range of charges which is summarised in table 5.1.

The pairwise charges form bands not too dissimilar to the pairwise forces which can be seen in figure 5.2. There appears to be a line of points across the plot with a charge of zero, inspection of these points reveals that the charges on every atom are zero meaning some structures returned an error when the charges were calculated. It was determined that a total of 24 structures out of 42353 had this issue.

We then calculate the pairwise force that results from the total charges using equation 2.7 in order to see the range of forces that is obtained (Figure 5.3).

We note from figure 5.3 that even for more distant interactions between lighter atoms (smaller values of the nuclear repulsion force) we see a significant range in the forces obtained within a band. This further illustrates the importance of accounting for changing charges during the course of an MD simulation.

Table 5.1: Charges on different atoms in aspirin for the training data

Atom ID	Average charge (e)	standard deviation
C1	-0.0594	0.00242
C2	-0.152	0.00552
C3	-0.155	0.00394
C4	-0.0808	0.00228
C5	-0.149	0.00378
C6	-0.144	0.00431
C7	0.175	0.00612
C8	0.663	0.016
C9	0.656	0.0172
O10	-0.603	0.0163
O11	-0.557	0.0155
O12	-0.52	0.0125
O13	-0.369	0.0133
H14	0.445	0.0109
H15	0.159	0.00397
H16	0.159	0.00544
H17	0.143	0.00343
H18	0.142	0.00341
H19	0.0826	0.00217
H20	0.0826	0.00217
H21	0.0826	0.00217

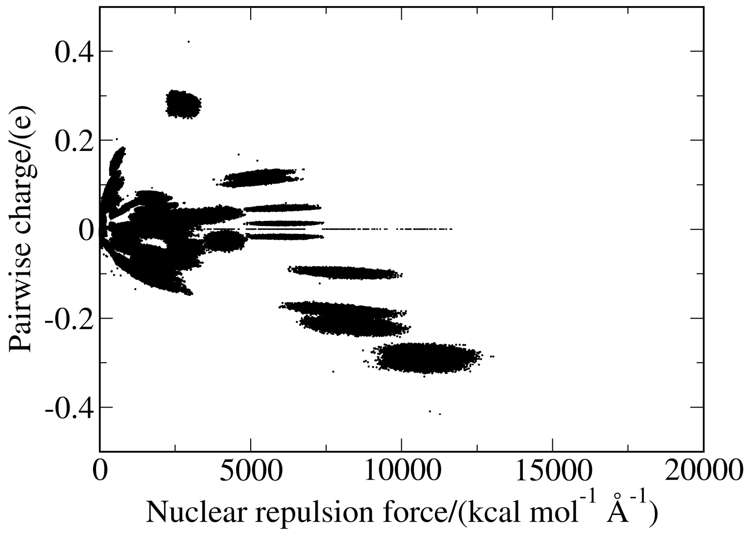


Figure 5.2: Decomposed charges versus the nuclear repulsion force for the aspirin training data.

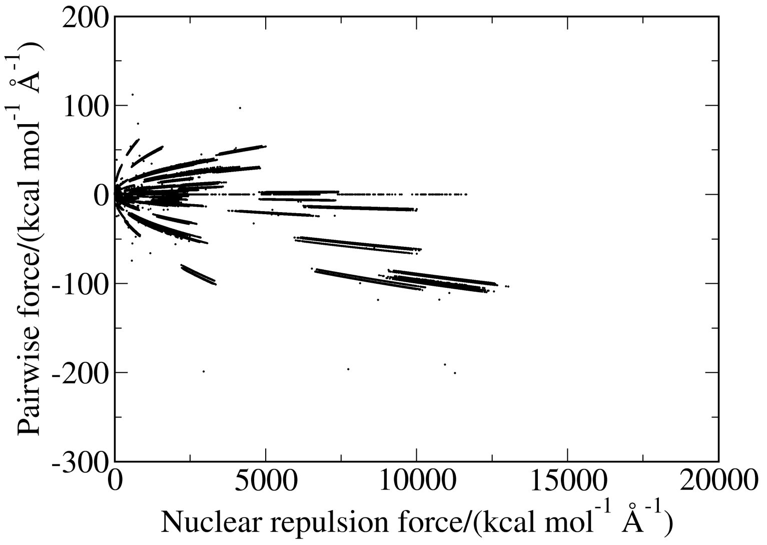


Figure 5.3: The range of forces, calculated as pairwise charge charge interactions observed for the aspirin training data

5.4 Network decomposed charges

5.4.1 Introduction

As for forces in section 2.6 we construct a neural network that internally decomposes the charges, making predictions of pairwise charges, recombining them and then evaluating the error of the total charge on each atom before backpropagating this error.

Choosing to decompose the charges in this manner conveys the same benefits as it did with the forces. The network can decompose charges as a function of all the structures and not just a single structure at a time, independently and secondly the network is not restricted to picking the solution with the smallest norm as we are when we use the Moore-Penrose pseudo inverse.

Whereas it is possible to predict the total charge for each atom and indeed others have done this[53 56] an advantage of predicting charges as fictitious pairwise interactions is the implicit guarantee that the charges will always sum to zero. Such an implicit guarantee is not offered by the cited methods which may cause issues in the context of a molecular dynamics simulation as charge is not necessarily conserved.

5.4.2 Method

A neural network was constructed which consisted of a single block consisting of 1000 neurons with a relu activation function followed by $\binom{N}{2}$ with a linear function. A sigmoid function was initially employed. However it was found that this did not yield satisfactory results possibly due to discontinuities in the calculated charges.

The dot product of the output layer of $\binom{N}{2}$ neurons from this block and a 'projection layer' which was a matrix of dimensions N by $\binom{N}{2}$ was calculated to give N charge predictions. The error was evalulated for these and back propagated to adjust the weights of the $\binom{N}{2}$ network predictions.

The projection layer was calculated similarly to equation 2.5 with the values of each pairwise interaction being given by 5.1.

$$e_{AB} = \frac{1}{||\mathbf{R}_A - \mathbf{R}_B||} \tag{5.1}$$

Similarly to the network decomposed forces we inspect the inner layer of the network containing $\binom{N}{2}$ neurons with the linear activation function in order to see the network predicted pairwise charges. This was done using by creating a model that used the same inputs but its output was determined by the layer of interest which was obtained using the 'get_layer' method in keras.

5.4.3 Results and discussion

When plotting the network decomposed pairwise charges we see a similar result, a series of band structures (Figure 5.4).

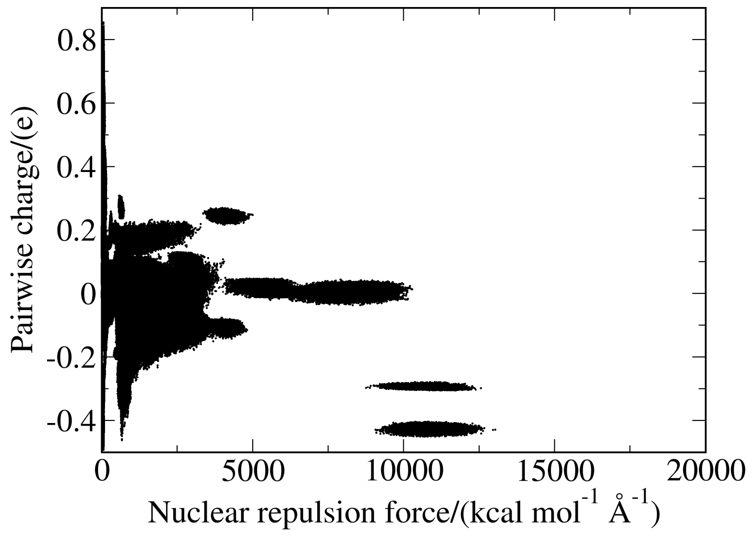


Figure 5.4: Decomposed charges versus the nuclear repulsion force for the aspirin training data.

Unlike the vanilla charge decomposition scheme, the network decomposed charges do not show the band at zero charge. This is likely because the network did not fit these points exactly and instead made more sensible predictions for them.

The error for predictions of the test charges can be seen in the S curve shown in figure 5.5. The S curve is calculated for predictions of the force based on charges predicted by the network.

The performance of the network appears poor for predictions of forces on the test set despite good performance on the training data which may indicate

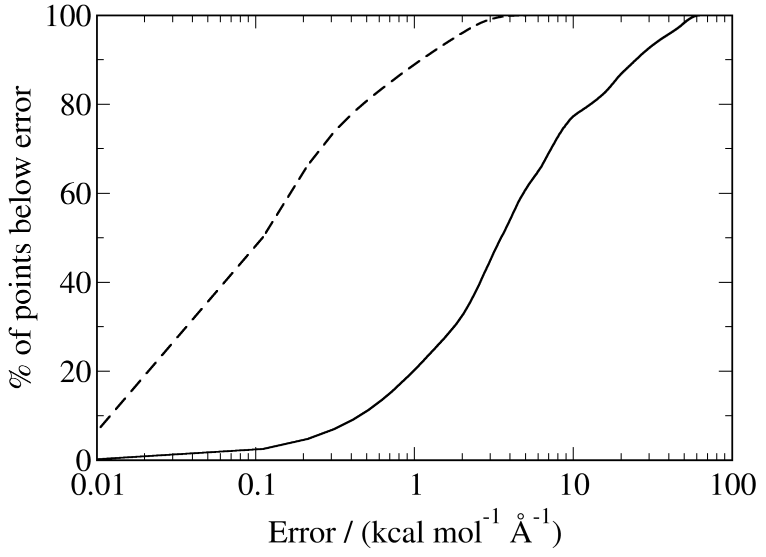


Figure 5.5: S curve for pairwise forces based on the predictions of charge by the network for the aspirin training data (dashed line) and the test data (solid line).

that the relationship between the inputs and outputs is complicated and therefore additional training data is required; however this would require further investigation.

5.5 Conclusions

In this short chapter we showed that the pairwise decomposition scheme developed for forces can easily be extended to other properties and we chose charges to demonstrate this. These pairwise charges could be fitted by a machine learning method and doing so would implicitly guarantee the conservation of charge.

Fitting charges based on the conformation of the solute may be useful when considering interactions with solvent and demonstrates a step forward from fixed charges.

We also show that like forces the machine learning method may also learn pairwise charges internally but recombine them to give total charges on each atom, implicitly guaranteeing that the charge remains constant in the system.

Although the charge will always sum to zero, in the case of a system with an overall charge, counter ions are oft employed, including these in the decomposition scheme will allow for the solute to have an overall charge.

Chapter 6

Conclusions

This project involved three key aspects, the development of an appropriate output for the prediction of Cartesian forces, the development of a machine learning method capable of fitting these forces and simulation using the predicted forces.

The pairwise force decomposition scheme developed to allow predictions of forces that implicitly obey Newton's 3rd law, allows for a simple and effective method for the acquisition of pairwise forces from Cartesian forces. The method requires very little computational overhead in comparison to schemes such as interacting quantum atoms (IQA) which can be used to find per atom energies and even with the simple vanilla scheme, chemically intuitive properties of these forces are obtained. Additionally we showed that by biasing this approach one can easily change which interactions have larger or smaller forces and developing new schemes for biasing the scheme, represents an interesting line of enquiry.

With an appropriate selection of a biasing scheme the forces may be tailored to be more chemically meaningful or to have the interactions to more distant atoms smaller or to cut connections completely with a bias of 0.

Though it is possible to decompose the forces internally as part of a machine learning routine and obtain a high level of accuracy, we note that this does not guarantee to yield chemically intuitive forces and we found that features such as minima having a force of zero are not necessarily guaranteed in the same way as the vanilla scheme.

We then fitted these decomposed pairwise forces, developing a novel neural network architecture called PFN3b to do so. PFN3b combined the advantages of building a network during the training process and freezing connections with shortcut connections and its superior performance compared with fixed architecture networks of a comparable size shows the advantages of this.

PFN3b could be further developed, perhaps with new blocks, training methods or additional features that enable it to better fit the pairwise forces. A key issue is the fit currently does not guarantee energy conservation. This is not an issue with the decomposed pairwise forces which recombine to give the Cartesian forces they were decomposed from implicitly. Therefore if the Cartesian forces were energy conserving, the pairwise forces will be as well.

We saw the issues caused by the lack of energy conservation in the molecular dynamics simulations which, combined with extrapolated points caused the simulation to fail. Consequently a redesign of PFN3b would be a priority for any further work, to ensure that it implicitly predicts energy conserving forces.

Another issue which must be resolved is the generation of training data in an appropriate manner. In this work we used datasets created by running MD simulations. This does not produce a dataset which will ensure we avoid extrapolation and as we saw during the course of a simulation, predictions were made for structures that lay outside the range of the training data, even before the energy gain was large.

One must consider the chemical space likely to be explored during the course of an MD simulation and sample it appropriately to ensure extrapolation does not occur or at least implement an appropriate scheme for the identification of extrapolated points and method for dealing with them.

Molecular dynamics is an inappropriate tool for creating a good training set, it does include transitions between conformers however predominately samples around minima. Given the importance of accurate predictions both around minima and over transitions between conformers an alternative approach is required.

A much more fundamental analysis of the problem to be solved is required. In fields such as computer vision or voice recognition, the manifold hypothesis states that despite the usually high dimensionality of the data fitted the solution space lies on a much lower dimensional manifold which is constructed through successive layers of a deep learning model. It is debatable whether or not the manifold hypothesis will apply to chemical space where arguable all 3N-6 coordinates are important. Furthermore, sampling must be reasonably dense as many interactions such as hydrogen bonding or halogen bonding are directional and potentially very sensitive to changes in position.

However with the continuously increasing availability of computational power, the construction of appropriate extremely large databases of quantum calculations becomes feasible which may serve to mitigate the aforementioned issues. With such datasets available, the construction of machine learned forcefields becomes more feasible and this work represents a step along the path to the development of a good methodology for doing so.

Bibliography

[1] P. P. Kore, M. M. Mutha, R. V. Antre, R. J. Oswal and S. S. Kshirsagar, *Open Journal of Medicinal Chemistry*, 2012, **02**, 139–148.

[2] E. Bielska, X. Lucas, A. Czerwoniec, J. M. Kasprzak, K. H. Kaminska and J. M. Bujnicki, *BioTechnologia*, 2011, **3**, 249–264.

[3] K. Rege and H. G. Lemu, *IOP Conference Series: Materials Science and Engineering*, 2017, **276**, 012027.

[4] R. W. Hockney, *Methods in Computational Physics*, 1970, **9**, 135–211.

[5] W. C. Swope, H. C. Andersen, P. H. Berens and K. R. Wilson, *The Journal of Chemical Physics*, 1998, **76**, 637.

[6] J. J. P. Stewart, *Journal of Molecular Modeling*, 2007, **13**, 1173.

[7] J. Wang, R. M. Wolf, J. W. Caldwell, P. A. Kollman and D. A. Case, *Journal of Computational Chemistry*, 2004, **25**, 1157–1174.

[8] P. E. Smith and B. M. Pettitt, *Journal of Physical Chemistry*, 1994, **98**, 9700–9711.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, IEEE computer vision and pattern Recognition, 2009.

[10] X. Glorot, A. Bordes and Y. Bengio, *Proceedings of Machine Learning Research*, 2011, **15**, 315–323.

[11] G. Cybenkot, *Math. Control Signals Systems*, 1989, **2**, 303–314.

[12] K. Hornik, M. Stinchcombe and H. White, *Neural Networks*, 1989, **2**, 359–366.

[13] K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, *IEEE Signal Processing Magazine*, 2017.

[14] Y. Bengio, A. Courville and P. Vincent, *IEEE transactions on pattern analysis and machine intelligence*, 2013, **35**, 1798–1828.

[15] J. Y. F. Yam and T. W. S. Chow, *Neurocomputing*, 2000, **30**, 219–232.

[16] K. K. N. Gurney, *An introduction to neural networks*, UCL Press, 1997, pp. 67–71.

[17] G. B. Goh, N. O. Hodas and A. Vishnu, *Journal of Computational Chemistry*, 2017, **38**, 1291–1307.

[18] J. S. Smith, O. Isayev and A. E. Roitberg, *Chem. Sci.*, 2017, **8**, 3192–3203.

[19] N. Mohammadi and S. J. Mirabedini, *Journal of mathematics and computer science*, 2014, **12**, 113–123.

[20] D. P. Kingma and J. Ba, *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2014.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu and X. Zheng, *TensorFlow: A System for Large-Scale Machine Learning*, 2015.

[22] R. Reed, *IEEE Transactions on Neural Networks*, 1993, **4**, 740–747.

[23] S. Fahlman and C. Lebiere, *Advances in neural information processing systems*, 1990, 524–532.

[24] K. O. Stanley and R. Miikkulainen, *Evolutionary Computation*, 2002, **10**, 99–127.

[25] K. O. Stanley, D. B. D'Ambrosio and J. Gauci, *Artificial Life*, 2009, **15**, 185–212.

[26] S. Dieleman, K. W. Willett and J. Dambre, *Monthly Notices of the Royal Astronomical Society*, 2015, **450**, 1441–1459.

[27] M. Ragoza, J. Hochuli, E. Idrobo, J. Sunseri and D. R. Koes, *Journal of Chemical Information and Modeling*, 2017, **57**, 942–957.

[28] M. A. Nielsen, *Neural Networks and Deep Learning*, Determination Press, 2015.

[29] Y.-L. Boureau, J. Ponce and Y. Lecun, 27th International conference of machine learning, Haifa, Israel, 2010.

[30] H. Kaiming, X. Zhang, S. Ren and J. Sun, *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, 770–778.

[31] W. Rawat and Z. Wang, *Neural Computation*, 2017, **29**, 2352–2449.

[32] R. K. Srivastava, K. Greff and J. Schmidhuber, *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, 2015, 2377–2385.

[33] K. T. Schütt, H. E. Sauceda, P.-J. Kindermans, A. Tkatchenko and K.-R. Müller, *The Journal of Chemical Physics*, 2018, **148**, 241722.

[34] P. Jackson, *Introduction to expert systems*, Addison Wesley, 3rd edn, 1999, pp. 383–386.

[35] J. Behler and M. Parrinello, *Physical Review Letters*, 2007, **98**, 146401.

[36] M. Rupp, A. Tkatchenko, K.-R. Müller and O. A. von Lilienfeld, *Physical Review Letters*, 2012, **108**, 058301.

[37] R. Sibson, *Interpolating multivariate data*, 1981, ch. 2, pp. 21–36.

[38] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller and A. Tkatchenko, *Nature Communications*, 2016, **8**, year.

[39] O. T. Unke, S. Chmiela, M. Gastegger, K. T. Schütt, H. E. Sauceda and K. R. Müller, *Nature Communications 2021 12:1*, 2021, **12**, 1–14.

[40] M. Blanco, A. Martín Pendás and E. Francisco, *Journal of chemical theory and computation*, 2005, **1**, 1096–1109.

[41] T. L. Fletcher and P. L. A. Popelier, *Journal of Computational Chemistry*, 2017, **38**, 1005–1014.

[42] J. C. R. Thacker, M. A. Vincent and P. L. A. Popelier, *Chemistry A European Journal*, 2018, **24**, 11200–11210.

[43] S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt and K.-R. Müller, *American Association for the Advancement of Science*, 2016, **3**, e1603015.

[44] S. Chmiela, H. E. Sauceda, I. Poltavsky, K. R. Müller and A. Tkatchenko, *Computer Physics Communications*, 2019, **240**, 38–45.

[45] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery Jr., J. E. Peralta, F. Ogliaro, M. Bearpark, J. J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, J. M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, . Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski and D. J. Fox, *Gaussian09 Revision E.01*.

[46] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous and Y. Lecun, Proceedings of the 18th International Conference on Artificial Intelligence and Statistics, San Diego, 2015.

[47] M. Mälicke, E. Möller, H. D. Schneider and S. Müller, *scikit-gstat: Version 0.6*, 2021.

[48] B. Fornberg, E. Larsson and N. Flyer, *SIAM Journal on Scientific Computing*, 2011, **33**, 869–892.

[49] D. Case, H. Aktulga, K. Belfon, I. Ben-Shalom, S. Brozell, D. Cerutti, T. Cheatham, III, G. Cisneros, V. Cruzeiro, T. Darden, R. Duke, G. Gi-

ambasu, M. Gilson, H. Gohlke, A. Goetz, R. Harris, S. Izadi, S. Izmailov, C. Jin, K. Kasavajhala, M. Kaymak, E. King, A. Kovalenko, T. Kurtzman, T. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, M. Machado, V. Man, M. Manathunga, K. Merz, Y. Miao, O. Mikhailovskii, G. Monard, H. Nguyen, K. O'Hearn, A. Onufriev, F. Pan, S. Pantano, R. Qi, A. Rahnamoun, D. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C. Simmerling, N. Skrynnikov, J. Smith, J. Swails, R. Walker, J. Wang, H. Wei, R. Wolf, X. Wu, Y. Xue, D. York, S. Zhao and P. Kollman, *Amber20*, 2021.

[50] L. L. Duan, Y. Mei, D. Zhang, Q. G. Zhang and J. Z. Zhang, *Journal of the American Chemical Society*, 2010, **132**, 11159–11164.

[51] J. Marañón and J. R. Grigera, *Journal of Molecular Structure: THEOCHEM*, 1998, **431**, 7–15.

[52] M. J. Mills and P. L. Popelier, *Theoretical Chemistry Accounts 2012 131:3*, 2012, **131**, 1–16.

[53] K. Kato, T. Masuda, C. Watanabe, N. Miyagawa, H. Mizouchi, S. Nagase, K. Kamisaka, K. Oshima, S. Ono, H. Ueda, A. Tokuhisa, R. Kanada, M. Ohta, M. Ikeguchi, Y. Okuno, K. Fukuzawa and T. Honma, *Journal of Chemical Information and Modeling*, 2020, **60**, 3361–3368.

[54] X. Wang and J. Gao, *RSC Advances*, 2020, **10**, 666–673.

[55] A. Jakalian, D. B. Jack and C. I. Bayly, *Journal of Computational Chemistry*, 2002, **23**, 1623–1641.

[56] R. Martin and D. Heider, *Frontiers in Genetics*, 2019, **0**, 990.