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

Learning Symmetries with Machine Learning

by

Aaron Miller

Supervised by Prof. Stefano Sanvito

A final year research project submitted in partial fulfillment of the requirements for the degree of

Bachelors of Arts Moderatorship in Theoretical Physics

School of Physics
Trinity college Dublin

Acknowlegements

I would like to express my deepest appreciation to my thesis advisor Professor Stefano Sanvito, who through numerous zoom meetings, has guided this thesis with both good humour and wisdom. I would also like to extend my gratitude to PhD student Laura Gambini, who's technical expertise, and salient advice was integral to this project. In a year like no other, I could not have had better support and guidance from these two physicists. Further thanks to my housemates for their mediocre company through every stage of this thesis. Finally a massive thank you to my classmates for a fantastic four years.

Contents

1 Abstract

Machine learning methods (MLMs) [1] are rapidly becoming an alternative means of describing chemical environments. Choosing the representation of the geometric data used is an important design decision for MLMs and for physical systems,invariant representations that respect symmetry are desired. Current work achieves this using atomic descriptors [2], however questions have been raised on the completeness of this representation [3]. This thesis has investigated how neural networks (NNs) can extract invariant representations and has explored ways NNs adapt to symmetries of Euclidean coordinates. To learn invariant representations, a custom loss function based on the distance between points was introduced and invariant representations of Euclidean triangles were successfully produced. The extracted representations were found to outperform Euclidean representations in predicting the Leonard-Jones potential for a toy triangular molecule. Constrained representations of triangles with symmetries were extracted and investigated. Future work was discussed, and a new model was proposed.

2 Introduction

2.1 Motivation

In recent years, machine-learning methods (MLMs) such as neural networks (NNs) [1] have proven extremely useful in various areas such as fluid simulation and astronomy [4]. They are exceptional at pattern finding and making predictions from datasets that are difficult to interpret. In physics, these patterns are physical laws. Their success is exemplified clearly in particle physics where, for example, they have been deployed in the classification of decaying particles [5]. They are used also in the case of quantum chemistry and materials science, where one of the tasks is to correlate the geometry of a molecule to it's properties [6]. In general, an atom or material, is presented to the MLM in the form of a vector of properties, called a feature vector. These feature vectors may contain energy, spin, polarisation, coordinates, or any other feature of a system. MLMs trained on different representations of the same data are not guaranteed to be preform equally however. That is, two different vectors may contain the same information but two models may not perform similarly when trained on either representation.

When dealing with geometrical properties in chemical environments, it would be convenient to use Cartesian coordinates to specify the feature vector as they provide a

simple and unambiguous description of the system's configuration. This representation offers challenges, it is not at all suitable for comparison of geometrical environments, as there are infinitely many feature vectors representing the same geometry. These vectors are all connected by rotations and translations, mapping them to each other. These operations form the Euclidean symmetry group $\mathbb{E}(3)$. For example, take i identical triangles given by nine Euclidean coordinates $\vec{x}_i = \left(x_i^1\ y_i^1\ z_i^1\ x_i^2\ y_i^2\ z_i^2\ x_i^3\ y_i^3\ z_i^3\right)$ as displayed in figure 1, where the upper-indices $1, 2$ and 3 refer to the points of the triangle. Now from the assumption of $\mathbb{E}(3)$ invariance, we desire a model to predict identical properties for each input triangular molecule, $\vec{x}_i$. Using the Euclidean coordinates for prediction in this way is a poor approach however, as predictions are unreliable and models take a long time to converge, as both the model and coordinates do not naturally assume $\mathbb{E}(3)$ invariance. Figure 18 in the appendix illustrates this showing poor performance of an energy predictor model trained on Euclidean coordinates.

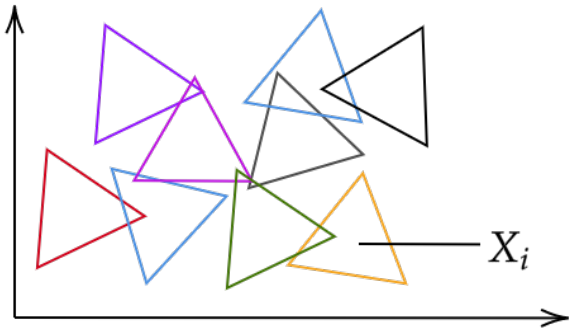


Figure 1: Diagram of identical triangles X_i related by Euclidean operations, from which we desire an MLM to make similar predictions for each triangle.

In order to create ML algorithms that deal with Euclidean coordinates and geometrical quantities, we must make our models respect $\mathbb{E}(3)$ symmetry. An approach is to identify, or approximate symmetries of the systems. Current work achieves this using many-body descriptors to create invariant representations of atomic environments [2]. These descriptors are mainly used in constructing machine-learned interatomic potentials and are functions mapping the coordinates to a representation specifying the environment, up to permutation and rotation of same atom types. These representations yield successfully predictive results, however, recent work demonstrates they do not uniquely specify molecular environment [3] and are thus not an optimal representation. This project approaches the issue of representing geometric environments using ML methods to extract invariant representations, and investigates how MLMs adapt to geometric symmetry.

2.2 Neural Networks

Neural networks (NNs) are a class of differentiable MLMs vaguely inspired by biological neural networks, also know as brains [7]. There is much hype surrounding neural networks due to their versatility, but they are simply nonlinear statistical models. They consist of 'neurons', or nodes, that take in a scalar input and output another scalar. Feed-foreword NNs consist of layers of these nodes, where the first layer takes as input training data and successive layers take in the scalar outputs of nodes from the previous layer, feeding data foreword. The final output produces a prediction and it is compared against the supplied desired output, or target data, using a *loss function*. These networks are described graphically by network diagrams, such as the three-layer feed-foreword neural network displayed in figure 2. Mathematically, the features Z_m are created from linear combinations of the inputs, X_p, and the output, Y_k, is created from linear combinations of Z_m,

$$\begin{aligned} Z_m &= \sigma_1\left(\alpha_{0m} + \alpha_m^T X\right), m = 1, \ldots, M, \\ Y_k &= \sigma_2\left(\beta_{0k} + \beta_k^T Z\right), k = 1, \ldots, K, \end{aligned} \tag{1}$$

where $Z = (Z_1, Z_2, \ldots, Z_M)$, and $Y = (Y_1, Y_2, \ldots, Y_K)$. The functions σ_1, and σ_2 are *activation functions* which introduce non-linearity into the network. The parameters α and β are the *learnable weights*. We want the output Y to approximate the target O, and the approximation is accessed by a loss function such as the mean squared error,

$$\mathcal{L}(O, Y) = \frac{1}{K}\sum_{i=1}^{K}(O_i - Y_i)^2. \tag{2}$$

The weights are iteratively updated according to the *optimization algorithm*, which is traditionally the gradient descent algorithm. In recent years advances have been made in optimizer technology, and this thesis uses the *Adagrad* optimizer [8]. This algorithm involves a derivative of the cost function, with respect to the weights imposing a continuity condition on the loss function. The weights of the system are updated proportional to the loss after a full sweep through the training set, or *epoch*, in a process called *backpropogation*. The training data is usually split into *mini-batches* with a size called the *batch size* which are fed to the network. Note that much of the mathematical detail and rigour have been omitted here for reasons of brevity. For further details consult *The elements of statistical learning* by *Friedman et. al* [1].

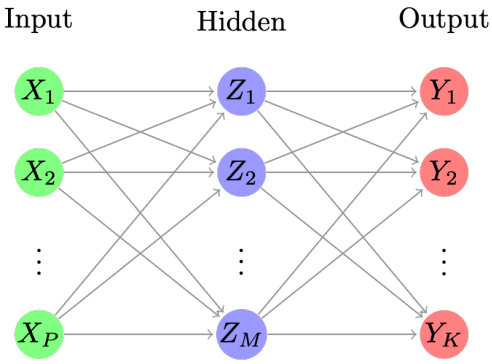


Figure 2: Schematic diagram of a three layer neural network where X,Z, and Y are the input, feature, and output vectors arranged in input, hidden, and output layers.

2.3 Autoencoders

An autoencoder (AE) is a NN utilised in the area of feature learning [9] as a class of model tasked with learning better representations of data through reproducing the input. Under-complete AEs (UAEs) do this by learning a dimensionally constrained identity function. They have two components: An encoder function $E(\vec{x}) = \vec{h}$ which maps an input vector $\vec{x} \in X$ to a lower dimensional representation $\vec{h} \in H$ of $\vec{x}$, and a decoder function $D(\vec{h}) = \vec{r}$ which creates a reconstruction $\vec{r}$ from $\vec{h}$. H is the *latent space* representation of the input data X. The operation $\mathbb{A}(\vec{x})$ of a UAE is the composition of these two functions,

$$\mathbb{A}(\vec{x}) = D(E(\vec{x})) = \vec{r} \approx \vec{x}. \tag{3}$$

Figure 3 displays a schematic diagram of this process. UAEs are encouraged to reproduce the input by a loss function defined by $\mathcal{L}(\vec{x}, \vec{r}) = 0$ when the input data and reconstruction agree, ie: $\vec{x} = \vec{r}$. Copying input to output may sound redundant, but the restriction of the dimension of H forces the model to learn an approximate representation of the data with only the important features of the input data residing in the latent space of the model [10].

A UAE with a single fully-connected hidden layer, a linear activation function and a squared error loss function, trains weights that span the same subspace as those spanned by the principal component loading vectors [11]. That is, a UAE in this configuration preforms the same dimensionality reduction operation as in principle component analysis. Thus a UAE, in certain circumstances, can be thought of as a

nonlinear generalisation of principal component analysis. AEs can also be used for generative purposes by sampling the latent space and decoding these samples. The latent space of UAEs is not usually regular, that is, two points close together in latent space are usually not decoded to similar outputs due to model nonlinearity. Variational AEs (VAEs) are a form of generative model, encouraged to create a latent space where similar inputs are encoded close together [12].

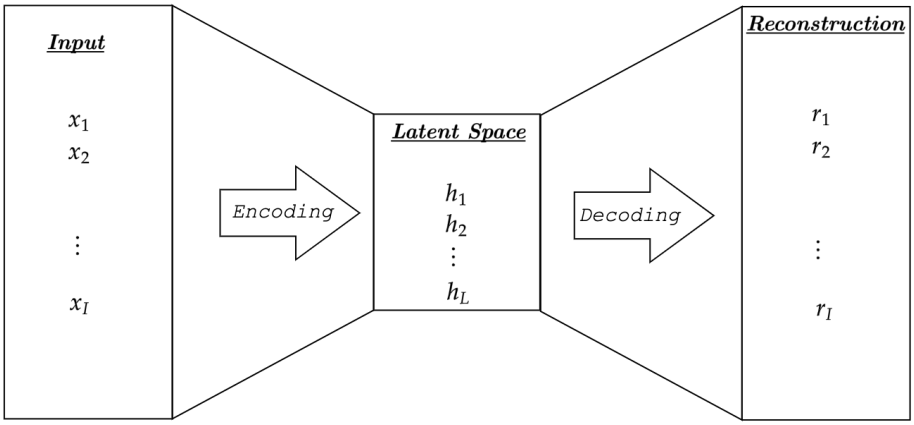


Figure 3: Schematic diagram of a UAE where an input $\vec{x} = \{x_1, x_2, .., x_I\}$ is encoded through a bottleneck to the latent space $\vec{h} = \{h_1, h_2, .., h_H\}$, and $\vec{h}$ is then decoded to a reconstruction $\vec{r} = \{r_1, r_2, .., r_I\}$ of the input.

In physics, AEs are emerging as a way to extract hidden representations of datasets. A recent article published in the *The Journal of Chemical Physics* [13] used AEs to compress physical properties, such as mass, and polarizability of atomic species. The latent representation was extracted and used along with descriptors of the geometry as a feature vector for molecular energy formation predictions, which improved previous results. This raises a question: can we use AE techniques to extract a representation of the geometric data?

3 Methods, Results, and Discussion

3.1 Autoencoding Triangular Geometries

This section investigates representation learning for a triangular toy model using AEs. The training set used consisted of 200000 triangles represented as (9×1) column vectors of the points of each vertex in Euclidean coordinates. The triangles were confined within the unit cube, as it is well established that NNs perform better on

normalized data. Stated mathematically, each vector $\vec{x}_i \in X$ for $i \in \mathbb{N}$ is given by

$$\begin{gathered} \vec{x_i} := \left(x_i^1 \; y_i^1 \; z_i^1 \; x_i^2 \; y_i^2 \; z_i^2 \; x_i^3 \; y_i^3 \; z_i^3\right)^T := \left(\mathbf{x}_i^1 \; \mathbf{x}_i^2 \; \mathbf{x}_i^3\right)^T, \\ \textit{where } \vec{x}_i \in \mathbb{R}^9, \; \mathbf{x}_i \in \mathbb{R}^3, \; min(\vec{x_i}) > 0, \; \textit{and} \; \; max(\vec{x_i}) < 1, \end{gathered} \tag{4}$$

where T is the transpose, $min(\vec{x})$ and $max(\vec{x})$ are minimum and maximum values in the vector, and $\mathbf{x}_i^1$, $\mathbf{x}_i^2$, and $\mathbf{x}_i^3$ are the points of the vertices. It is worth questioning what physical properties we want the AE to learn. Ideally, it would learn the minimum amount of data needed to uniquely represent a triangle in a coordinate-free, invariant manner. For the triangular case, this is three: three sides, two sides and one angle, or linear combinations of these quantities.

The 3-layer AE described in table 1 was used to learn a representation of the triangular dataset, where h is the latent space dimension in the model. The leaky rectified linear unit, (*Leaky ReLu*) layers add non-linearity to the model and were used due to combat the *dying ReLu problem* [14]. The *MSE* loss function from equation 2 was used along with the *Adagrad* optimiser [8] during training. With $h = 9$, this model acts as a compression-less approximation of the identity, which we use as a performance baseline. Figure 4 exhibits a log-log plot of the models loss after 120 epochs for a variety of dataset sizes. The rate of change with respect to dataset size can be seen to plateau in the region of 200000. Due to the NNs stochastic learning, an optimal dataset size is impossible to find and therefore this value is used respecting computational time limitations, while maintaining low reconstruction loss.

	Layer	*Size*	*Activation*
1.	Input	9	Leaky ReLu
2.	Encoded	h	Leaky ReLu
3.	Decoded	9	Linear

Table 1: 3-layer AE model used in section 3.1, where h is the latent dimension.

Figure 5a displays the models performance after 120 epochs against the latent dimension size, h, between one and ten. This yields a loss decreasing linearly when $1 \leq h \leq 8$ and levelling off when $9 \leq h$. This is a reasonable result, as to specify a random triangle in Euclidean coordinates you need precisely nine degrees-of-freedom, one for each value of the (9×1) input vector $\vec{x}_i$. However, the aim is to learn a coordinate-free representation of the triangle, and the *MSE* loss only measures the coordinate difference between the target and reconstruction so is not a good metric.

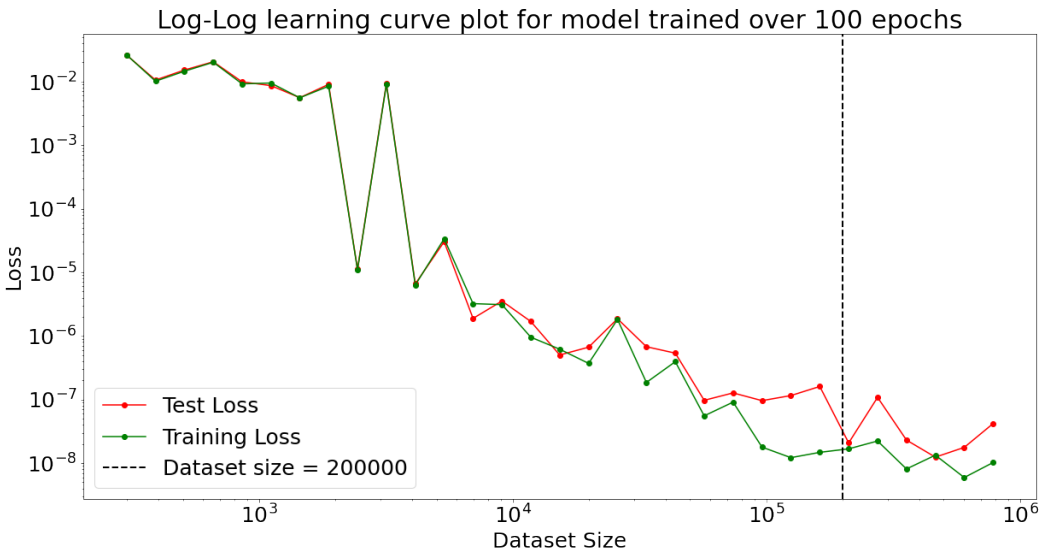


Figure 4: Log plot of the model described in table 1's training and test loss for $h = 9$, against dataset size. For each dataset the model was trained for 120 epochs and the dotted line corresponds to the selected dataset size of 200000.

A good metric for comparison is a measure of the mean total side difference,

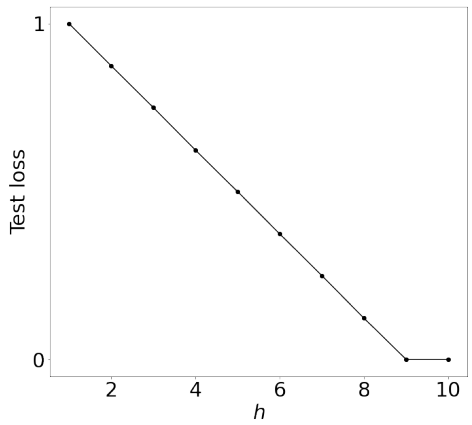
$$\mathbb{M}(\vec{x}, \vec{r}) = \frac{1}{3}\sum_{j=1}^{3} \|\mathbf{x}^j - \mathbf{x}^{(j+1)\%3}\| - \|\mathbf{r}^j - \mathbf{r}^{(j+1)\%3}\|, \tag{5}$$

where $\mathbf{x}^j$ and $\mathbf{r}^j$ are the points of the input and reconstructed triangles vertices in Euclidean coordinates and '%' is the modulo operator. Figure 5b plots the metric over the same range of h, showing us the same trend as figure Figure 5a. This implies that the representation the model is learning is entangled with the coordinates .

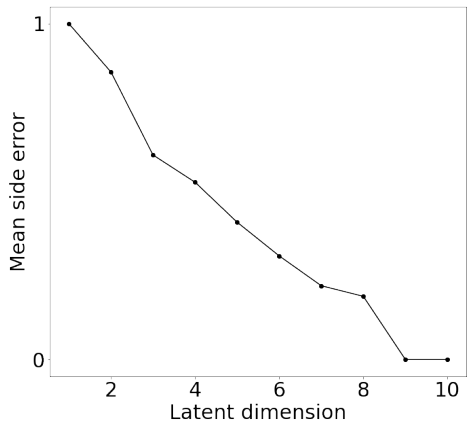
Consider now the baseline model approximating the identity function, desiring $\mathbb{A}(\vec{x}) = \vec{r} = \vec{x}$. As this is an identity function, we would expect linearity so

$$\mathbb{A}(\vec{x} + \vec{y}) = \mathbb{A}(\vec{x}) + \mathbb{A}(\vec{y}), \quad \textit{and} \quad \mathbb{A}(a\vec{x}) = a\mathbb{A}(\vec{x}), \tag{6}$$

where a is a scalar, and $\vec{x}$ and $\vec{y} \in \mathbb{R}^9$. A perfect reconstruction function would be linear, but the function $\mathbb{A}$ is nonlinear due to the leaky ReLu functions. Thus we cannot achieve perfect reconstruction as equation 6 will not hold and $\mathbb{A}$ can only be approximately linear. This deviation from the actual identity will act as a lower bound on the error, and the error will propagate through application of any learned latent representation.



(a) MSE loss (equation 2) vs h



(b) Side error (equation 5) vs h

Figure 5: Plots demonstrating the loss plateau for the model in table 1 when $h \geq 9$, and the linear relation between the reconstruction loss and h for $h \leq 8$.

3.2 Learning Translation

This section investigates how NNs adapt to translational symmetries of triangular datasets. We move away from AEs as we do not want to replicate the input, while using the network table 1. The datasets used contain 200000 triangle pairs translated according to the format described in table 2. The translations used each had norm less than one. Normalised plots of the side error and test set loss against h for the various datatypes are displayed in figure 6. Figure 7 illustrates the input and target, and reconstructions of the datatypes.

	Input data-set structure	*Output data-set structure*
a)	Same triangle on the $z = 0$ plane	Random z-translation
b)	Random triangle on the $z = 0$ plane	Constant translation
c)	Random triangle within unit cube	Random translation

Table 2: Three different translated triangular datasets used in section 3.2.

Figure 6a displays the results for dataset *a)*. The best results are obtained for $h = 1$, with the loss fluctuating as h increases. This is a feature of normalisation as the variance of the loss is low. Sampling the latent space for $h = 1$, we find a constant value for all sample triangles, meaning that for any input the network applies the same operation and the same triangle is output. This output reconstruction is the same triangle as the input, but translated by 0.5 in the z-direction as shown in figure 7a. The network has learned the mean value of the z-translation and it reconstructs

the triangle in that plane to minimize the loss. The reason one dimension is sufficient to generate acceptable reconstructions is that the weights of the network adapt to the single triangle in the training set, so no dimension is needed to store translation information.

Figure 6b visualises the loss of dataset 2 and shows the loss plateauing at $h = 6$. This value is the minimum dimension needed to represent a triangle in this dataset, as the network disregards information concerning the z-components and the weights adapt to the constant z-translation. Therefore for each point we need $(3-1)+(3-1)+(3-1) = 6$ dimensions. Figure 7b shows the reconstructions for this h-value. In the same way, figure 6c shows a plateau at $h = 9$, corresponding to 9 dimensions needed to represent the data. The network needs no dimensions to specify the translation as it is averaged over, as seen in dataset c). This is a demonstration of how AEs can be used to find the minimum degrees of freedom needed to describe certain geometric datasets.

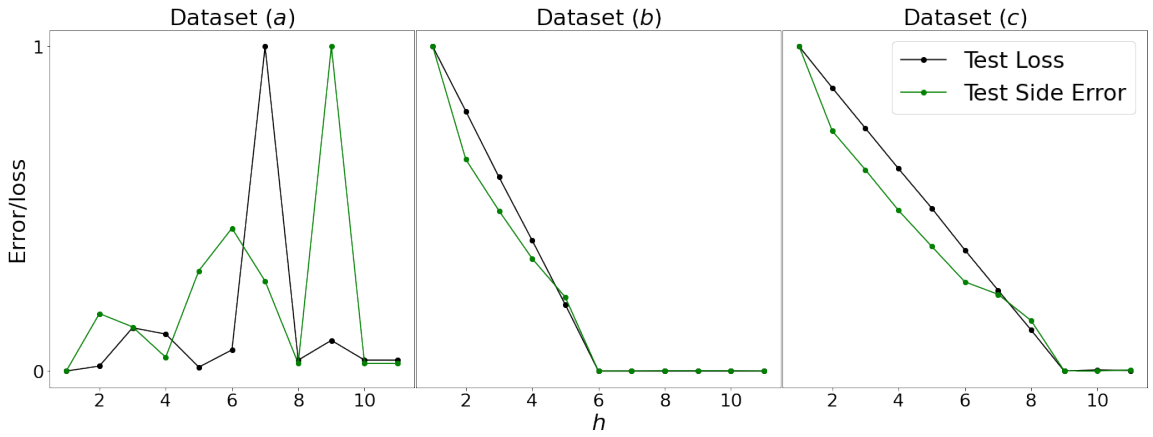
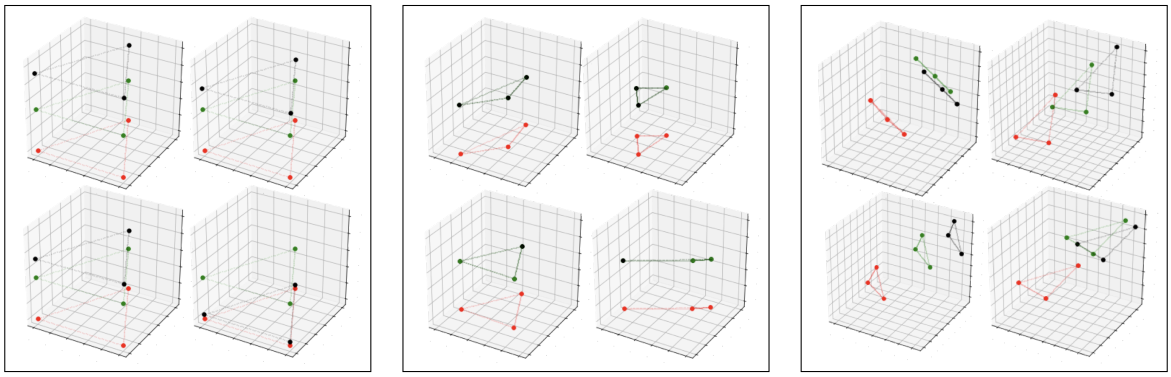


Figure 6: Normalised reconstruction loss (black line) and side errors (green line) plotted against h. Plateaus occur when the minimum degrees of freedom in the dataset is found.

3.3 Customising Loss Function

Thus far, no invariant representations have been created. The reason for this is that the model has not been incentivised to, rather, the model is trying to minimise the *MSE* loss function which encourages replication of the Euclidean feature vector. To progress, the model needs to be motivated to create an invariant representation. We can do this by changing the demands of the loss function $\mathbb{L}(\vec{x}, \vec{r})$, so that it equals



(a) (b) (c)

Figure 7: Plots illustrating the input (red), target (black) and predictions (green) from the datasets described in table 2 of the 3-layer model in table 1 where h was chosen from figure 6 as the lowest value minimizing the loss of each dataset.

zero when the metric $\mathbb{M}(\vec{x}, \vec{r})$ defined by equation 5 is zero. That is,

$$\mathbb{L}(\vec{x}, \vec{r}) = 0 \quad \textit{when} \quad \mathbb{M}(\vec{x}, \vec{r}) = 0. \tag{7}$$

This would encourage a learned representation respecting the side lengths of the input and reconstructed triangles. To translate this to code it is important to note that the loss function needs to be continuous for the backpropogation algorithm. While formulating a continuous algorithm, tensor operations from the *Tensorflow* Python3 library were used. The algorithm is broken into four steps:

3.3.1 Loss Function Algorithm

1. Reshaping input $\vec{x}$ to a 3x3 matrix,

$$\vec{x} = \begin{pmatrix} x^1 & y^1 & z^1 & x^2 & y^2 & z^2 & x^3 & y^3 & z^3 \end{pmatrix}^T \rightarrow \begin{pmatrix} x^1 & y^1 & z^1 \\ x^2 & y^2 & z^2 \\ x^3 & y^3 & z^3 \end{pmatrix}.$$

2. Permuting the rows, subtracting from the original matrix, and squaring entries,

$$\left[\begin{pmatrix} x^1 & y^1 & z^1 \\ x^2 & y^2 & z^2 \\ x^3 & y^3 & z^3 \end{pmatrix} - \begin{pmatrix} x^2 & y^2 & z^2 \\ x^3 & y^3 & z^3 \\ x^1 & y^1 & z^1 \end{pmatrix} \right]^2 = \begin{pmatrix} (x^1 - x^2)^2 & (y^1 - y^2)^2 & (z^1 - z^2)^2 \\ (x^2 - x^3)^2 & (y^2 - y^3)^2 & (z^2 - z^3)^2 \\ (x^3 - x^1)^2 & (y^3 - y^1)^2 & (z^3 - z^1)^2 \end{pmatrix}.$$

3. Summing over horizontal entries and taking piece-wise square-root ,

$$\begin{pmatrix} \sqrt{(x^1-x^2)^2+(y^1-y^2)^2+(z^1-z^2)^2} \\ \sqrt{(x^2-x^3)^2+(y^2-y^3)^2+(z^2-z^3)^2} \\ \sqrt{(x^3-x^1)^2+(y^3-y^1)^2+(z^3-z^1)^2} \end{pmatrix} = \begin{pmatrix} l^1 \\ l^2 \\ l^3 \end{pmatrix},$$

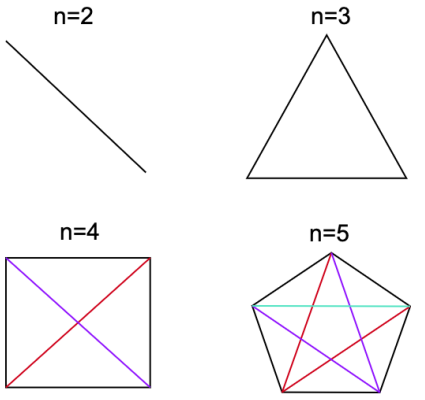
where l^1, l^2, l^3 are the side lengths of the triangle.

4. Column is summed over, process repeated for $\vec{r}$, and results are subtracted and absolute value taken,

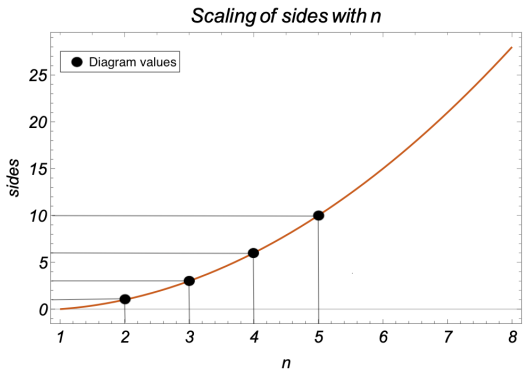
$$\mathbb{L}(\vec{x}, \vec{x}') = |(l^1 + l^2 + l^3) - (l^{1\prime} + l^{2\prime} + l^{3\prime})|. \tag{8}$$

Note that when implementing this loss function, we use batches of size bs to input the data. Thus our input is size $(bs \times 9 \times 1)$ and we have a $(bs \times 1)$ output for our loss function. The code is available on *Github* at www.github.com/diagonal-hamiltonian.

It is important to note that by using this loss function we lose some generality due to computational constraints. For instance, take an n-particle system. The number of distances specifying the geometry is $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$. This relationship tells us that the number of calculated sides in the loss scales quadratically with particle number. This becomes computationally unmanageable, as for every increase of n there are $\frac{(n+1)n}{2} - \frac{n(n-1)}{2} = n$ more sides to calculate, and after $n = 19$ we have more sides then the input Euclidean coordinates. This scaling is illustrated in figure 8a and 8b.



(a) Diagrams of the first four geometries where $n = 1, 2, 3, 4$.



(b) Sides scaling $n(n-1)/2$: Values corresponding to 8a are black dots.

Figure 8: Number of sides between n points for $n = 2, 3, 4, 5$ given by $n(n-1)/2$ which is plotted in 8b. In 8a exterior sides are black and different colour are distinct lines between each interior vertex.

To extract a representation of the triangles, the loss function was used in conjunction with the model described in table 3, as it was found experimentally to create good reconstructions from a latent space dimension $h = 3$. The encoding and decoding layers were introduced in this model to perform the operations needed to find Euclidean distances and decode the latent space. Here, 256 nodes were chosen for these layers as 256 it is a power of four, and this fact is know to bolster GPU performance [15]. The model achieved a reconstruction error of 0.0020 after 2000 epochs, which is $\times 82$ better than the same setup using the *MSE* loss function which gave an error of 0.16.

	Layer	*Size*	*Activation*
1.	Input	9	Leaky ReLu
2.	Encoding	256	Leaky ReLu
3.	Encoded	h	Leaky ReLu
4.	Decoding	256	Leaky ReLu
5.	Decoded	9	Linear

Table 3: 5-layer AE used the loss function from equation 8. The latent dimension h is 3 for sections 3.3 and 3.4, but is reduced to 2 for section 3.5

To investigate the representation created by the model, the latent space was sampled by splitting it into its encoder and decoder functions. The encoder was sampled using a data set consisting of 1000 distinct random triangles each rotated and translated randomly 500 times within the unit cube. The resulting latent space is shown in figure 16, where the 1000 distinct triangles are mapped with the same colour 500 times. We observe blobs of tightly packed points, showing that the 500 triangles are encoded in the same region of latent space. The representation is thus invariant under rotation and translation. Appendix section 5.1 visualises the AE mapping and tells us that we can interpret the latent space as a nonlinear combination of side lengths.

3.4 Predicting the Leonard-Jones (L-J) Potential

Having extracted an $\mathbb{E}(3)$ invariant representation of the dataset, we apply it to the task of energy-prediction of a triangular molecule given by the dimensionless L-J potential. The potential energy V, energy of the i^{th} particle ϵ_i, and the total molecular energy E is given by:

$$V_{\text{LJ}}(r) = \left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}, \tag{9}$$

$$\epsilon_i(r_{ij}) = \sum_{j \neq i} \left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6} \quad \text{for} \quad i, j \in [1, 2, 3], \tag{10}$$

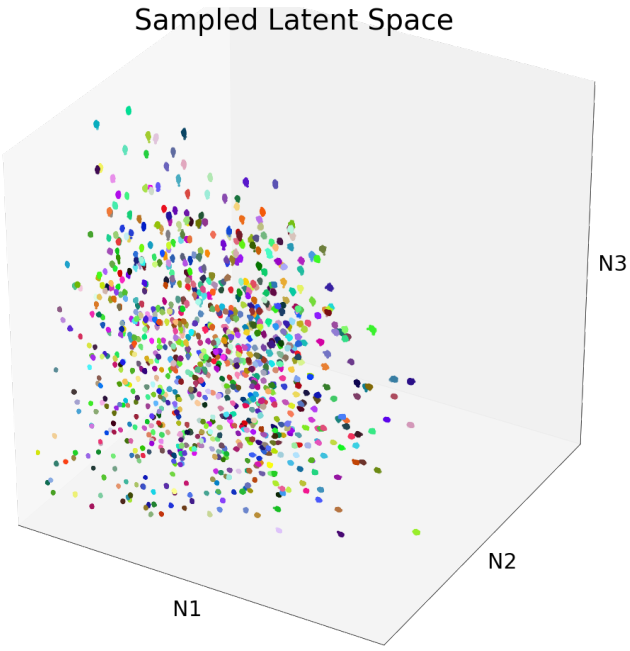


Figure 9: Sampled 3D-latent space for the model in table 3, where N_1, N_2, and N_3 are the node activations. The sampling dataset contained 500 translated, and rotated pairs of 1000 distinct Euclidean triangles each with a common random color shared between the 500 pairs. Blobs of common colour emerge meaning similar triangles are encoded to the same region of latent space.

$$E(\epsilon_1, \epsilon_2, \epsilon_3) = 2(\epsilon_1 + \epsilon_2 + \epsilon_3), \tag{11}$$

where r_{ij} is the distance between the i^{th} and j^{th} atom, and σ parameterises the model with units of length. The performance of the predictor was tested on Euclidean, side length and latent space data. The triangles were conditioned with a minimum length of 0.48 as molecules forming at short distances like this are non-physical due to exploding potential. The predictor's architecture is shown in table 4, and it was trained for 500 epochs on the different datatypes. A schematic of the MLM is displayed in figure. 10

	Layer	*Size*	*Activation*
1.	Input	I	-
2.	Hidden Layer	64	Leaky ReLu
3.	Hidden Layer	32	Leaky ReLu
4.	Hidden Layer	16	Leaky ReLu
5.	Output	1	Linear

Table 4: Energy predictor model used with a *MSE* loss, and ADAgrad optimizer. Various input types of dimension I are tested using this NN.

The model converges to losses of $6 \cdot 10^{-6}$, $9 \cdot 10^{-5}$, and $4 \cdot 10^{-4}$ for the side length,

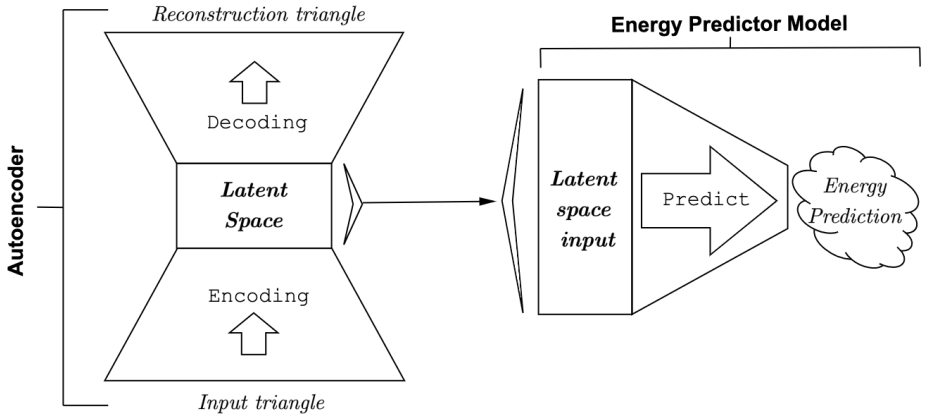


Figure 10: Energy predictor MLM schematic where a latent space is created by the AE and then used as input for a predictor model.

latent and Euclidean data-types. Figure 11 displays the normalised actual versus predicted energy and figure 18 in the appendix shows the reconstructions of the molecule after varying a triangle's side. The side length data performs best which is expected when predicting a function of side length. The latent data is an order of magnitude, and the Euclidean data is two orders of magnitude less accurate. The predictions become less accurate at higher energies as seen in figure 11 due to low sampling of the shorter side lengths. These results show we can effectively apply the extracted representations, however, we have observed it is best to use exact representations due to error from the encoding process. In situations where the direct representation is ambiguous, this method could be applied to find a more faithful representation of the data.

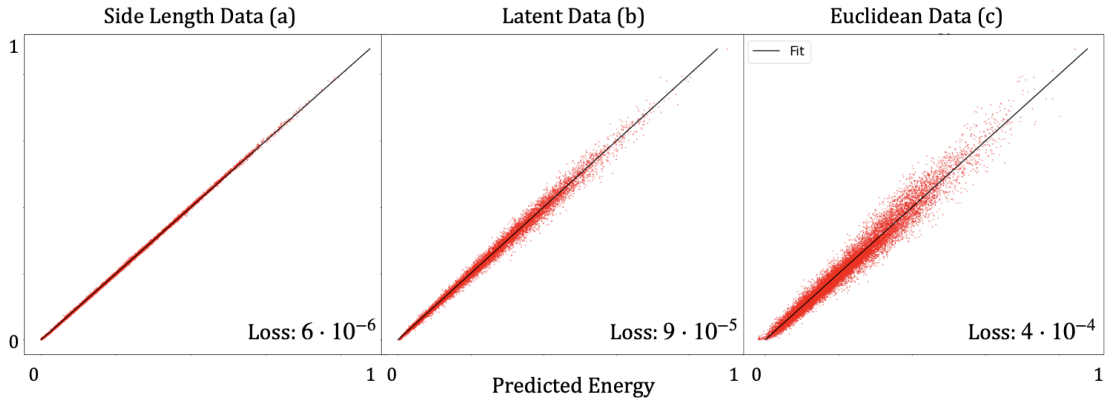


Figure 11: Normalised actual against predicted energy from the NN in table 4 for side (a), latent (b), and Euclidean (c) datasets. Fitted black lines show optimal performance when predicted and actual energy agree. Losses are displayed.

3.5 Hidden Symmetries

A NNs ability to find hidden structure within data-sets is highly regarded. Therefore it is natural to investigate how the AE and energy predictor model adapt to constrained data. The constraint used to test the performance was a constant side length in the triangular Euclidean training data. Three data-sets of triangular Euclidean data were encoded by AEs with a latent space dimension of $h = 2$. One with random triangular data for comparison, another with data containing a constant side between the same two points and a third with a constant side between two random points. L-J potential energy predictions were made from a NN predictor structured as table 4 and trained on the latent spaces produced from encoding these datasets, and also on an unencoded set containing two randomly selected sides of a triangle with a constant side. The datasets and results are summarised in table 5, and a plot of the reconstruction loss against latent space size for the various datasets are shown in figure 12.

	2D Input Data	*Rec. Loss*	*L-J Loss*
i)	Random Euclidean data, encoded	$6 \cdot 10^{-2}$	$2 \cdot 10^{-3}$
ii)	Euclidean data, one constant side, encoded	$2 \cdot 10^{-1}$	$1 \cdot 10^{-2}$
iii)	Euclidean data, same constant side, encoded	$5 \cdot 10^{-3}$	$8 \cdot 10^{-5}$
iv)	Two triangle Side lengths, one constant side	-	$5 \cdot 10^{-3}$

Table 5: Table of dataset types with reconstruction (Rec.), and L-J losses.

Figure 12 compares the reconstruction loss over a range of h for the AE trained on data-types *i)*, *ii)*, and *iii)* in table 5. The blue line shows the loss for datatype *iii)* plateauing at $h = 2$, which is the minimum dimension needed to represent a triangle with a constant side length. The green line shows the results for datatype *ii)* which contains the same information as *i)*, yet with a greater loss at $h = 2$. The model therefore, does not understand the symmetry as well as with dataset *i)* due to the constant side being between random points. This demonstrates the need to consider how NNs adapt to unintentional symmetries introduced when creating datasets. Dataset *ii)* shows improved performance from the random dataset *i)*, showing information can be learned from hidden symmetries. The energy predictions in table 5 of the models trained on the two-dimensional latent spaces, show that a lower reconstruction loss corresponds to a lower prediction loss, as expected.

We now interpret the results of our MLM for the different datatypes in table 5 using figure 13. The first two rows of the figure visualise how the AE encodes the

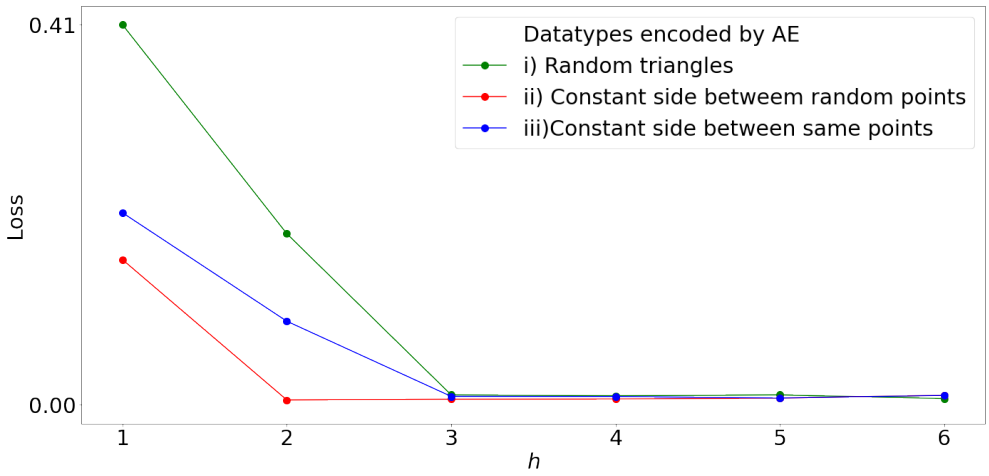


Figure 12: Loss of the datasets *i)* (blue), *ii)* (green), and *iii)* (red) described in table 5 against latent space size. As expected we observe dataset *i)* performing best when $h \leq 2$, followed by *ii)* and *iii)*.

Euclidean data (represented as side lengths) to a 2D latent space, and then how the L-J energy predictions are made from the space. A colour mapping conditional to the total side lengths illustrates where similar triangles are mapped throughout. The L-J energy predictions are plotted against the summed input as opposed to the total side lengths as it gives us a clearer idea of why the network outputs given its inputs.

For datatype *iii)* we observe that the gradient of colour is preserved between the side length plot, and the latent space. This is a good sign and shows that similar triangles are encoded close together in the latent space. This produces faithful energy predictions as can be seen from the linear actual versus predicted graph. The energy curves as a function of total node length here are similar to the energy curve as a function of total side length (see figure 14a), due to linear relationship between total node length and total side length as we saw in appendix 16. The lengths plot for *ii)* shows similar data as *iii)* but due to the permutation of the data there is no longer a distinct plane defined by the constant side length which is observed in *iii)*. For the case of *iii)* the AE finds the side lengths, and projects onto the 2D latent space, but in *ii)* this projection is more difficult due to the permutation. This causes a malformed latent space leading to less faithful results. Case *i)* further exemplifies this malformed latent space which is heavily mixed due to the uncertainty introduced from encoding a 2D latent space, when exactly 3-dimensions are needed to represent the triangles in the dataset. The mixed latent space of *i)* generates useless predictions as shown by the actual and predicted energy plots.

The actual versus predicted plot in figure 13i) is peculiar as it shows an abrupt cut-off at low energies below 0.15, which is uncharacteristic of NNs. The explanation is clear once we consider figure 14 produced from the model trained on datatype *iv)*. This model gives us an idea of how uncertainty propagates through the predictor as the data is two random sides of a triangle where one side is constant. From 14a we see that there is a low energy cut-off when plotted against the total side lengths. Figure 14b shows actual and predicted energy curves against the summed input lengths for when the input contains the constant side. The spread of the actual data shows the unachievable predictions that we ask the network to make. Unachievable as the side containing essential information is randomly removed. The predicted data forms a line resembling the L-J curve, and it shows that the network deals with uncertainty in the same way as with random translations as discussed in section 3.2, by averaging over the uncertainty. Figure 14c resembles the actual versus predicted energy graph of figure 13i. It shows that when the network is given two random sides it preforms well as there is no uncertainty, but when the network is given the constant side the predictions are poor, as there is uncertainty about the other side length. Now, as this graph is of similar shape to the corresponding graph in figure 13i, we deduce that this problem of uncertainty is inherent to the encoded representation.

3.6 Future Work

This thesis was completed during the COVID-19 pandemic, which unfortunately did not allow for the full exploration of this rich topic. Extending the ideas developed here to multiple molecules with different types of atoms, with higher degrees of freedom, would be a test of scalability. Optimal representations from these high degree-of-freedom geometries may be unknown and could be extracted. Different AE networks such as VAEs would also have been explored as ways to extract representations.

Extracting invariant latent representations have proved successful, however it undoubtedly took a massaging of the loss function. This reduced generality due to the implementation of an operation scaling the number of calculations needed in the loss function quadratic with number of molecules. To combat this, this thesis proposes a network that may learn invariant representations of data more generally. Consider a data structure like that of section 3.4 with input coordinates and target energy. The $\mathbb{E}(3)$ invariant quantities of the system are defined in the relationship between the coordinates and the potential. Therefore it makes sense to learn a representation from the physical laws which are naturally invariant. A diagrammatic representation of a

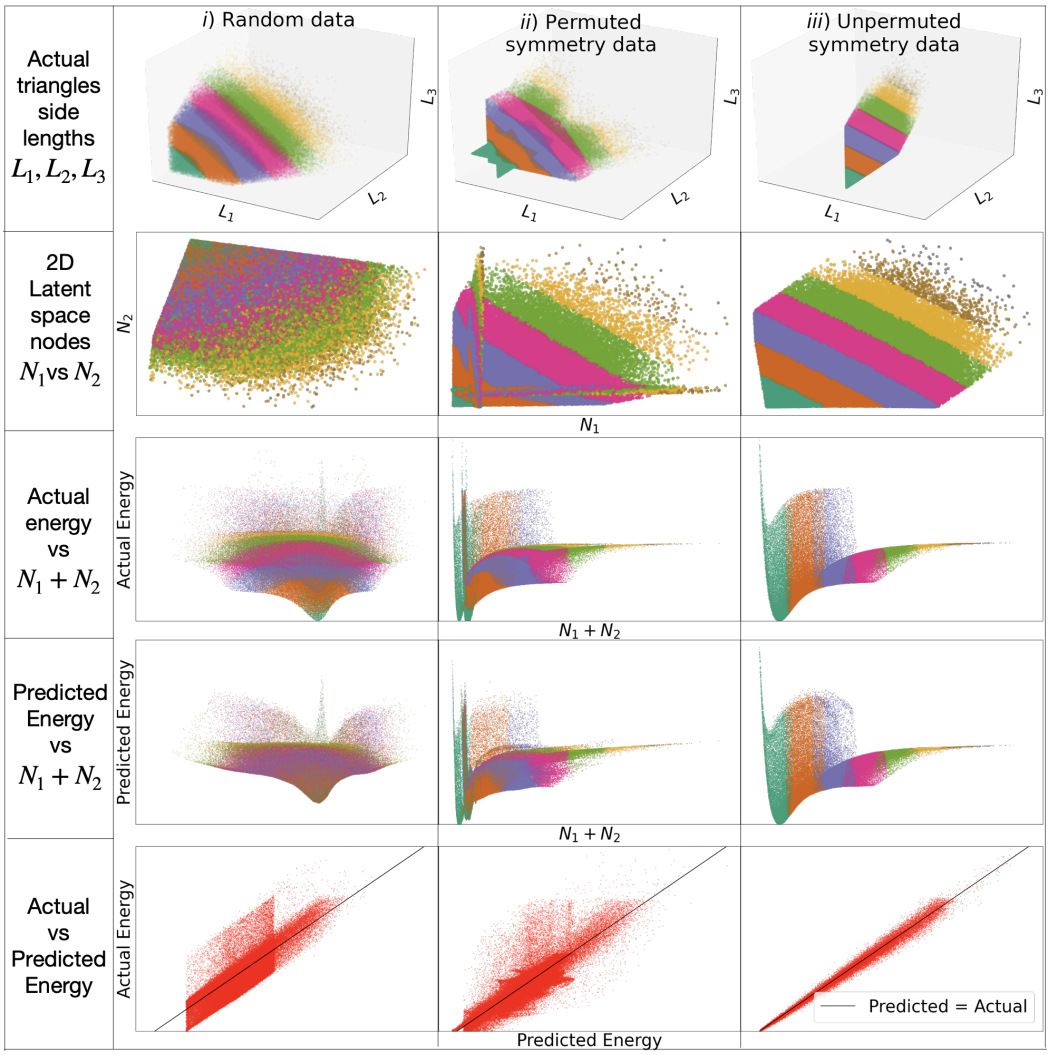
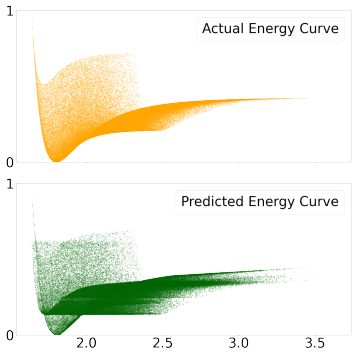
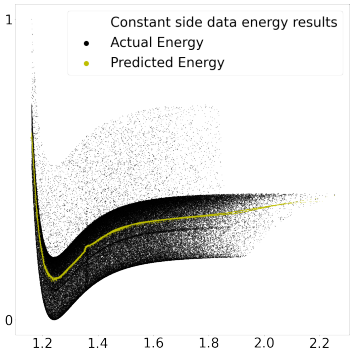


Figure 13: Column one shows a color mapping from sampled triangles represented as side lengths, to a 2D latent space. The second column displays actual, and predicted energy values from the latent space. The third column exhibits the versus predicted energies. Rows i), ii), and iii) correspond to data-types in table 5.

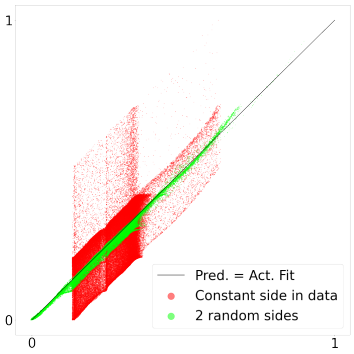
model attempting this is displayed in figure 15. The model takes coordinates as input and energy as target data. The input is encoded into a latent space and a reconstruction and energy prediction is made from this space. A loss function accesses the performance of these operations and the weights are updated accordingly. This could encourage representation learning by acting as an AE, while also enforcing invariance through prediction of the energy.



(a) Energy vs $L_1 + L_2 + L_3$



(b) Energy vs $L_1 + L_2$



(c) Actual vs Pred. energy

Figure 14: Results from predictor trained on dataset iii), two side lengths. Black(Yellow) in 14b are actual(predicted) energies given two sides with one constant. Red(Green) in 14c show predictions given one(two) random side(s).

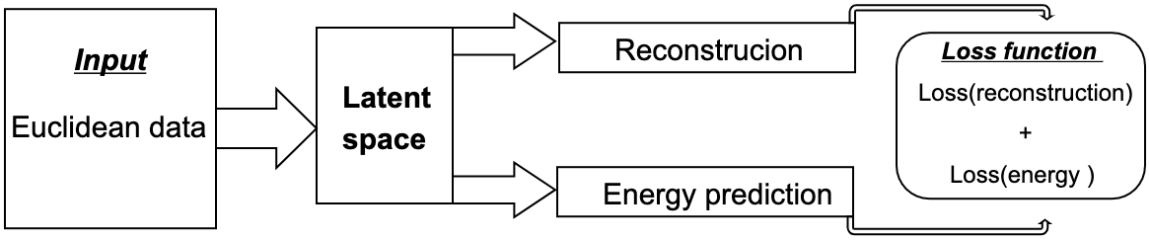


Figure 15: Schematic representation of a proposed model attempting to learn an invariant representation coordinate-energy relationship.

4 Conclusion

In this thesis, an understanding of how small neural networks adapt to symmetries within geometric datasets has been developed which is essential for understanding how NNs work when applied at scale in areas such as condensed matter physics. Symmetry was understood by training NNs to extract invariant representations of data. In particular, attempts were made using standard AE design to extract representations of triangular molecules represented by nine Euclidean coordinates. This was unsuccessful as the network attempts to replicate coordinates. A NN was applied to learning translation of various triangular data sets and it was found to adjust to translational symmetry through the weights of the network, by averaging over translations. The model was also shown to act as a means of finding the minimum information needed to specify a system. In order to mitigate replication of coordinates and incentivise representation learning, a new loss function 8 was proposed and used, instead of the *MSE* loss used previously. This was successful and produced a lower-dimensional in-

variant representation of data within the latent space of the AE. The latent space was used to predict the Leonard-Jones potential of a triangular molecule 11. Section 3.4 shows that an energy predictor model trained on the latent space, outperforms a model trained on Euclidean data, but underperform in comparison to side length training data, due to reconstruction error in the encoding process. The adaptation of this network to a hidden symmetry of a constant side length was tested and it was found that the network adapts well and it was demonstrated how the NNs may adapt to unintentional symmetries in data sets.

Geometric representation extraction is an interesting area, where it has been shown NNs can be applied and there is room for further work as discussed in section 3.6. It must be noted that the custom loss function developed here, was used as a proof of concept and considerations to optimisation of this and the other models developed here, should be considered in future.

References

[1] J. Friedman, T. Hastie, R. Tibshirani, et al., *The elements of statistical learning*, Vol. 1, 10 (Springer series in statistics New York, 2001).

[2] J. Behler, International Journal of Quantum Chemistry **115**, 1032 (2015).

[3] S. N. Pozdnyakov, M. J. Willatt, A. P. Bartók, C. Ortner, G. Csányi, and M. Ceriotti, Physical Review Letters **125**, 10.1103/physrevlett.125.166001 (2020).

[4] G. Carleo, I. Cirac, K. Cranmer, L. Daudet, M. Schuld, N. Tishby, L. Vogt-Maranto, and L. Zdeborová, Reviews of Modern Physics **91**, 045002 (2019).

[5] C. collaboration et al., Journal of Instrumentation **15** (2020).

[6] R. Ramprasad, R. Batra, G. Pilania, A. Mannodi-Kanakkithodi, and C. Kim, npj Computational Materials **3**, 1 (2017).

[7] J. Jesan and D. Lauro, Ubiquity **2003**, 2 (2003).

[8] J. Duchi, E. Hazan, and Y. Singer, Journal of machine learning research **12** (2011).

[9] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: a review and new perspectives*, 2014.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning* (MIT Press, 2016).

[11] E. Plaut, *From principal subspaces to principal components with linear autoencoders*, 2018.

[12] C. Doersch, arXiv preprint arXiv:1606.05908 (2016).

[13] J. E. Herr, K. Koh, K. Yao, and J. Parkhill, The Journal of Chemical Physics **151**, 084103 (2019).

[14] L. Lu, Communications in Computational Physics **28**, 1671 (2020).

[15] Nvidia, *Deep learning preformance documentation*, (2020-07-17) https://docs.nvidia.com/deeplearning/performance/dl-performance-convolutional/index.html (visited on 03/15/2021).

5 Appendix

5.1 Sampling the Custom Loss Autoencoder Model

The AE from table 3 was sampled using the same dataset of triangles as 3.1. Figure 17 visualises this sampling where each Euclidean triangle is represented using three side lengths L_1, L_2, and L_3, in 3D-space with a colour map showing where similar triangles are mapped through the AE. A relationship between triangular side lengths and the encoded latent space activations is evident. From figure 16 we can see that node activations are almost linear as a function of different linear combinations of the side lengths. This implies that the latent space can be vaguely interpreted as an almost linear combination of the side lengths, but the spread of the graphs may contain needed reconstruction information due to network non-linearity.

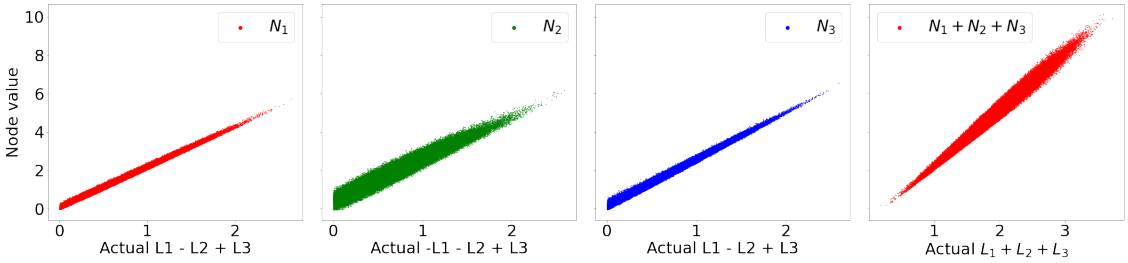
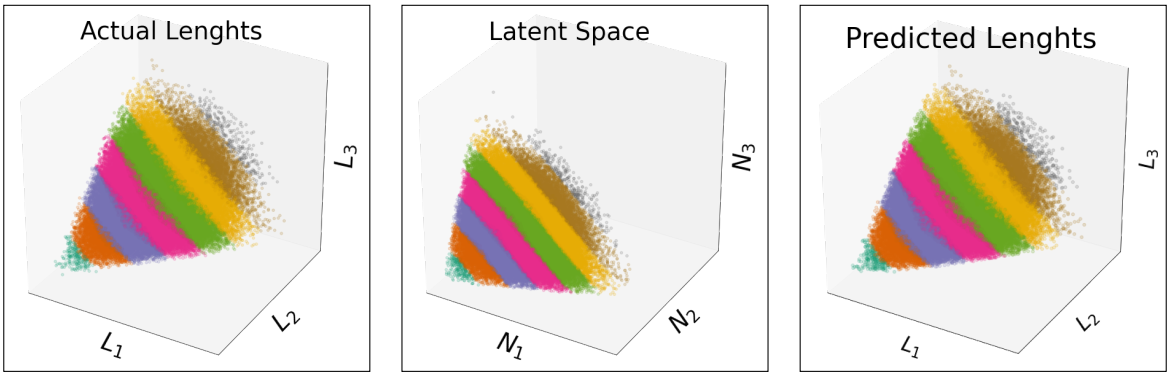


Figure 16: Graphs displaying latent node activations for the model in table 3 against linear combinations of side lengths.

5.2 Supplementary Figures



(a) Input side lengths | (b) Latent space activation's | (c) Output side lengths

Figure 17: Side length visualisation of input (17a) and predicted triangles (17c), with a colour mapping each triangle through the latent space 17b each diagram.

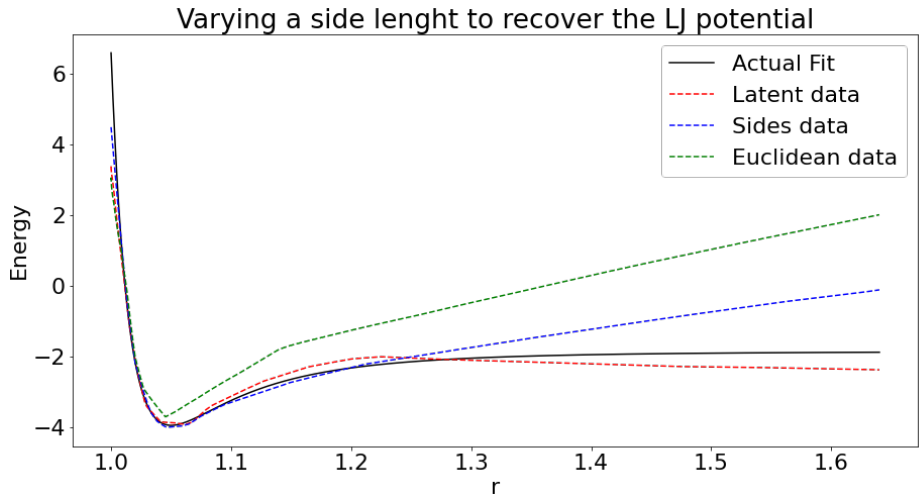


Figure 18: Varying a side length to recover the L-J potential for various datatypes described in the plot legend.