Master of Science in Industrial and Applied Mathematics
(MSIAM M2)
at
Grenoble INP ENSIMAG University of Grenoble Alpes
Grenoble, France

Variable Selection of the Parrinello-Behler Descriptors for Neural-Network Potentials in Material Science

Ghimire Suraj

Academic Session:- 2020-2021

Research project performed at Grenoble INP SiMaP

Under the supervision of:

Noel JAKSE
SIMAP, Grenoble

Emilie DEVIJVER
LIG, Grenoble

Defended before a jury composed of:
Pierre Etoré
Noel JAKSE
Emilie DEVIJVER

August 2021 Grenoble, France

Abstract

The purpose of this Master's Thesis is to have preliminary research dedicated to the construction of empirical potentials with a systematic and automated search of the optimal set of parameters. This will be carried out by means of machine learning techniques inspired from artificial intelligence concepts, using large scale ab initio databases, more precisely variable selection on structural descriptors of the Behler-Parrinello form of the database will be used to train and test anÂ Artificial Neural Network to fit empirical potentials with ab initio accuracy.

We first introduce the concept of Behler Parrinello methods and High Dimensional Neural Network Potentials (HDNNP). We describe datasets, various parameters, and hyperparameters used to train and predict the model.

Then we look at various NN Models and training processes where we search for hyperparameters combinations that predicts Potentials with high precision. What will be the change in loss function when we increase our features from 22 to 329? Can this NN be replaced with Random Forest for efficient prediction of Potentials?

As we model with different numbers of variables, we look on to the possibility of selecting variables on the basis of dropping variables with the respect to weights or least validation loss. We also look at the significance test of the models with the best-performing model.

Finally, we briefly talk about the approach of Lasso Net that can bring automation to the variable selection of the optimal set of parameters.

Acknowledgement

I would like to express my sincere gratitude to everyone who has been part of the journey of my life since I came to France. Your pleasant words and hands of support will always be a memorable moment for me. This journey wouldn't be complete without the support of my family, and I am thankful to them.

I thank my supervisors Noel and Emilie from the bottom of my heart for their effort and contribution during my internship SIMAP. Material Science has been a new field for me, but all these days, Noel and Emilie have been helpful and extremely supportive by answering every small doubt from Material Science to Neural Networks. As a tutor, and as a human being, I shall remember them for a lifetime.

These six months will be a cheerful moment of my life.

All Glory to God.

Contents

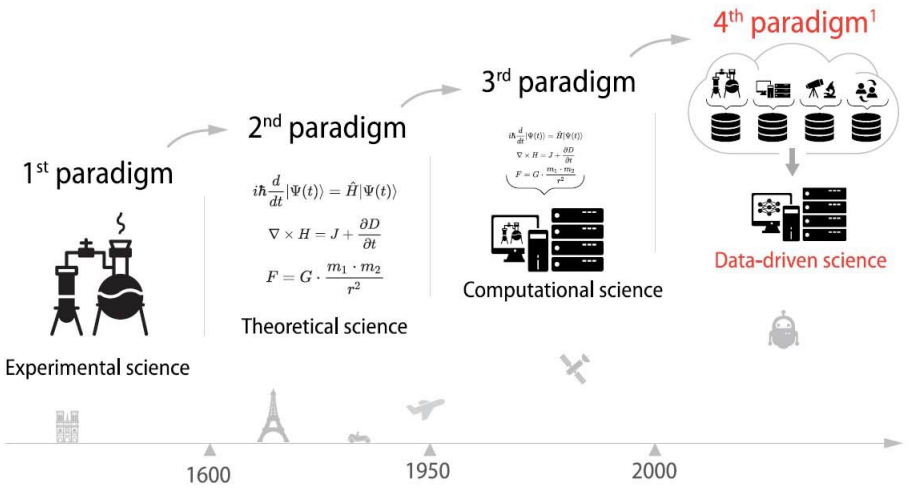
List of Figures

List of Tables

— 1 —
Introduction

Aluminum alloys represent, after steels, extremely attractive metallic materials for many applications. Due to their lightweight, high ductility, and high strength, they enter a broad range of use in strong structures with high engineering added value for the automotive, aeronautic, and aerospace industries. Challenges still face researchers from a fundamental point of view in the understanding of the structure and dynamic properties and their interplay at the atomic scale for better control and efficient industrial casting methods.

There is now some evidence that structural features not only at the local atomic scale but also in the medium range might govern the early stage of crystal nucleation. This is particularly important for rapid (additive) solidification where it is desirable to lower the dynamics, i.e. diffusion phenomena, in the liquid while taking advantage of equiaxed nucleation.



[1] Jim Gray, 2007 [GRAY]

Figure 1.1: Paradigm cycle of Molecular Science

Although the investigation of the local order is routinely performed even with ab initio simulation techniques, this is not the case for the intermediate-range order. The latter requires in addition large-scale molecular dynamics simulations with empirical potentials with the goal of

investigating systematically the medium range-order and treating more complex alloys.

Physical Experiments needs measurements and results obtained are expressed in numeric form. The set of mathematical equations of any model needs to be validated by its ability to describe the behaviour of system within certain pre-defined constraints. In computer experiments such model still need to be validated, but such calculations can be carried out by machine. This process has helped to simulate physical problems of extreme interest which would be difficult to perform with the experiments. The figure 1.1 shows timeline of different paradigms of Science.

Molecular dynamics (MD) is a computer simulation method to analyze the physical movement of atoms and molecules. The time evolution of sets of interacting atoms is followed by integrating their equation of motion. Molecular dynamics follow the law of classical mechanics law. The dynamics of each atom is given by the Newton's second law of motion

$$\mathbf{F}_i = m_i \mathbf{a}_i \equiv m_i \frac{\mathrm{d}^2 \mathbf{r}_i}{\mathrm{d}t^2}$$

where

- m_i is mass of the atom
- $\mathbf{F}_i$ is the force on the atom i
- a is its proper acceleration.

Molecular dynamics method helps us to investigate materials in the fine details at the atomic scale. This method will be reliable only if interactions between atoms are known accurately from quantum mechanics (ab initio) but this approach is too heavy to treat more than 10^2 atoms.

This is where Neural Network (NN) comes into play to construct atomic interactions via potential energies from large ab initio databases. The Behler Parrinello methods are introduced to translate local atomic environments in appropriate features for the NNs.

The objective of this thesis is to focus on the variable selection of an existing Ab-initio database. This database is extended with a different range of temperature and existing Deep learning tools were used for variable selection on structural descriptors of the Behler-Parrinello so as to optimize the empirical loss.

In Chapter two, we describe the construction of High Dimensional Neural Network potentials where we start from Behler Parinello's description and the algorithm used to simulate this approach. The deep-learning algorithm and different parameters and hyper parameters used to hyper tuning are explained in this section. We built our databases with 329 different variables and 21 different ranges of temperatures. The detailed methods of building of database are explained.

In chapter three, we explain different methods we have used in this approach. In addition to deep learning, we also used the random forest to test our dataset which we have mentioned.

In chapter four, we narrow down our number of features from 329 to 57. We compare this model with other sets of variables to find the optimal sets of parameters that give us the least

loss. In the same part, we also compare our results from the analytical and physical perception. We also explain other possible forms of variable selections.

In chapter five, we conclude our findings.

This research is an extension work on the Master thesis [7] by Anthony Saliou which he did under the supervision of Prof Noel Jakse in the SIMAP laboratory of Grenoble INP in 2020. Anthony in his thesis had constructed a High dimensional Neural Network.

This work is done under the auspices of the Chair of Materials in the Multidisciplinary Institute in Artificial intelligence (MIAI) through SIMAP and LIG laboratories in Grenoble as well as the Institute of Materials Physics in Space of the German Aerospace Centre (DLR) in Cologne, Germany.

— 2 —

Constructing High Dimensional Neural Network Potentials

In this section, we will talk about the approach of Behler Parrinello descriptors to construct a high-dimensional artificial neural network. We will also describe our neural network and hyperparameters used to hyper tune our algorithm.

2.1 Behler Parrinello Methods

The Behler-Parrinello method aims to represent Potential Energy Surfaces (PESs) using an ANN. This ab-into method gives a more precise description of material using microscopic features based on the atomic positions within the material. The need is to predict the energy from atom positions within a given configuration.

Cartesian coordinates are not a good choice to describe the structural input of NN potentials [1]. One of the solutions is to describe molecular structures using internal coordinates such as inter-atomic distances. The cutoff function of the functional forms of the symmetry functions is given as:-

$$f_{\mathrm{c}}\left(R_{ij}\right) = \begin{cases} 0.5 \cdot \left[\cos\left(\frac{\pi R_{ij}}{R_{\mathrm{c}}}\right) + 1\right] & \text{for } R_{ij} \leq R_{\mathrm{c}} \\ 0 & \text{for } R_{ij} > R_{\mathrm{c}} \end{cases} \tag{2.1}$$

where R_{ij} is the distance between atoms i and j, R_c is the cutoff Radius
These functional forms help us build symmetry functions. Behler has proposed three radial symmetry functions to describe the radial environment of i^{th} atom as

$$G_i^1 = \sum_j f_{\mathrm{c}}\left(R_{ij}\right)$$

$$G_i^2 = \sum_j e^{-\eta\left(R_{ij}-R_s\right)^2} \cdot f_{\mathrm{c}}\left(R_{ij}\right)$$

and

$$G_i^3 = \sum_j \cos\left(\kappa R_{ij}\right) \cdot f_{\mathrm{c}}\left(R_{ij}\right)$$

G^1 is nothing but the sum of cutoff functions with respect to j^{th} atom while G^2 is a sum of Gaussian functions multiplied with the cutoff function f_c

He further proposed two types of angular functions as summation of cosine functions of the angles $\theta_{ijk} = \text{acos}\left(\mathbf{R}_{\text{ij}} \cdot \mathbf{R}_{\text{ik}} / R_{ij} \cdot R_{ik}\right)$ defined as

$$G_i^4 = 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all}} \left(1 + \lambda \cos\theta_{ijk}\right)^{\zeta} \cdot e^{-\eta\left(R_{ij}^2 + R_{ik}^2 + R_{jk}^2\right)} \cdot f_{\text{c}}\left(R_{ij}\right) \cdot f_{\text{c}}\left(R_{ik}\right) \cdot f_{\text{c}}\left(R_{jk}\right)$$

and

$$G_i^5 = 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all}} \left(1 + \lambda \cos\theta_{ijk}\right)^{\zeta} \cdot e^{-\eta\left(R_{ij}^2 + R_{ik}^2\right)} \cdot f_{\text{c}}\left(R_{ij}\right) \cdot f_{\text{c}}\left(R_{ik}\right)$$

Anthony Siliou [7] worked with G2 and G5 functions, one each from angular and radial in his work of LJ and Al. We shall be using G_i^2 and G_i^5 to create our data.

2.2 Introduction to Dataset

The data to be trained is in a different environment that includes solid and liquid state. These dataset also represents state of high pressure and normal pressure.

The initial data had followed characteristics.

Temperature	State	#NO. of obs
10 K	Solid	1000 obs
300 K	Solid	342 obs
500 K	Solid	344 obs
800 K		1000 obs
1000 K	Liquid	1000 obs
1250 K	Liquid	1000 obs
1350 K	Liquid	1000 obs
1500 K	Liquid	1000 obs
1600 K	Liquid	1000 obs
1700 K	Liquid	1000 obs
1500 KH	Liquid	346 obs
Total		9032 obs

Table 2.1: The initial Dataset

Each observation was configured with 256 atoms and they have different energy levels at that point. This also means our initial data had $9032 \times 256 = 2,312,192$ rows of data + one header.

Overall, our dataset has G2, G5 variables, Forces in three directions, and Energy. In this thesis, we will only be using Energy.

The goal of our research is to select variables that give us energy for each atom with high precision.

2.3 The approach of Artificial Neural Network

Deep learning is a subfield of Machine learning. It is also a mathematical framework for learning representations from data which focuses on learning successive layers of increasingly meaningful representations. These layers' representations are learned via models called neural networks.

The term neural networks (NN) is a reference to neurobiology where some concepts of deep learning were developed in part with the inspiration from the brain. In this thesis, we have used the Multi-perceptron layer which is a feed-forward Artificial neural network.

Artificial Neural Networks (ANNs) connects inputs and outputs (mostly known as predictors and predictions respectively) with the help of functions that can be linear or non-linear. Deep learning can do both classification and regression problems. In the regression problems, we can also have multiple outputs.
ANN has weights for every input and a bias element that contributes to the output. These weights are updated as they get trained while ANNs learn from the dataset, just like brain learn from its experiences [3]. With N_{inputs} as number of inputs, and w_i as weight of i^{th} input, we can represent 2.1, a single-layer perceptron network mathematically as

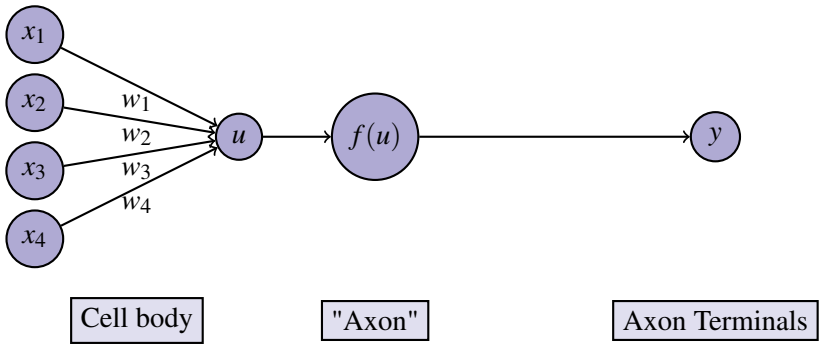


Figure 2.1: Single-layer perceptron network

$$y = f(u) = f\left(\sum_{i=1}^{N_{\text{inputs}}} w_i x_i + b_i\right) \tag{2.2}$$

If a function would be a linear function, we have $u = f(u)$ in (2.2). For simplicity, keeping $x_0 = 1$ so as to accommodate bias terms within weight, we can rewrite equation (2.2) as

$$y = f\left(\sum_{i=0}^{N_{\text{inputs}}} w_i x_i\right) \tag{2.3}$$

2.3.1 Multi-layer perceptrons (MLP)

In a Multi-layer perceptrons network, we have at least three layers of nodes that include a minimum of one hidden layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP uses supervised learning techniques called backpropagation for training the network.

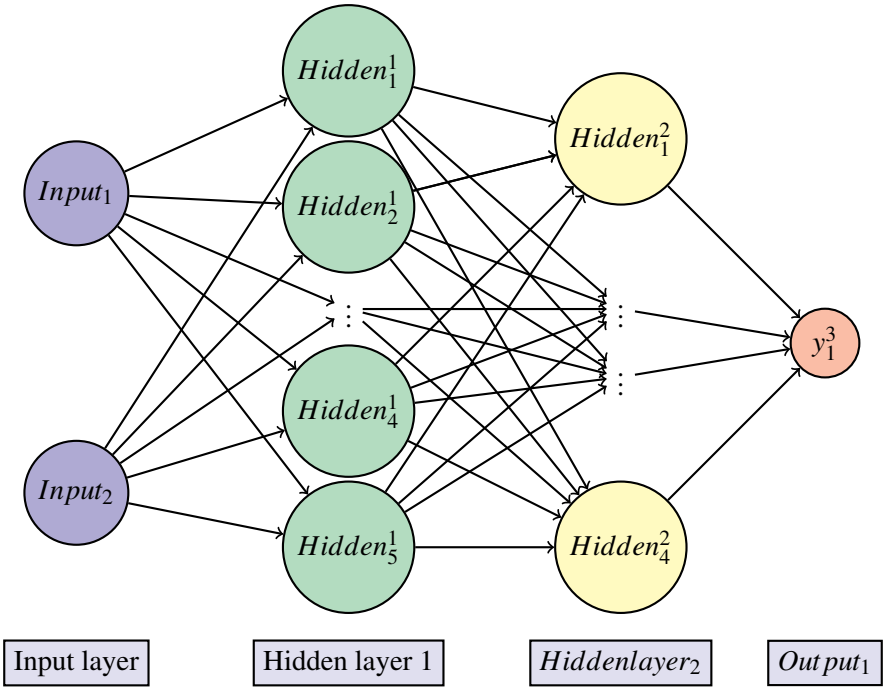


Figure 2.2: Example of Multi-layer perceptron network of structure 2-5-4-1

Figure 2.2 is a type of MLP with 2 input nodes and one output node. They have 2 hidden layers where the first hidden layer has 5 neurons and the second hidden layer has 4 neurons. Each of the nodes is connected by a bridge with a weight and a bias for each node other than a input node.

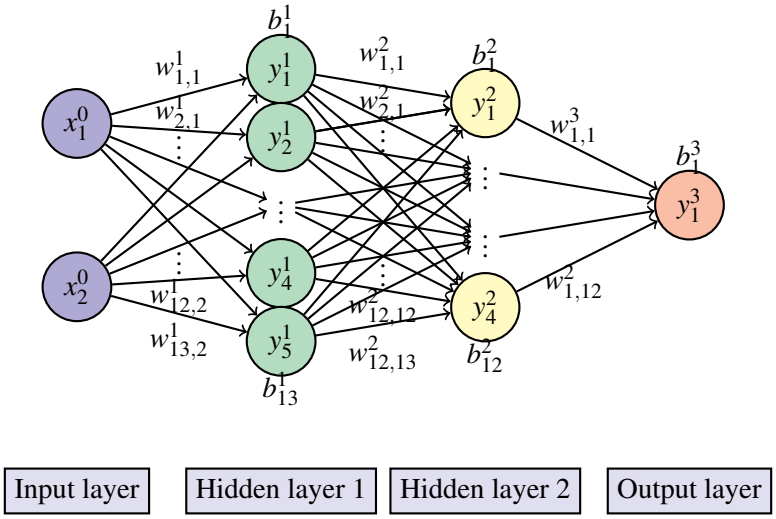


Figure 2.3: Example of Multi-layer perceptron network

If N_l is the size of the l^{th} layer, the generalized equation of MLP Network can be written as

$$y_i^l = f^l(u_i^l) = f^l\left(\sum_{j=1}^{N_{l-1}} w_{ij}^l y_j^{l-1} + b_i^l\right) \tag{2.4}$$

This can be extend to the total energy expression

2.3.2 High Dimensional Neural Network

Most conventional NNPs use a single Multilayer Feed Forward (MLFF) NN to predict atomic configuration and the potential energy, but a large input nodes can make NNP inefficient. There will be not only too many weight parameters but the calculation of output energy will also be slow. An atom added to the system means three new degrees of freedom [2] need to be included as new input nodes.

One of the fundamental problem is the potential energy of the structure can be different even if any two bonds are chemically equivalent. This can be solved by constructing the energy E_s of the system as the sum of E_i,

$$E_{\mathrm{s}} = \sum_{i=1}^{N_{\mathrm{atom}}} E_i.$$

This is where 256 networks predicts Potential with symmetry function mentioned on

2.4

2.4.1 Common terminologies to be used in our neural network

During our training, we first experimented with different structures and the number of hidden layers the find out the combination that gives better precision. Common terminologies that we used to tune our model are described below

Batch size:- It refers to the size of the training dataset used to train at one instance. That one particular instance would count as one epoch. The smaller batch size means training duration for any epoch will be less and time duration increases as batch size increases.

Preconditioning The dataset we have collected is trained in the subatomic NN to get the weights, so as to predict the atomic energies and then potential energy. We precondition the data so that each atomic NN can make an accurate prediction of energy for the given atom.
The preconditioning methods of scaling, shifting, or standardization are used as per the activation function. In our research, we have mostly used function $G_{i,s}^{scale}$ which scales input to $[-1,1]$

$$G_{i,s}^{\mathrm{scale}} = \frac{2\left(G_{i,s} - G_s^{\mathrm{min}}\right)}{G_s^{\mathrm{max}} - G_s^{\mathrm{min}}} - 1 \tag{2.5}$$

Maximum epochs is the maximum number of times a batch of data is trained in the whole training process. If the early stopping callback is satisfied, the training process stops before the maximum epoch.

If **Shuffle of dataset during training** is set as False, it means the dataset will not shuffle during the training and it will train with the same sets of data. This means the model will get

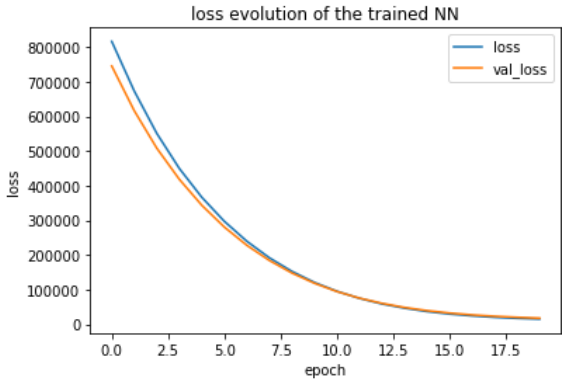


Figure 2.4: Train-Validation Loss graph for Maximum epoch 20

more information from that particular set, but will not collect all information from the overall dataset.

Adaptive learning rate is the process where the learning rate drops with drop_rate after a certain number of an epoch. It will continue till min_learning_rate or till the time early stopping parameter is fulfilled during the training process.

Minimum delta is the minimum change needed in monitored quantity to qualify as an improvement, i.e. an absolute change of less than min_delta will count as no improvement, and training will be stopped.

Patience is the number of epochs after which training will be stopped if there is no improvement.

```
#NN parameters
batch_size = 64; shuffle = True
alpha_l2 = 1e-5 #l2 norm regularization
hidden_layer_sizes = (10,10) #NN hidden layers
max_epochs = 20000 #maximum number of epochs algorithms should train if
    ↪ no early_stopping
activation_function = 'tanh' #layers' activation function

## Adaptative Learning Rate (reducing rule)
learning_rate = 0.01 # initial learning rate
min_learning_rate = 0.0001

rule = 'step' #
# rules implemented
# 'constant' : no reduction

# 'step' : stepwise reduction
drop_rate = 0.50 #0.5
```

```
epochs_drop = 100.0

# 'exp' : exponential
k_exp = 0.0001

## Early_stopping
min_delta = 1e-5 ; patience = 50 #early_stopping parameters

#Which preconditioning do you apply to the data before the training?
#('shift','scale','stand' or 'scale and shift')
preconditioning = 'scale'
```

Activation function

The function that converts a node to a neuron is an activation function. As we said earlier, it can be linear or non-linear. There are various types of activation functions in the Neural network which are applied as per the necessity. In our case, we have tested our network with *tanh*, *sinh*, *Relu*, and *LeakyRelu*.
The choice of activation function should come with the suitable pre-conditioning of input.

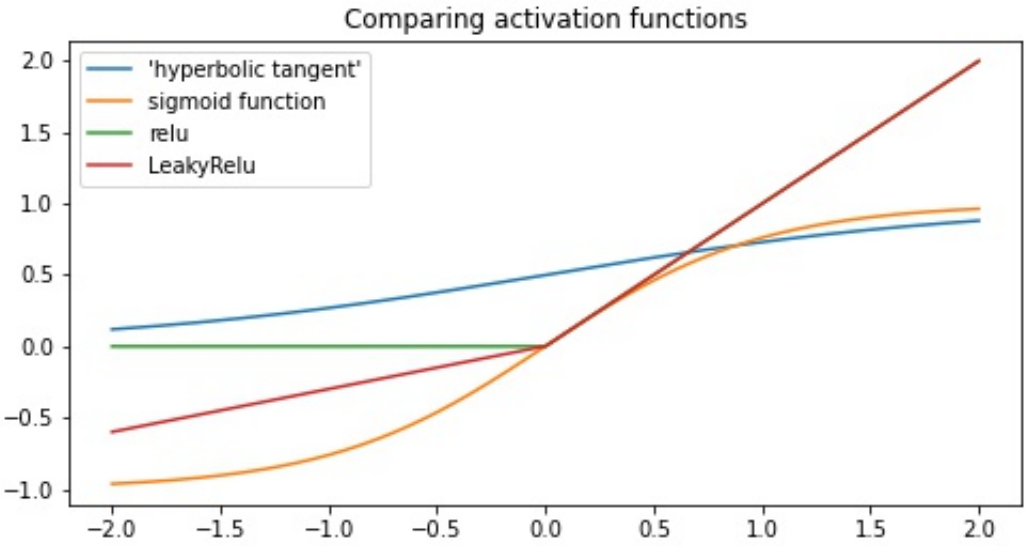


Figure 2.5: Different activation functions plotted individually

The most common activation function tanh, hyperbolic tangent function, projects R on the interval $[-1,1]$ while the logistic function sigmoid projects R onto $[0,1]$.
ReLU (Rectified linear unit) is a cutoff function that maps all negative input to 0.

$$f(x) = max(0,x)$$

.
Unlike, Relu, LeakyRelu doesn't kills the input by mapping to 0 with the function but instead allows the small gradient when the unit is not active.In figure 2.5, $\alpha = 0.3$

$$f(x) = \alpha \times x \text{ if x} < 0, f(x) = x \text{ if x } >= 0$$

,

Cost Function

The goodness of any model is determined by the cost of the model. For every epoch, using input (X), and output (y) of the training set, the Neural Network computes prediction y^* and compares prediction with the real output of the database. The generated loss is associated with a cost function Γ where w denotes the weights. We have used Mean-squared error (MSE) throughout our research.

$$\Gamma \equiv \Gamma(\mathbf{w}, X, y) = \frac{1}{2N} \sum_{i=1}^{N} (y_i - y_i^*)^2 \tag{2.6}$$

This loss is calculated for both training and validation sets. For each training epoch, the gradient descent algorithm computes new weight w_{ij}^l

Gradient descent algorithm

It is an iterative optimization algorithm that helps in the minimization of the cost function. The repeated steps taken in the opposite direction of the gradient help update weights for the coming iteration so that cost function (Γ) can be minimized with the help of step size (η) also called as the learning rate. The update of weight can be mathematically written as

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \nabla_{\mathbf{w}_k} \Gamma(\mathbf{w}) \tag{2.7}$$

Adam Optimization algorithm

Adam is an optimization algorithm that is used to update network weights iterative based on training data. In the paper, the author [4] describes Adam as a method to compute individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. It realizes the benefits of both Adaptive Gradient Algorithm (AdaGrad) and Root mean Square Propagation (RMSProp).

Parameters that configures Adam are:

- η the learning rate
- β_1 (default value of 0.9) the exponential decay of the first moment estimation;
- β_2 (default value of 0.999) the exponential decay of the second moment estimation, which should be close to 1 to avoid issues with a sparse gradient;
- ε a very small number to prevent any division by zero $\left(\sim 10^{-8}\right)$

If we denote the gradient $g_t = \nabla_{\mathbf{w}_t}\Gamma(\mathbf{w})$, the Adam optimizer is iteratively computed as

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\, g_t$$
$$v_t = \beta_2 v_{t_1} + (1-\beta_2)\, g_t^2$$

where m_t and v_t are estimates of the first and second moment (resp. mean and uncentered variance) of the gradient.
If m_t and v_t are initialized as zero, it create biases. There is where Adam uses the first and second moment estimates:

$$\hat{m}_t = \frac{m_t}{1-\beta_1}; \quad \hat{v}_t = \frac{v_t}{1-\beta_2}$$

Finally, the weights are updated according to the Adam update rule as follow:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t$$

```
optimizer_adam = optimizers.Adam(learning_rate=user.learning_rate,
    ↪ beta_1=0.9, beta_2=0.999, epsilon=1e-08, amsgrad=False)
```

2.5 The methodology of training NN

We train our high dimensional neural network regressions problem with Python Keras and scikit-learn modules. The neural network was build during the simulation of LJ [7] and we replicate the same code with slight modification.

The whole neural network is written in five different python scripts.

- **run_NNP.py**: We call this script to run the Neural network. This script loads all the necessary files and calls the needed function in the following order.
  - Step 1: Generate the dataset
  - Step 2: Build, Compile, train the model
  - Step 3: Check the prediction and the actual energy
- **user_NNP.py** :- This file helps to collect values of different hyperparameters so that we don't need to change the other scripts to change the structure and format of our model.
- **data_preprocessing.py**:- This script transform from 2 dimensional to 3 dimensional. This script loads the .csv file (input data) does all preprocessing including scaling.
- **model_building.py** It contains the function to build the neural network. Here we build 256 different neural networks and add layers to our model. This compiles and trains the model by updating the learning rate and weights in every epoch. If early stopping criteria is fulfilled before the completion of *nth* epoch, our training stops and does the final prediction with testing data.
- **HDNNP.py** This python script file has a function to predict the test split of the neural network.

For simplicity, we denote a structure of neural network with a number of features, a number of dense features, and number of output, each separated by a hyphen (-). While the first and last term will mean number of input and output variables, remaining other term will show number of neural networks.

For example, 22-10-10-1 means the neural network has 22 features, one output, and two inner layers with depth (10,10). 57-20-100-20-1 means the neural network has 57 features, one output, and three inner layers of depth (20,100,20).

Train test split

As explained in Section 2.2, each observation has 256 different configurations. From the generated data, a model is built, compiled, and trained. The train-test-split function of Scikit-learn splits is an easy tool to split train and validation set on the ratio of 80:20.

```
train, test = train_test_split(configurations, test_size=0.2,
    ↪ random_state=101)
```

Build the model

Since we would like to see the same testing environment throughout, we fixed $random\_state = 101$ to split our data. To use preconditions for our data, we have

```
train_X,G_parameters = data_precondition(train_X,N_atoms,X_label,
    ↪ preconditioning=user.preconditioning)
test_X = data_precondition(test_X,N_atoms,X_label,preconditioning=user.
    ↪ preconditioning,G_parameters=G_parameters)[0]
```

For every epoch during training and testing, there will be a shuffle of data from the training set so as we get to train with different data at every epoch. This is how we build the layers for two dense layers.

```
for i in range(0,N_atoms) :
#creating the layers
subnet_in = Input(shape=(N_features,))
subnet_h1 = layer_h1(subnet_in)#subnet_h1 = Dense(hidden_layer_sizes[0],
    ↪ input_dim = (N_features,),activation = activation_function, name='
    ↪ atom'+str(i)+'h1')(subnet_in)
subnet_h2 = layer_h2(subnet_h1)#subnet_h2 = Dense(hidden_layer_sizes[1],
    ↪ input_dim = (N_features,hidden_layer_sizes[0]),activation =
    ↪ activation_function,name='atom'+str(i)+'h2')(subnet_h1)
subnet_out = layer_out(subnet_h2)
```

In the above python code, $N\_features$ is the total number of features (22 in the above caseC.4). All 256 subnetworks with their respective inputs and outputs share the same dense layers. The compilation is done with optimizer adam and mean squared error as a loss.

```
sub_networks_in.append(subnet_in)
sub_networks_out.append(subnet_out)
```

```
out = add(sub_networks_out)
#out = Dense(1,name='output')(concatenated)
merged_model = Model(sub_networks_in,out)
```

Compile the model

To compile the model we have build, we use

```
merged_model.compile(optimizer=optimizer_adam,loss='mean_squared_error',
    ↪ metrics=['mean_squared_error'])

#save the model
merged_model.save(model_label)
```

Fit the model

This saved model is loaded on the TensorFlow Keras model.

```
model = tensorflow.keras.models.load_model(model_label)
#early stopping parameters
early_stopping = EarlyStopping(monitor='val_mean_squared_error',
    ↪ min_delta=user.min_delta,patience=user.patience, mode='min')


#fit the model
fit = model.fit(train_X,train_y,validation_split=0.2,batch_size=user.
    ↪ batch_size, shuffle=user.shuffle, epochs=user.max_epochs,verbose=2,
    ↪ callbacks=callbacks)
```

Our Keras model looks like in the figure C.4

The left side of the plot of fig 2.7 is a loss graph of the above structure with a depth of (10,10) validation. To see the real fluctuations of data in the lower level, we trim the first few rows preserving their index so as to get the plot at the right side of the fig 2.7

Predict the model

This new prediction can be plotted to see the final performance of the Keras model in the figure 2.8.

```
#load the model
model = tensorflow.keras.models.load_model(model_label)

if type(test) == str :
    test = data_preprocess(test,N_atoms,test_data=False)[0]

#separate inputs and outputs
test_X = test[0] ; test_y = test[1]
```

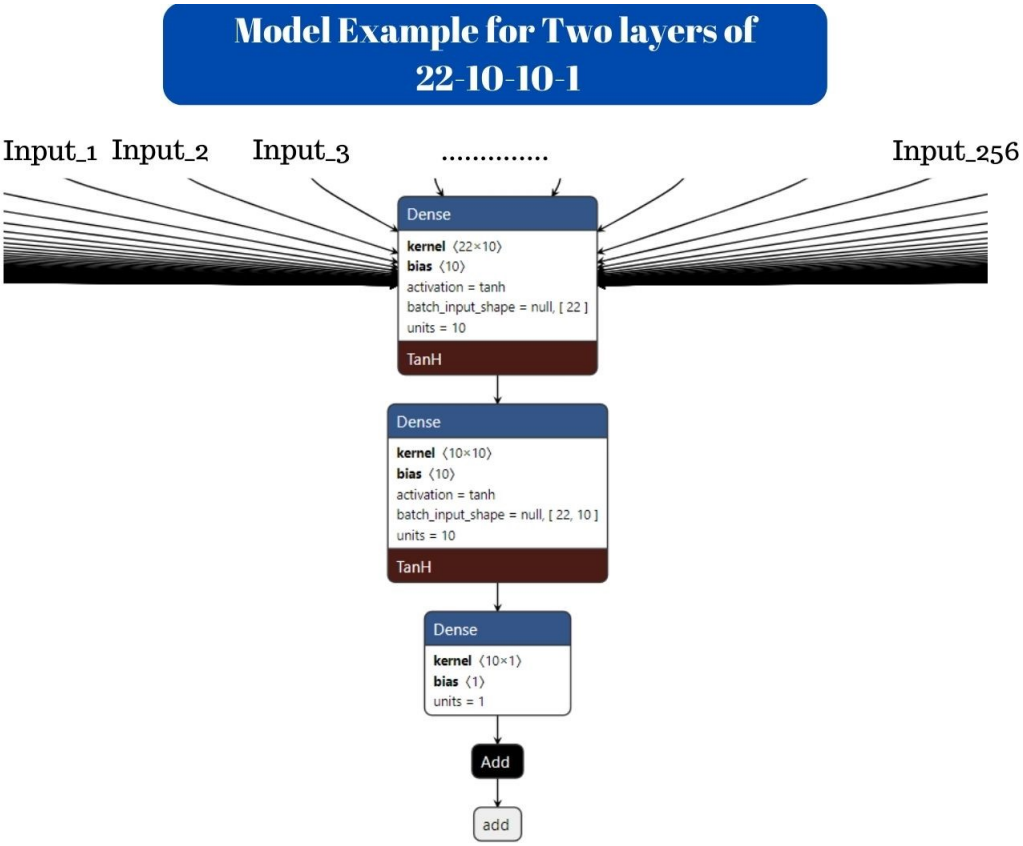


Figure 2.6: Example of ANN Model of 22-10-10-1

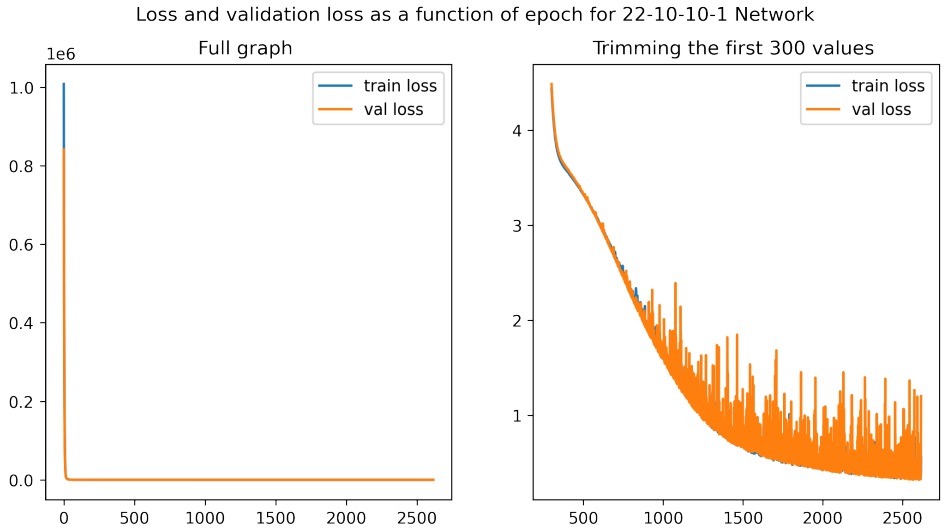


Figure 2.7: Train loss full data sample

```
#get the predictions
predict_y = model(test_X)
```

```
test_y = test_y[:,0::N_atoms,:][:,0]
```

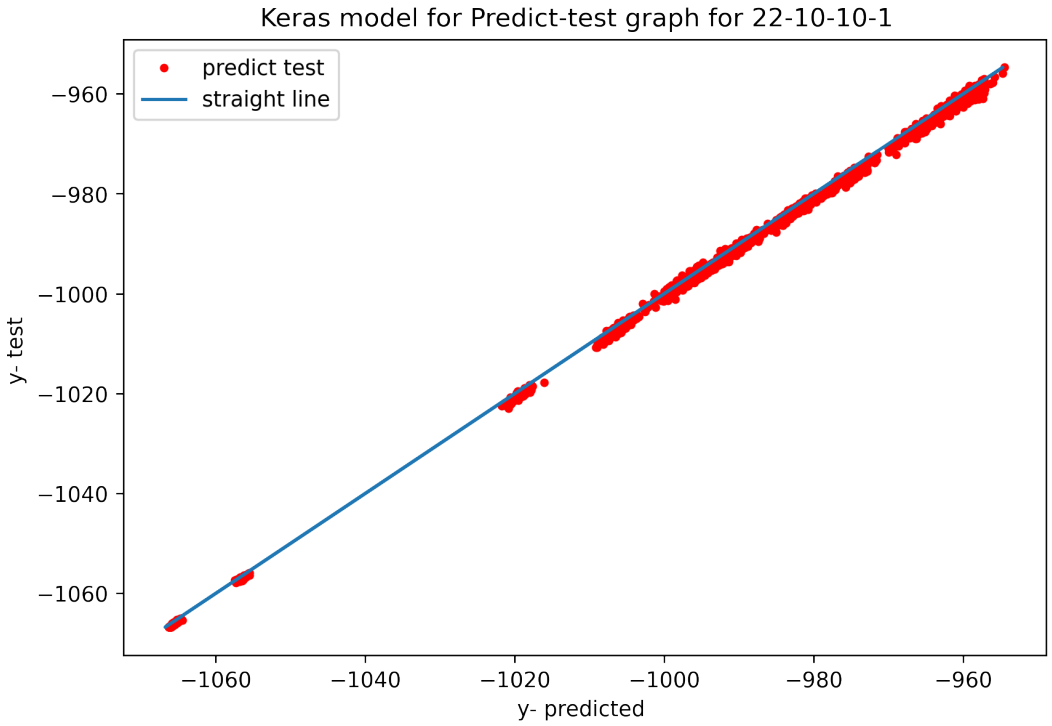


Figure 2.8: Plotting of Test-Predict while predicting test set

In the next chapter, we will use this model to train our initial dataset and we shall also talk about our extended dataset.

— 3 —
Methods of Simulations

In this chapter we discuss about our method of training and simulating our Neural Network. With methods of creation of dataset described on the previous chapter, this chapter explains dataset of 22 variables. We interpret them with respect to parameters of Behler Parrinello methods. We train and compare different ANN models to get the best structure to train further models. We retain that structure and extend our dataset. We aim to search for a suitable variable structure that can predict energy with highest precision. To get a higher precision, we reducing our dataset by dropping number of variable so as to work with Behler features that gives us optimum loss.

We also compare our performance of HDNNP with random forest and state findings briefly in this chapter.

3.1 Training of the NN with 22 features

The early result was provided by Anthony Saliou in his Master thesis paper [7]. In his thesis, he used HDNNP to mostly work with Lennard Jones (LJ), and in the final part he build a potential on ab initio data for the Aluminium (Al). He used 22 Behler features : (12 radial symmetry functions G^2 and 10 angular symmetry functions G^5. Our final aim is to get higher accuracy in energy predictions as far as possible

We started with 22 variables that consists of 12 $G2$ and 10 $G5$ features. These early datasets would give us a brief view of Behler parrinello features for Aluminium.
We varied the depth of both inner layers from 5 till 50 and trained the model for 12 $G2$ variables. The graphical representation of gave us following observations in the table 3.1. The physical representation of the $G2$ variables for 12 variables can be seen in figure 3.2 where $R_C = 6.8$, $\eta \in [0, 1.85]$ and $R_s \in [0, 6.105]$
The initial simulation gave us few possible insights which we shall look in coming sections.

- Time for an epoch for smaller depth (<=20) is less as compared to structure with depth (>20)
- As seen in the figure 3.1 the structure with smaller depth gave a better precision of energy prediction than the one with the higher depth.

The figure represents the minimum loss per atom with 12 symmetry radial features .

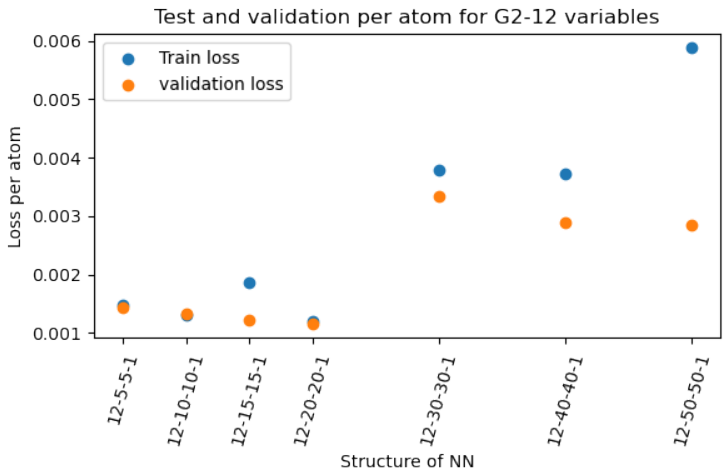


Figure 3.1: Loss per atom with 12 radial (G2) features

time for an epoch	NN Structure	For whole Configuration		For One atom		Epoch	
		train_loss	val_loss	train_loss	val_loss	Max	minima
4s	12-5-5-1	0.3788	0.3687	0.001480	0.001440	2886	2835
8s	12-10-10-1	0.3361	0.3403	0.001313	0.001329	2787	2736
7s	12-15-15-1	0.4757	0.3149	0.001858	0.001230	2120	2069
4s	12-20-20-1	0.3052	0.2964	0.001192	0.001158	2415	2364
20s	12-30-30-1	0.9721	0.8571	0.003797	0.003348	1174	1123
17s	12-40-40-1	0.9548	0.7397	0.003730	0.002889	1222	1171
20s	12-50-50-1	1.5048	0.7291	0.005878	0.002848	1243	1192

Table 3.1: Train and validation Loss for different structure with 12 features .

We also simulated combining the entire 22 dataset that had both radial and angular features. We experimented to see what combinations of depth- validation split- number of features can give results with higher precision.

The least loss across the various depth were fluctuating, however, along with the simulation, structures with smaller depth had higher precision. These loss values generated from such simulations were insightful to proceed with further simulation.

This inference can further be boosted by the box plot in fig 3.4 which shows loss data of last 50 epochs. While models with smaller depth structure had small variations, models with larger depth has larger variations. This boxplot can further tell us the possibility of bigger noise. Let us compare by looking at the exact train-test loss graph of two different simulation.

Since we would mostly focus on the performance of validation loss, training more models with smaller number of depth would give us new insights.

We also looked what type of train-test split would make a better fit. So instead of keeping validation split as 0.8, we changed the data to see if other split would give us better result. For this, we trained on structures of (5,5), (10,10) and (15,15). In the figure 3.5 we have trained models with validation split 0.8, 0.7 and 0.6.

Looking at the overall loss, we choose our validation split as 0.8 for further training.

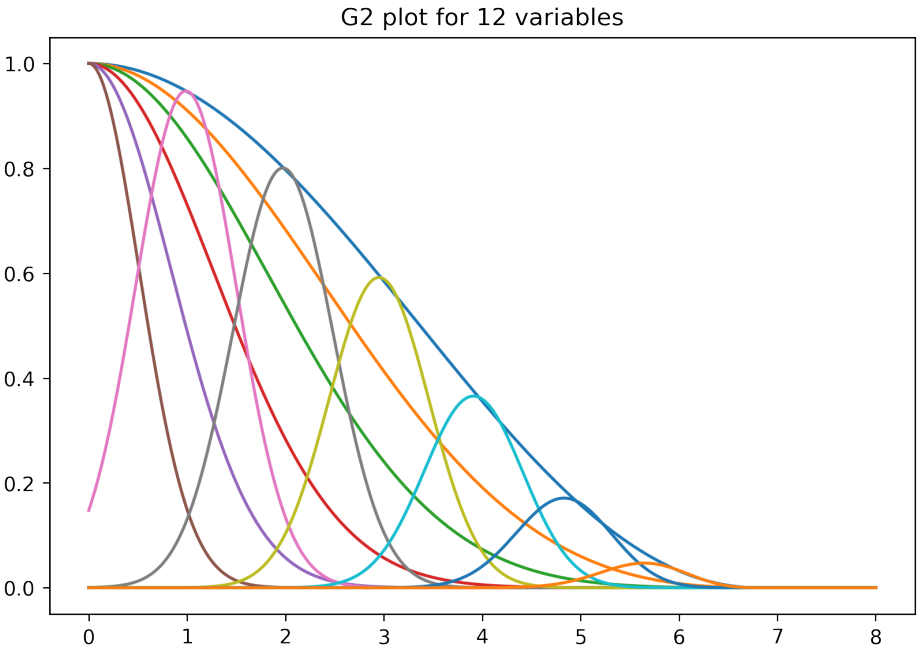


Figure 3.2: Physical representation of G2 for 12 variables

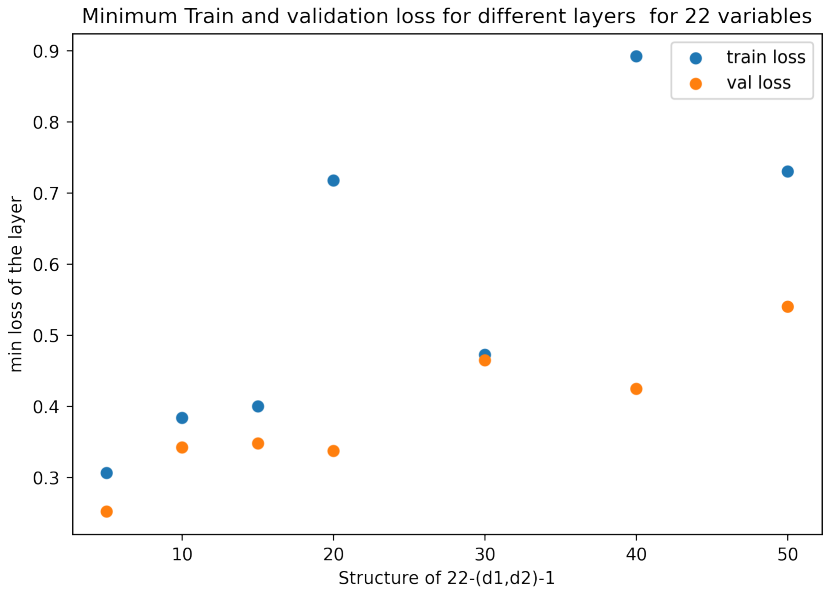


Figure 3.3: Training loss and validation loss for different layers

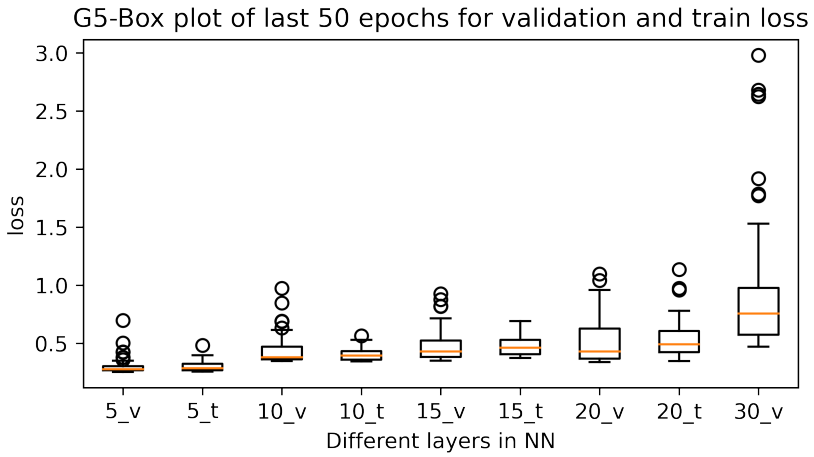


Figure 3.4: Box plot of training loss and validation loss for different layers

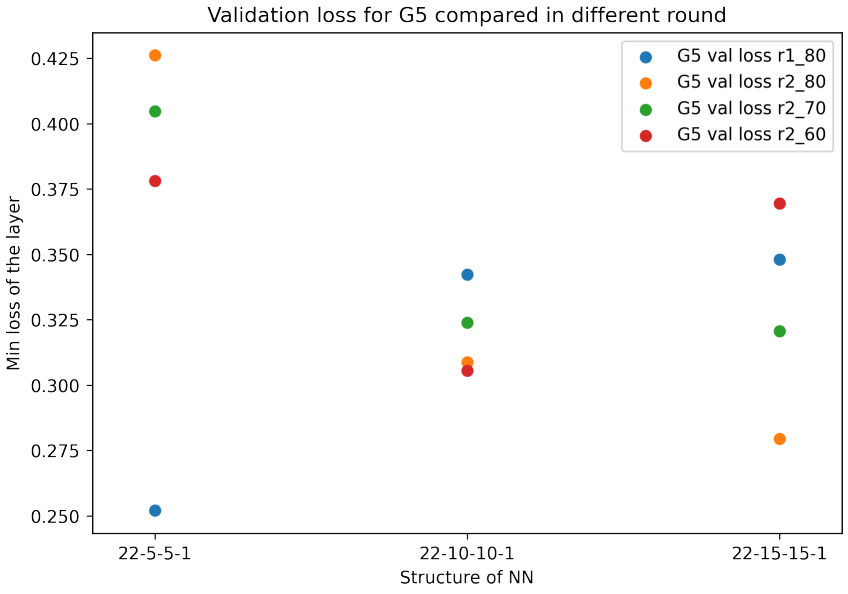


Figure 3.5: Validation loss for different structures with different sets of validation split

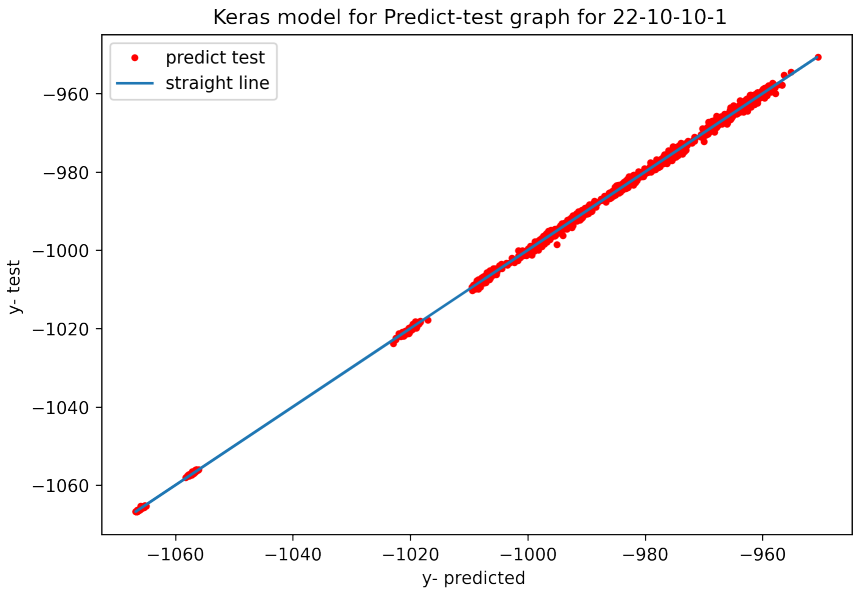


Figure 3.6: keras model test-predict line

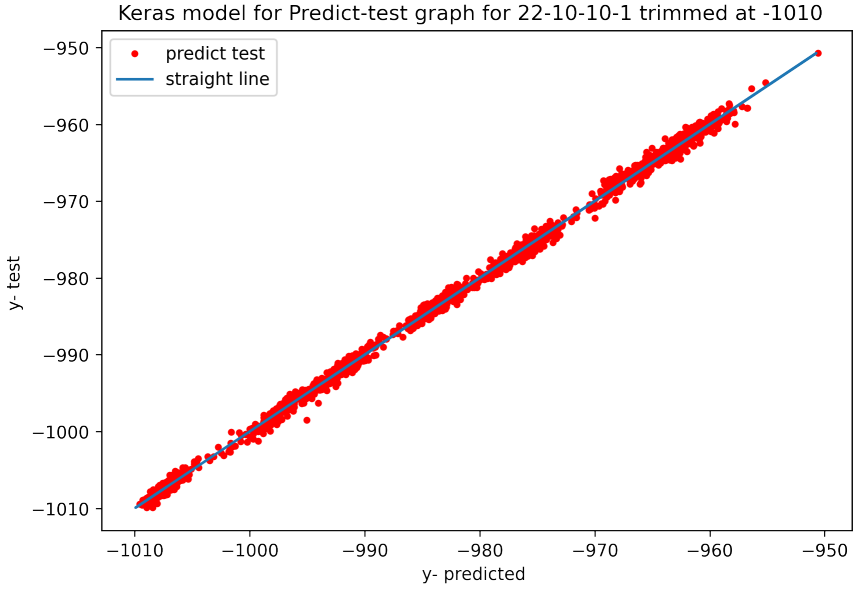


Figure 3.7: A closer look after cropping

3.2 Selection of depth, structure and hyperparameters

From the various simulations we had in our training set, we came with the conclusion that following structure gave a better precision of energy prediction.

So, in the upcoming simulation, we will mostly be sticking with following structure for our Artificial Neural Network.

- **Train Validation split** :- 0.8
- **Number of layers** :- 2
- **Depth of inner layer** : - $(10, 10)$
- **Activation Function**:- *tanh*

3.3 Extending dataset to 333 varaiables

After some rounds of expansions we finalized with 329 variables (269 G2 variables, 60 G5 variables).

3.3.1 Extending G2 variables from to 269

This is how we generated 269 G2 features for our final data. By adding 3 directional component of Forces and One Energy, we have total 333 variables in our feature.

```
#format r_c,eta,r_s
x = np.linspace(0.0,2.0,201)
x1 = np.ones(68)*1.85
e1 = np.linspace(0.05,6.8,68)
e = np.zeros(201)
r = np.ones(269)*6.8
x = np.append(x,x1)
e = np.append(e,e1)
x[0] = 0.0013
features_G2_=[]
for i in range (269):
    features_G2_.append([r[i],x[i],e[i]])
```

The symmetrical radial features (G2) can be classified into two group, one with $e = 0$ and other with $e \neq 0$. We have shown them on the graph.

3.3.2 Extending G5 variables to 60

```
rc = np.ones(10)*4.4
eta = [1,1,2,2,4,4,16,16,64,64]
lambd = [1,-1,1,-1,1,-1,1,-1,1,-1]
x = np.linspace(0.0,2.0,6)
x[0] = 0.0013
features_G5_ = []
for i in range(len(x)):
    for j in range(10):
        features_G5_.append([rc[j],x[i],eta[j],lambd[j]])
```

These datasets are build for the following temperature. There were 5 files for each temperature and each file had 200 obs. As earlier, each observation had input data of 256 configuration.

```
['1000K', '10K', '1100K', '1250K', '1350K', '1500K', '1500KH', '1600K', '
    ↪ 1700K', '300K', '400KS', '500K', '500KS', '600K', '600KS', '650K',
    ↪ '700K', '700KS', '750K', '800K', '800KS', '950K']
```

Because of some outliers in the data, we removed the temperature 1000K from our database. Further investigation showed us that 200 observation (One file) from each all 5 files would give us a result. This would save computation time and memory. Hence, we proceed with the dataset with 4200 observations.

	Extended Data	After Rivision
Total Types of Temperature	22	21
Total File For each temperature	5	1
Total Files Included	200	200
Total Observations	22000	4200

Table 3.2: Description of data

To check if the past data of 22 variable is representation of new data with 329 variables (say v_329) We decided to train the new data with model with 22 variables . These features would be of the closest representation that we have for 22 variables.

The plot of physical representation in 12 G2-features in 3.8 of both of the case would closely match with each other. We can see the red line (as old features) and green line new_v3 as extracted from 269 elements.

We trained the extracted dataset with two layers and inner depth of (10,10). This training was done with G2 features and G2+G5 features. The simulation tells us that extracted data clearly gives us better result both in terms of radial variables and radial+angular variables.

Simulation	Train_loss	Val_loss	Max_epoch	minima epoch
12-10-10-1	0.2342	0.2562	829	779
22-10-10-1	0.2225	0.2402	2469	2419

Table 3.3: Train and validation Loss for extracted data

We can compare the loss value of first data (v22) vs extracted part from full data.

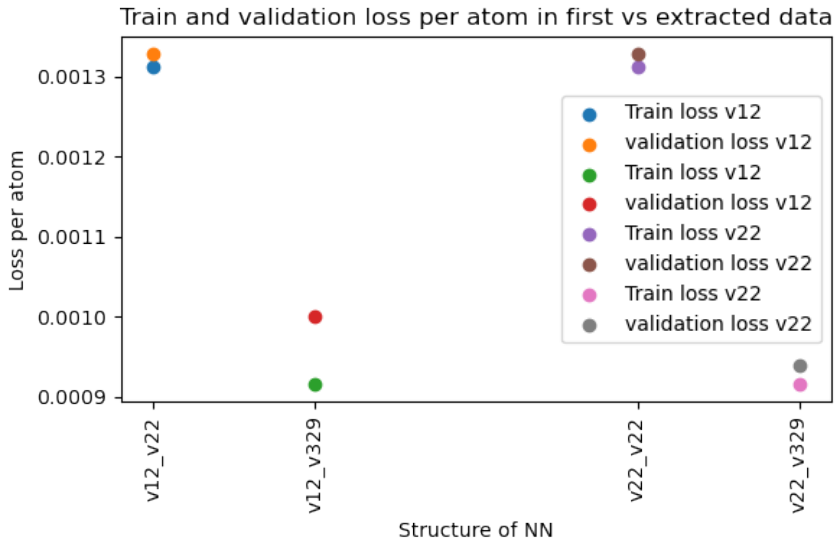


Figure 3.8: Comparing least loss of 12 and 22 variables in the case of old data and extracted data

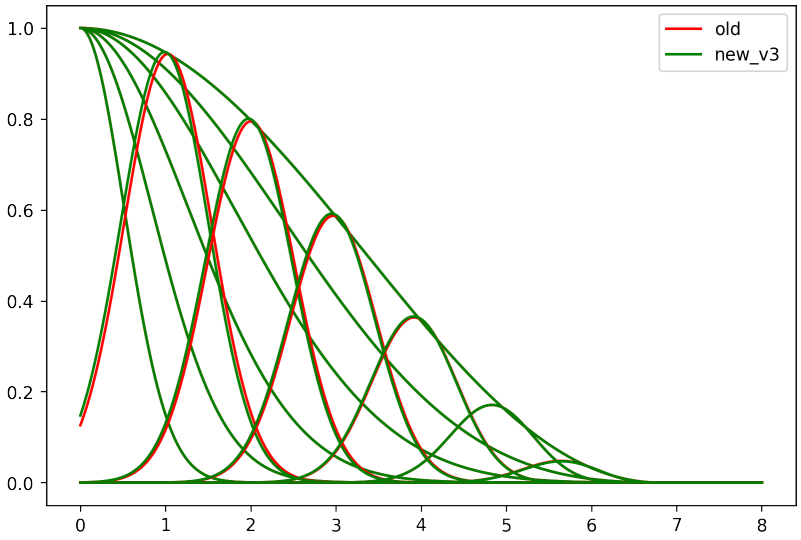


Figure 3.9: Matching G2 feature of the both case

3.4 Random Forest Regressor

While we wanted to try our methods with different machine learning models other than a neural networks, we gave a try with Random Forest as it is a great fit for high dimensional data. Thanks to its versatility, Random Forest can solve both regression and classification. It is an ensemble of decision tree algorithms. It means it uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees.

Random forest, with its bootstrapping techniques may not use the whole dataset to build each tree. It uses an average to improve the predictive accuracy and control over-fitting.

Apart from all these, the random forest would make variable selection easy process with scikit learn features which could reduce the computational cost of any machine learning model.

After some simulation, while modeling different versions of random forests, I realized some difficulty wouldn't make the random forests a great fit.

- Our NN had 256 sub-models which meant it isn't normal deep learning. Running an entire data set with one random forest model means we consider all 256 environments. This wouldn't be a good approach

- If we could reshape n_features * 256 input in one row, we would have 256 outputs on Random forest regressor. While multiple-output for the random forest is possible, but in this sense, multiple outputs would be dependent on the environment of 256 atomic configurations. This would be a wrong approach

- Building 256 different Random Forest Model would mean a hectic process. 256 different models would also mean the same features suppose (G2-165) can have different weights over each model.

We simulated with various parameters where our mse_loss value from the Random Forest regressor was much higher (see Annex for Random Forest Regression) from the loss values we got from Random Forest. We also worked to predict the forces where our model couldn't predict with the correct precision.

While random forest would predict a mean for every similar input discriminating the regions that have the same output, our atomic energy in the data is in the form of Gaussian distribution with a very small variance. As the estimator is the mean over the observations in the region, the prediction would always be more or less the mean of the energy. This means it would give predictions with no or least precision.

To know problems at a smaller level; we simulated just for a 950K temperature. While mse for 20 as max_depth reached around 2.0, predict-test graph would say a different story.

The purpose of keeping the random forest part despite the model heavily failed here is to make a reference for the future so as a researcher would refer from the model and results we have trained during this research.

While Random Forest Regressor couldn't give a better precision, Artificial Neural Network that we predicted is giving us prediction with a higher precision level.

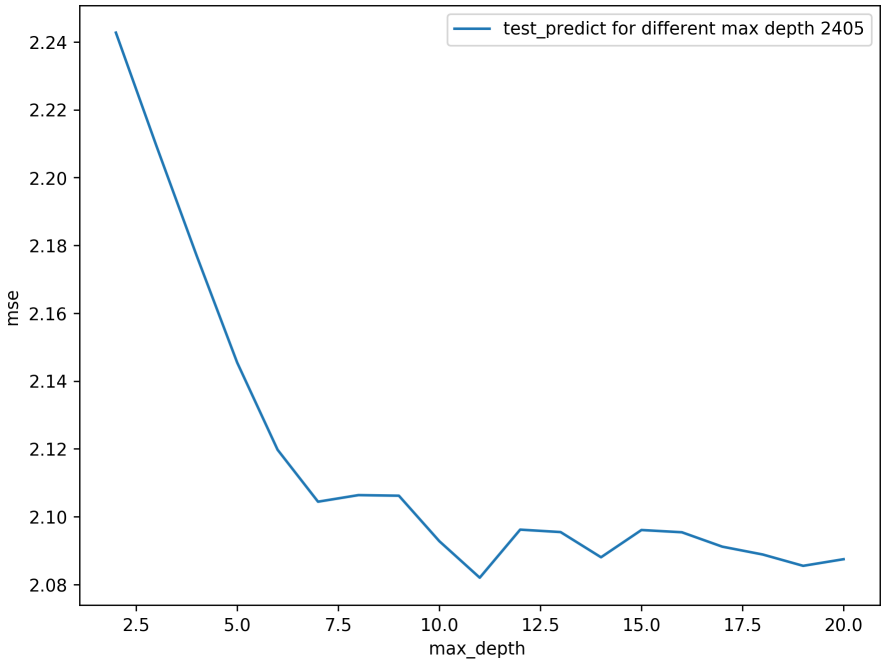


Figure 3.10: RF MSE Error for 22 variables for 950K

Since the expansion of data from 22 variables to 329 variables seems to be working well, we will move back to simulate with ANN with 329 variables.

In the next session, we will also make an attempt to select only relevant variables which is the core part of our research.

Few more plots including partial tree plot of RF model is shown in Appendix B.

— 4 —

Selection of Variables through Artificial Neural Network

In previous chapter, we created data with 329 variables which consists of 269 G2 (symmetrical radial Behler features) and 60 G5 (symmetrical angular Behler features). We will dive deeper into the group of variables so as to drop some certain or group of variables.

Variable (Feature) selection in deep learning is an emerging field where lots of research are going through out the world.

One of the most common approach of variable selection is Sequential Backward Elimination (SBS) [6] where full set of features are trained at the beginning. The next step would be to drop each variable one at a time and see their overall effect. If the least loss decreases with removal of the feature, we remove the particular feature, if it increases we keep back feature. This step continues for all variable. However, removing variables one by one can go undetectable because of the stochastic effect of the batch.

4.1 Structure of 329 variables

We look at the set of 329 input variables where our G2 variables can be divided on the basis of their physical representation in figures 4.1 and 4.2

When trained NN models separately according to symmetry functions, the least validation loss of angular symmetrical features were 1.3418. It took 2204 epochs to get this value. This cost is much much higher than the individual contribution from G2 with η equals to 0 and η not equal to 0 trained separately.

The figure 4.3 tells us that G2 parameters with non-zero η (the final 68 features of 269 G2 features) can contributed more information than features that have zero η. However, features that has zero η is 201 in number compared to 68 of G2 parameters with non-zero η

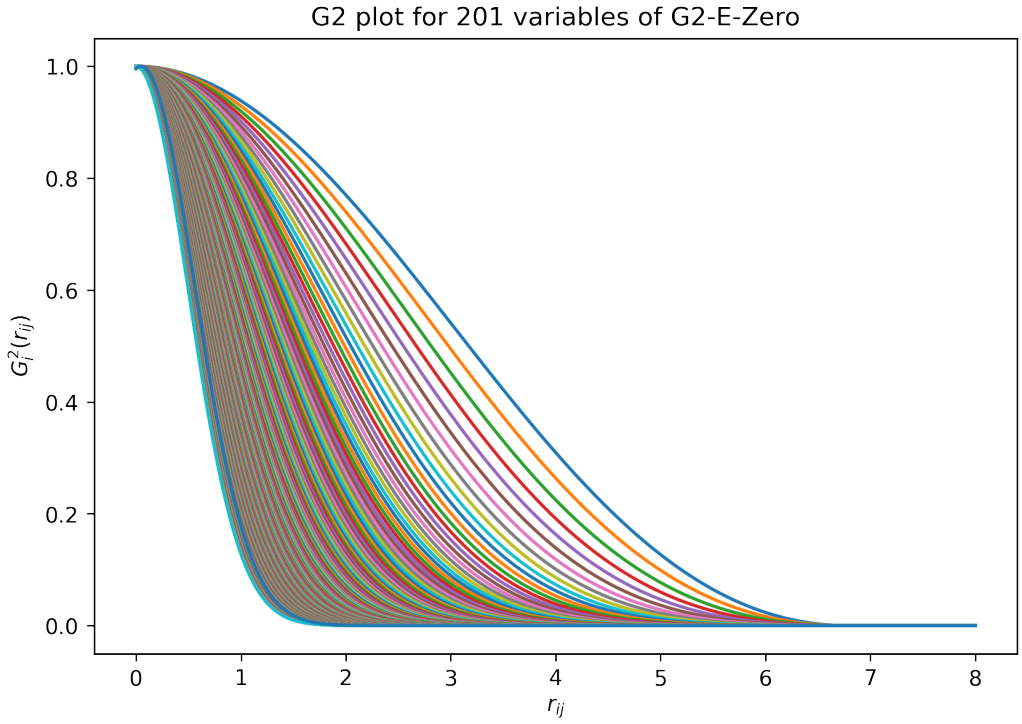
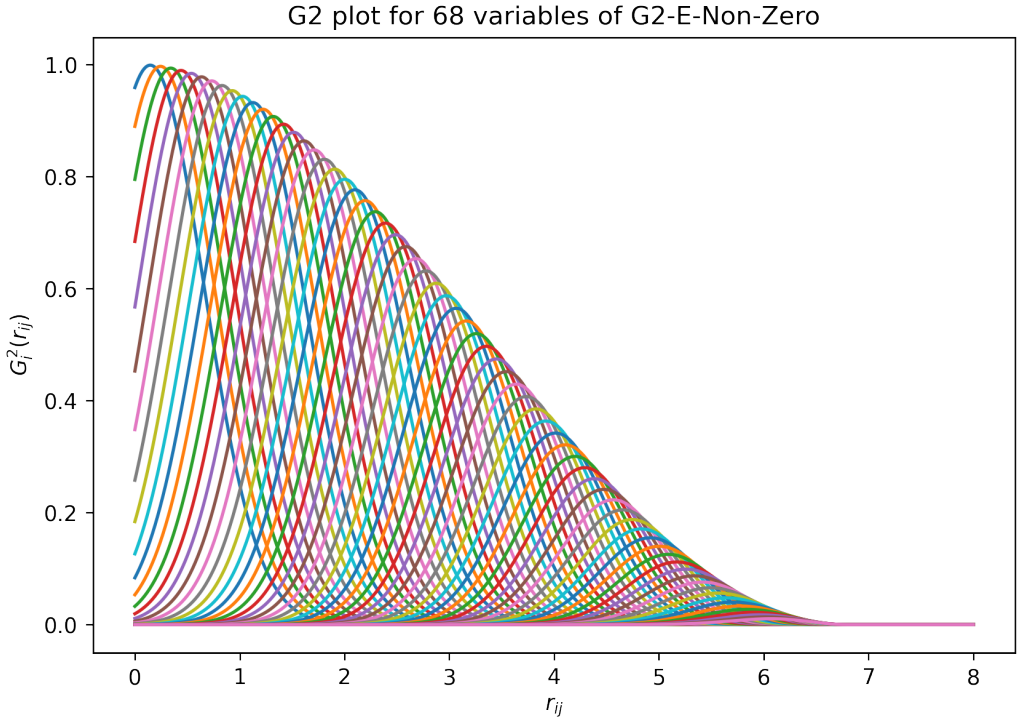


Figure 4.1: 201 variables of G2-E-Zero



htb!

Figure 4.2: 68 variables of G2-E-Non-Zero

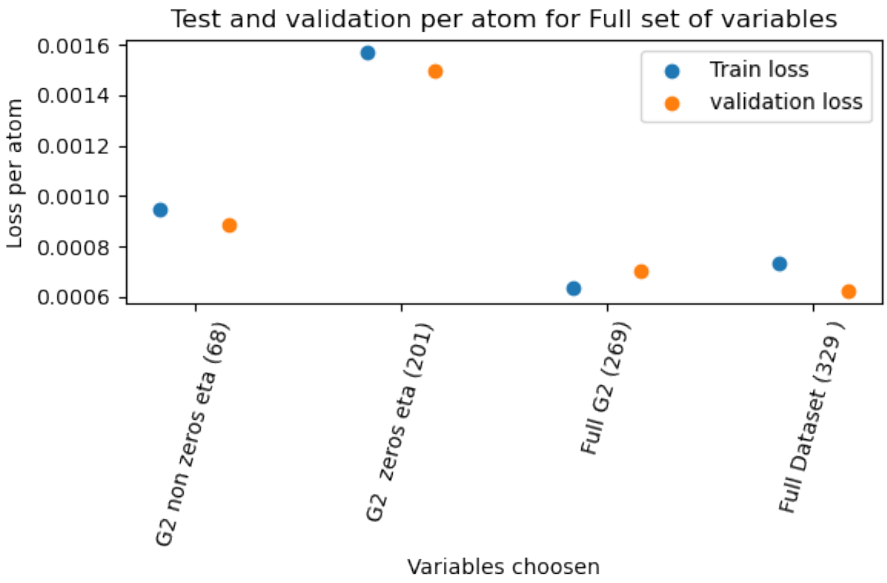


Figure 4.3: Training with variables defined by η

4.2 Modeling with other combinations of variables

To check the precision of various variables, we created a line-space of the certain parameters as shown in the code to randomly select even spaced variables.

In this training process, we trained more than 200+ models, its total number of variable from 1 to 329, most of the models had 50 to 100 variables. The details are shown in table 4.1.

While the least validation loss with models of 1 variable, 2 variables and 4 variables were (0.084614 , 0.041685, 0.01625, 21.661367 , 10.671511, 4.160973, rest of the models had validation loss below 1.0. In the figure 4.4, to focus on loss of nearby variables, we leave first three models and vizualize remaining of them.

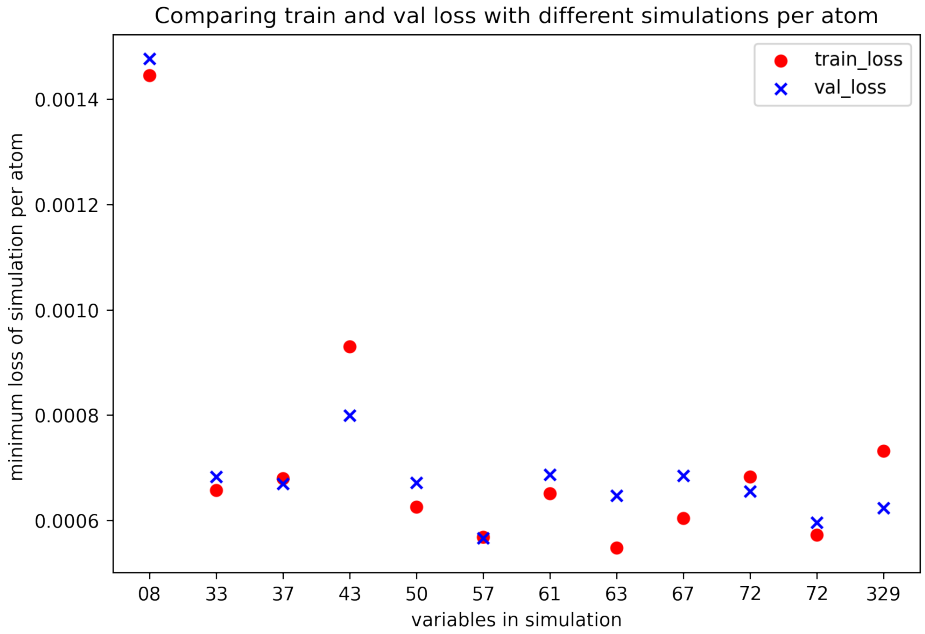


Figure 4.4: Training with various number of variables

Figure 4.4 shows that the model with 57 variables is the best performing model among those models selected for the training. Model 57 has the least validation loss of 0.14478.

However, there is a problem. A big concern that if we again train another model in the same environment with same structure and variables, the previously achieved loss may not be achieved.

To have a deeper insights, additional 57 models were created, dropping a variable from the set of 57 variables one each at a time. The new model would have 56 features. The least validation loss of each cases are given in the table D.1.

In the table D.1, a model where G2(feature41) was removed has the least loss of 0.14248. In fact this loss is even smaller than the loss for 57 variables, but training them once again didn't give us the lowest loss as we achieved before.

A model with 53 variables (by removing G2(feature4), G2(feature33), G2(feature97), G2(feature185) was trained. Its least validation loss was 0.17716, larger than the models with 56 variables.

Simulation	train_loss	val_loss	val_loss_atom	train_loss_atom	Max_epoch	minima epoch
1-10-10-1	23.79942	21.66137	0.084615	0.092966	377	327
2-10-10-1	10.69884	10.67151	0.041686	0.041792	1872	1822
4-10-10-1	4.29620	4.16097	0.016254	0.016782	3139	3089
8-10-10-1	0.37002	0.37806	0.001477	0.001445	1794	1744
33-10-10-1	0.16829	0.17480	0.000683	0.000657	1053	1003
37-10-10-1	0.17391	0.17132	0.000669	0.000679	1372	1322
43-10-10-1	0.23797	0.20454	0.000799	0.000930	1470	1420
50-10-10-1	0.15996	0.17176	0.000671	0.000625	2222	2172
57-10-10-1	0.14539	0.14478	0.000566	0.000568	1046	996
61-10-10-1	0.16667	0.17574	0.000686	0.000651	1509	1459
63-10-10-1	0.14009	0.16553	0.000647	0.000547	1517	1467
67-10-10-1	0.15445	0.17527	0.000685	0.000603	1005	955
72-10-10-1	0.17484	0.16777	0.000655	0.000683	1577	1527
72-10-10-1	0.14639	0.15255	0.000596	0.000572	1498	1448
329-10-10-1	0.18733	0.15951	0.000623	0.000732	2199	2149

Table 4.1: the least Loss per atom of all variables

With data from table D.1 this tells us data treating HDNNP as normal neural network might not be the suitable way.

4.3 Further digging to 57 variables

Those 57 variables are not random 57 variables but they were generated from 329 variables with equal spacing between them. So, without disturbing the combination of G2-G5 variables, we shift variables by one step so as to form a group. It is done as follows. To generate variables of SetB, SetB is uncommented and other sets A, C, D are commented

```
#Randomly selecting even spaced variables
a,b,c = 8,4,4 #for 57 variables
x,y,z= 1,202,1 #set A
#x,y,z= 2,203,2 #set B
#x,y,z = 3,204,3 #set C
#x,y,z = 4,205,4 #set D
g2_4_1 = np.array(range(x,201,a))
g2_4_2 = np.array(range(y,270,b))
g2_4 = np.concatenate((g2_4_1, g2_4_2), axis= None)
g5_4 = np.array(range(z,61,c))
```

Through the above, part D gives a validation loss of 0.150827 per configuration (0.00059 per atom).

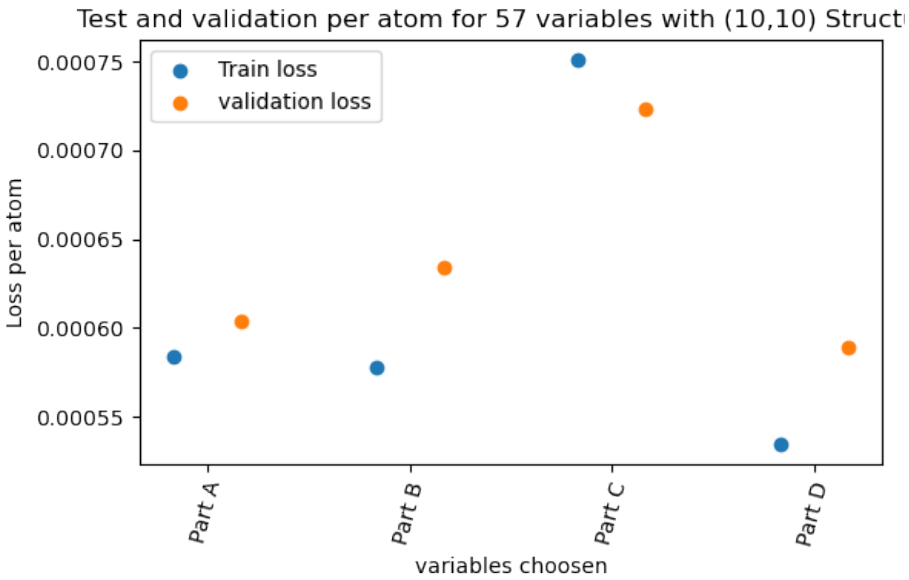


Figure 4.5: Training with various groups of 57 variables

4.4 Dropping variables with respect to weights

There are also research papers that talk much about variable selection through the analysis of weights. One of the variable selection methods uses the Greedy elimination algorithm in the penalized neural network [8]. In this approach, variables are dropped or retained with respect to the change of weight tied to the variable.
This can look interesting, but it is well noted that with the shuffling of data during the epoch, the weight of a variable on two different simulations under a similar environment can be different. As said in section 1, because our ANNs having sub-networks in a model, it brings difficulty in deciding variables with respect to weight.

During this thesis, I dropped/added a variable with respect to their weights, but they were inconsistent throughout multiple simulations and there wasn't a clear conclusion drawn despite numerous attempts.

4.5 Significance test of various Models with the model of 57 variables

The significance test is used to check if two samples are drawn from the same population or not. We check the validation loss of various models vs the best model working model (model with 57 variables) with the t-test method to find out if any significant difference exists.

If the p-value between the two models is less than the chosen threshold, our observations are not so unlikely to have occurred by chance and there is a significant difference between the two models.

In python, the t-test can be checked with the help of scipy as

```
U1, p = ttest_ind(dstbest, othermodel[i])
 if p < 0.05:
       print("Significance␣difference␣between␣the␣model)
else:
"Model drawn from same population"
```

We checked the significance test with various models and there was no significant difference in the model as shown in the figure4.6.

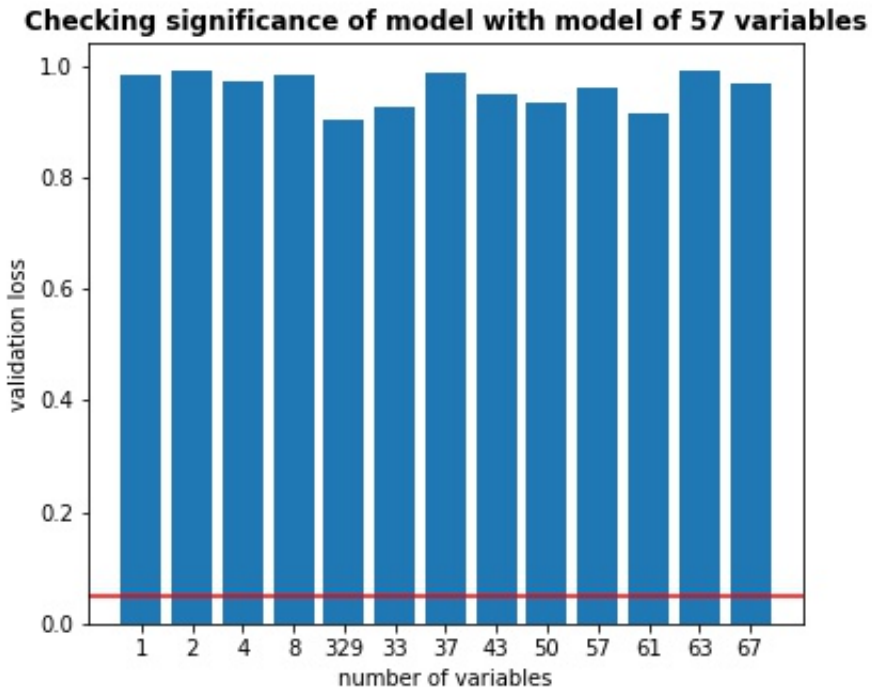


Figure 4.6: Checking significance of model with the model of 57 variables

4.6 LassoNet Regression

LassoNet extends lasso regression [5] and its feature sparsity properties to feed forward neural networks. It achieves feature sparsity by allowing a feature to participate in a hidden unit only if its linear representation is active. Two main ingredients of LassoNet are

- An introduction of penalty to the original empirical risk minimization that encourages feature sparsity. This process, by varying the level of penalty, transforms the combinatorial search to a continuous search.

- A proximal gradient algorithm which will needs just few lines of codes to a standard neural network.

Lassonet can be used for both regression and classification purpose, so it might be possible to extend it in the HDNNP situation too. This will be helpful as it can automate feature selections.

LassoNet will have one single residual connection and an arbitary feed-forward neural network. A hierarchical soft-thresholding optimizer helps the residual layer and the first hidden layer pass jointly.

There is a direct implementation of lassonet on pytorch, however, there are challenges to adapt lasso-net in the context of HDNNP.

— 5 —
Conclusion

In this thesis, we have made an attempt to select features in our artificial neural network.
In the previous section, we found that the highest precision of energy prediction was when our data had 57 features. If we compare our least loss, We have increased our precision. The least loss decreased from 0.28 for 22 variables to 0.14478 for 57 variables. (around .00109375 to 0.0005655 per atom). This drop is around 48%

The use of an adaptive learning rate has also given the flexibility to get a better precision by using just one-fifth of observations.

While the precision of prediction through 57 variables is higher than anyone in the model, It can also be noted that there is not much difference in the least loss across models with 50 variables to 72 variables.

If trained in a different environment, the precision can be attained at some other combinations while making sure selected variables is able to gather maximum information from the Behler Parrinello features.

Further study of analyzing weights taking care of 256 sub-models within a model can be another path to understand the proper set of variables that gives us precision at a higher level.

LassoNet [5] can be extended to HDNNP which might provide possibility of automated filtering of variables in the regular deep learning models. This method can be extended to our ANNs so we achieve smaller losses.

— A —
Building Dataset for 22 variables

A.1 Building Dataset for 22 variables

These are the parameters that we used to build initial 12 G2 variables

```
features_G2_12 = [[6.8,0.0013,0],\
                  [6.8,0.04,0],\
                  [6.8,0.1,0],\
                  [6.8,0.26,0],\
                  [6.8,0.66,0],\
                  [6.8,1.85,0],\
                  [6.8,1.85,1.017],\
                  [6.8,1.85,2.035],\
                  [6.8,1.85,3.052],\
                  [6.8,1.85,4.070],\
                  [6.8,1.85,5.087],\
                  [6.8,1.85,6.105]]
```

These are the parameters that we used to build 10 G5 variables.

```
features_G5_old = [[4.4,0.0013,1,1],\
              [4.4,0.0013,1,-1],\
              [4.4,0.0013,2,1],\
              [4.4,0.0013,2,-1],\
              [4.4,0.0013,4,1],\
              [4.4,0.0013,4,-1],\
              [4.4,0.0013,16,1],\
              [4.4,0.0013,16,-1],\
              [4.4,0.0013,64,1],\
              [4.4,0.0013,64,-1],\
```

— B —

Deciding Batch Size and Number of observations

Batch size is a hyperparameter that defines the number of observations to train before updating the weights and other parameters. Increasing batch size means at every epoch, there will be large computation which ultimately increases the total computation time. Therefore, choosing a proper batch size can decrease computational cost and also increase the prediction with accuracy.

We use adaptive learning process to decide the effects of batch and observation size in a model. Unlike a regular modelling which had 21000 observations, using adaptive learning helped us attain the precision in just 4200 observations.

In figure B.1, We can see that for the batch size of 64, a computational time (in terms of epochs) is quite smaller than models with batch size of 700 and 4200. We can also see that batch size 61 has a least validation loss minimized to 0.26 while batch size N=700 and 4200 has respectively 0.51 and 0.69. This training was done with inner layer of depth (10,10) with full data set of 4200.

Figure B.1, prepared prior to this thesis, helped us decide to go with batch size 64 and total observation 4200 for remaining part of our thesis.

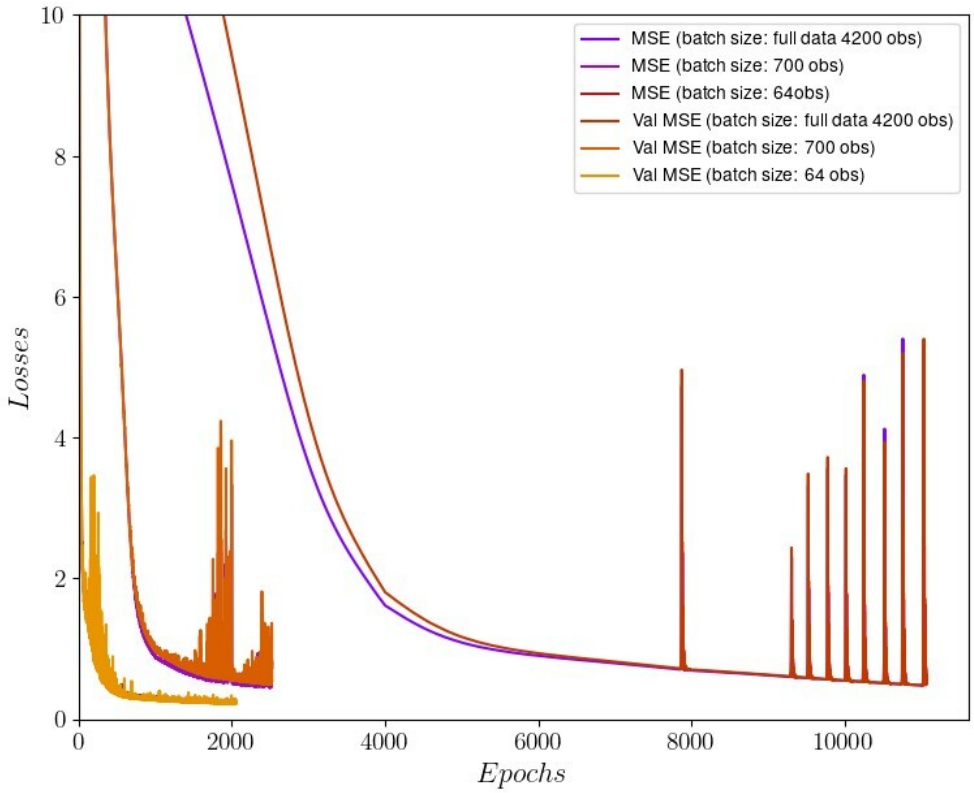


Figure B.1: Model trained with batches of 64, 700 and 4200 obs. Picture

— C —

Random Forest Regressor

An attempt to predict force and energy by adding temperature and solid/liquid as a categorical variable, we still didn't have a good accuracy in prediction. This is done for the whole dataset of 22 variables extracted from 329 features.

We started with a small dataset to find the effectiveness of Random Forest. Some of the results here are the continuity of the chapter four.

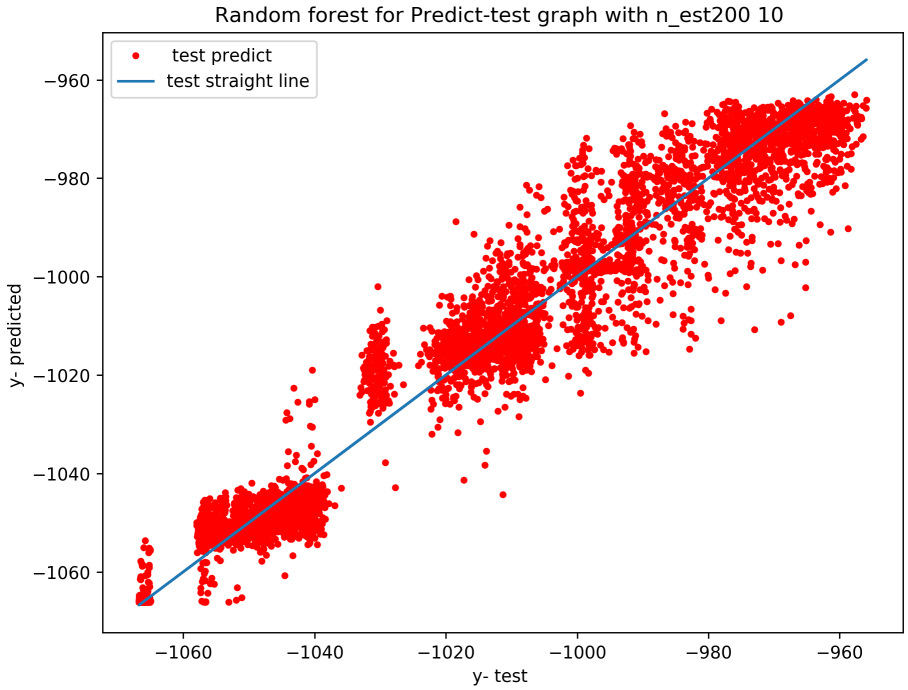
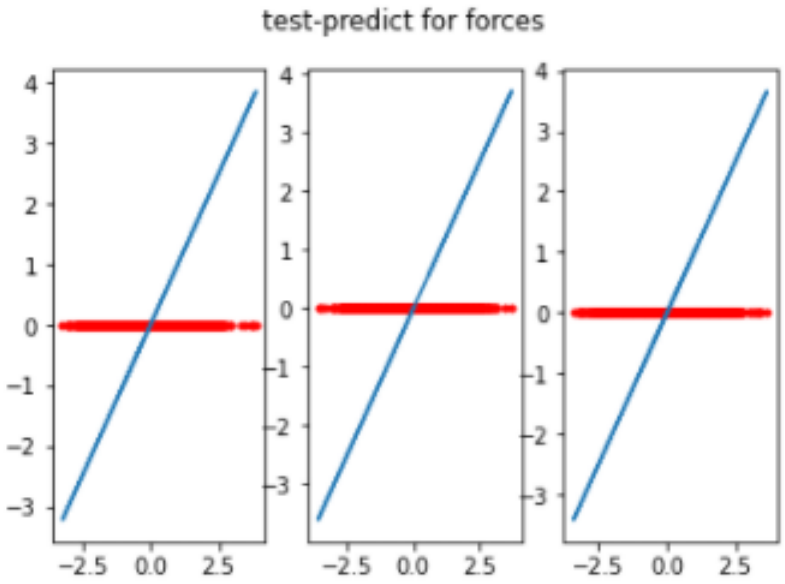


Figure C.1: A closer look after cropping

As we see in C.1 These energies at the test-predict graph is mostly the same value (or almost).

We also tried working to predict the force, but still, three outputs across the direction didn't work well with Random Forest.

The model has a noticeable mse for forces in each direction, but their test-predict for forces are almost near to zero.



	max_depth	mse_x	mse_y	mse_z
0	20	0.344343	0.34433	0.344338

Figure C.2: Test-Predict of Forces for max_depth 20

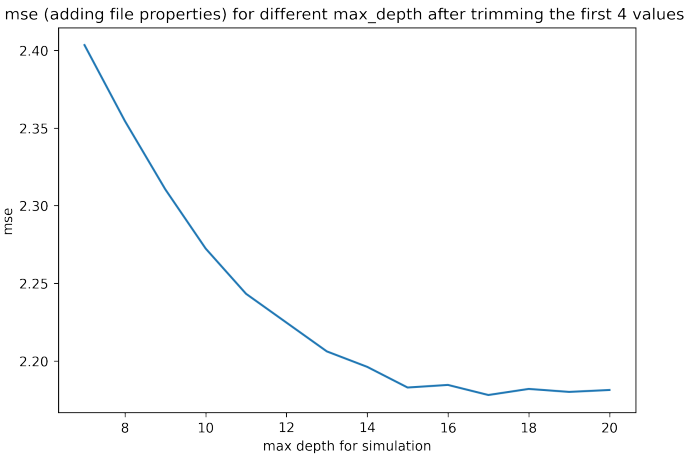


Figure C.3: Example of ANN Model of 22-10-10-1

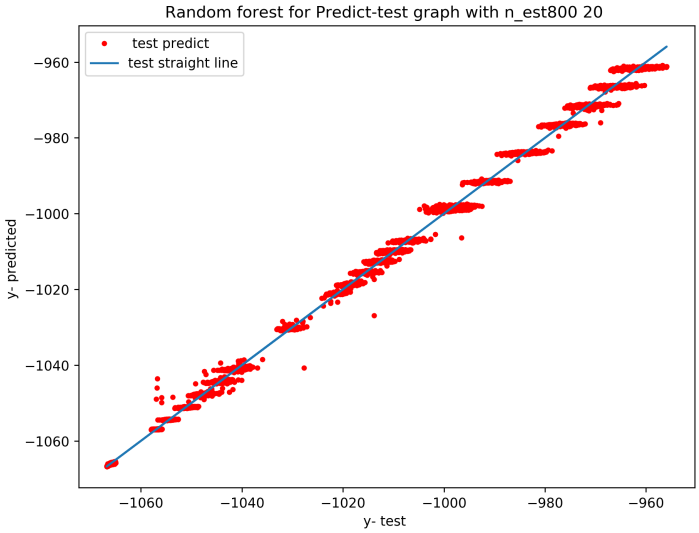


Figure C.4: Example of ANN Model of 22-10-10-1

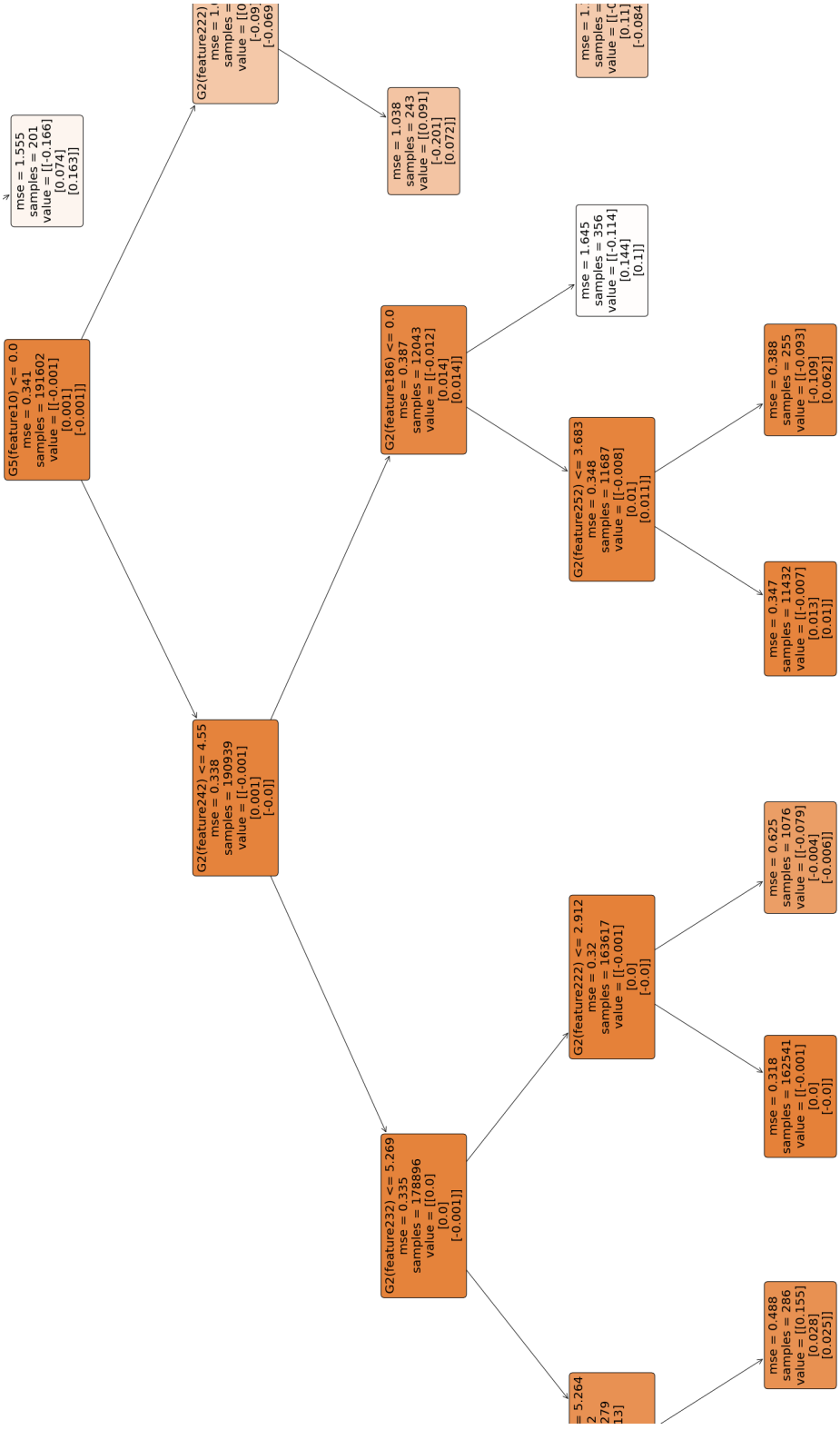


Figure C.5: Prediction of Forces through Random Forest Model:- Partial image

— D —

Removing One Variable at a Time from 57 variable

Index	removed_var	val_loss	Index	removed_var	val_loss
6	G2(feature41)	0.14248	50	G5(feature29)	0.16612
5	G2(feature33)	0.14633	16	G2(feature121)	0.16630
13	G2(feature97)	0.14868	10	G2(feature73)	0.16737
24	G2(feature185)	0.14914	37	G2(feature246)	0.16841
20	G2(feature153)	0.15038	7	G2(feature49)	0.16900
42	G2(feature266)	0.15087	31	G2(feature222)	0.16901
40	G2(feature258)	0.15098	4	G2(feature25)	0.16901
3	G2(feature17)	0.15167	29	G2(feature214)	0.16928
27	G2(feature206)	0.15365	18	G2(feature137)	0.16937
44	G5(feature5)	0.15540	28	G2(feature210)	0.17036
9	G2(feature65)	0.15566	54	G5(feature45)	0.17143
53	G5(feature41)	0.15571	36	G2(feature242)	0.17197
12	G2(feature89)	0.15597	1	G2(feature1)	0.17252
19	G2(feature145)	0.15671	35	G2(feature238)	0.17550
23	G2(feature177)	0.15678	11	G2(feature81)	0.17681
21	G2(feature161)	0.15704	22	G2(feature169)	0.17688
2	G2(feature9)	0.15809	45	G5(feature9)	0.17716
47	G5(feature17)	0.15822	55	G5(feature49)	0.17994
14	G2(feature105)	0.15875	48	G5(feature21)	0.18067
26	G2(feature202)	0.15933	49	G5(feature25)	0.18239
57	G5(feature57)	0.16087	38	G2(feature250)	0.18247
56	G5(feature53)	0.16096	46	G5(feature13)	0.18284
32	G2(feature226)	0.16205	43	G5(feature1)	0.18439
34	G2(feature234)	0.16208	51	G5(feature33)	0.18444
17	G2(feature129)	0.16217	33	G2(feature230)	0.18783
39	G2(feature254)	0.16316	25	G2(feature193)	0.19029
41	G2(feature262)	0.16409	8	G2(feature57)	0.19122
15	G2(feature113)	0.16551	52	G5(feature37)	0.19351
30	G2(feature218)	0.16565			

Table D.1: the least Loss of Model with 56 variables after dropping particular variable

Bibliography

[1] Jörg Behler. Atom-centered symmetry functions for constructing high-dimensional neural network potentials. *Journal of Chemical Physics*, 134(7), 2011.

[2] Jörg Behler. Constructing high-dimensional neural network potentials: A tutorial review. *International Journal of Quantum Chemistry*, 115(16):1032–1050, 2015.

[3] François Chollet. *Deep learning with Python*. Manning Publications Co, Shelter Island, New York, 2018. OCLC: ocn982650571.

[4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980 [cs]*, Jan 2017. arXiv: 1412.6980.

[5] Ismael Lemhadri, Feng Ruan, Louis Abraham, and Robert Tibshirani. Lassonet: A neural network with feature sparsity. *Journal of Machine Learning Research*, 22(127):1–29, 2021.

[6] Enrique Romero and Josep María Sopena. Performing feature selection with multilayer perceptrons. *IEEE Transactions on Neural Networks*, 19(3):431–441, 2008.

[7] Anthony Saliou. Kungliga Tekniska Höngskolan, Building neural network potentials for Lennard-Jones and Aluminium systems Anthony Saliou Master Thesis for the MSc in Engineering Physics KTH , Royal Institute of Technology. (July), 2020.

[8] Mao Ye and Yan Sun. Variable selection via penalized neural network: A drop-out-one loss approach. *35th International Conference on Machine Learning, ICML 2018*, 13:8922–8931, 2018.