A Deep Learning Study of the Potential Energy Surfaces of Small Triangular Molecules

Kartik Bhide, Nishil Mehta, Sharang Iyer and Alok Shukla[1,2]

[1)] *Indian Institute of Science Education and Research, Thiruvananthapuram, Kerala, India 695551*

[2)] *Department of Physics, Indian Institute of Technology Bombay, Powai, Mumbai 400076, India*

In this paper we discuss the use of neural networks in the study of the potential energy surfaces of small triangular molecules. *Psi4*, an open-source suite of *ab-initio* quantum chemistry programs, has been used to construct the potential energy surfaces of the molecules we have considered at the Hartree-Fock level of theory. Using a self-written simple neural network performing gradient descent calculations via back-propagation, we demonstrate the change in accuracy of prediction as the architecture of the network changes. Specifically, we show that neural networks are more accurate when larger input and hidden layers are implemented. We demonstrate the accuracies of prediction for a few combinations of optimizers and cost functions, employing neural networks programmed using Keras, an open-source machine learning library written in Python. We suggest the use of specific algorithms and cost functions for the study of PESs for a range of training set sizes. Finally, we discuss methods of further increasing the accuracy of results predicted by neural networks.

I. INTRODUCTION

Ab-initio molecular dynamics calculations are vital in the study of molecules and how they interact with each other. While Monte Carlo simulations are limited by the accuracy of the potential energy surface (PES) itself, often first-principles calculations come at great computational costs. As a result, while studying potential energy surfaces, a common technique is to calculate the energies of a small number of discrete geometries, and interpolate the remainder of the PES. Classical techniques of interpolation fall short of accurately describing PESs when it comes to systems with a large number of degrees of freedom; usually one must impose restrictions on the system in order to obtain and present good results. On the other hand, modern techniques such as machine learning and deep learning can be arbitrarily scaled to any number of dimensions with relative ease, and due to recent advancements stemming from a commercial demand, they can be done even on computing systems with basic hardware and software. While some approach such black-box methods with distrust, there is mounting evidence that deep learning is a valuable tool in a physicist's repertoire.

Quantum mechanical calculations, which in most practical cases involve numerically solving the Schrödinger equation, are usually computationally expensive. In order to speed up calculations, deep learning presents itself as a powerful data analysis and prediction tool. Characterized by the used of neural networks (NNs), deep learning has been used regularly as a computational tool in modern physics. NNs can be used to shorten computation time by predicting the properties of a system over a large output space based on an initial set of values. This predictive property of neural networks can be used in an immense range of settings and applications, one specifically being the study of potential energy surfaces. NNs have been used in the study and prediction of PESs for a long time, with a range of accuracies, network architectures and algorithms considered. While they have been used to varying degrees of success, there is no conclusive proof that NNs are the go-to tool when studying PESs. Further, there are not many findings regarding the use of NNs in the study of the PESs of small triangular molecules.

A common practice in data science is to split a dataset into two so called training and testing sets; models are trained using the former, and tested for their accuracy of prediction using the latter. In order for networks to capture patterns present in the dataset, usually the training set is much larger than the testing set: an 80:20 split is normally a good starting point, and in most cases gets the job done. In this paper, we go against the grain and split the datasets in the opposite fashion. The motivation for doing the same is as follows: as the theory and algorithms behind neural networks advance, in tandem with the progress made in the systems that run such codes, it is of increasing importance to identify the minimum amount of computation required in order to obtain results upto any arbitrary degree of precision. Identifying a general thumb-rule for all PESs is beyind the scope of this paper, yet we have nonetheless made an effort in this direction.

We have systematically studied the PESs of the molecules we have considered using both Keras neural networks, and a self written neural network. For the latter, we have looked at the change in the accuracy of prediction as the architecture of the network changes. Specifically, we have varied the size of the hidden and input layers. Further, using Keras neural networks, we show that the implementation of certain combinations of optimizers and cost functions are more accurate than others for different training set sizes.

II. PROCEDURE

For the purpose of this study, datasets were constructed using *Psi4*, an open-source density functional theory program on consumer-grade laptops. Due to computational limitations, energy calculations were performed at the Hatree-Fock level of theory using the cc-pvdz basis set. In order to reduce the degrees of freedom, the geometries of the molecules were varied by changing their physical parameters, i.e. bond lengths and bond angles. By finding the optimized geometry of the considered molecules, using inbuilt *Psi4* functions, datasets were constructed around the optimized geometry using the following classification: the PESs of molecules not exhibiting Jahn-Teller distortion, i.e. possessing a D_{3h} optimized geometry (hereafter referred to as JT- molecules) were constructed using the physical variables described in figure 1a, whereas the PESs of those molecules that did show Jahn-Teller distortion , i.e. possessing a C_{2v} optimized geometry (hereafter referred to as JT+ molecules) were constructed using the physical variables as described in figure 1b. This classification has no effect on the results presented, and was merely considered in order to simplify and understand the code.

Specifically, we have constructed the PESs of the following molecules: B_3, B_3^+, B_3^-, Li_3,

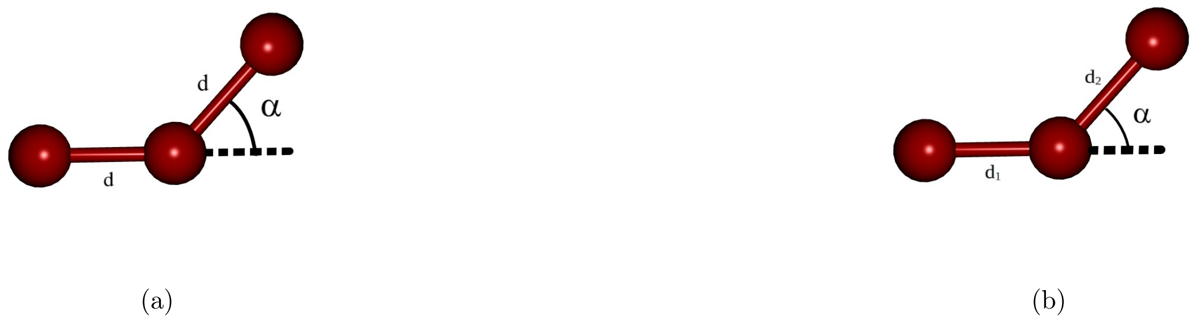


Figure 1: Physical parameters used for JT- and JT+ molecules respectively

Li_3^+, Be_3 and H_2O. Among these, B_3, B_3^+, Li_3^+, Be_3 and H_2O have been classified as JT- molecules, and B_3^- and Li_3 have been classified as JT+ molecules.

Datasets were split into training and testing sets, the former required for training networks, while the latter being used for testing the accuracy of prediction of the network. Throughout the study, the datasets were split in such a way that the training set was 1 % to 10 % of the parent dataset. In every case, inputs to the networks were either ordered tuples of the physical parameters with which the dataset was constructed, or algebraic forms of the same ordered tuples.

Due to the random nature of the inital weights and biases, each individual run of a neural network has the possibility of giving different results. From an abstract perspective, this can be imagined as the optimizer settling in different local minima of the cost function, due to random starting points on the surface of the cost function itself. In order to avoid this, networks were run multiple times for every set of network architecture, hyper-parameters, optimizers and cost functions, with the accuracy of the best performing network being noted. In this study, we have considered root-mean-squared (RMS) error between the predicted and testing sets as a measure of accuracy of neural networks.

In Section IV B where we vary the size of the input layers of the neural networks, we have modified the inputs as follows: taking $x = (d, \alpha)$ and $x = (d_1, d_2, \alpha)$ as ordered tuples of the physical parameters in the case of JT- and JT+ molecules respectively, we construct the input layer by concatenating various functional forms of X, i.e. x, x^2, $x^{0.5}$, $\sin(x)$, $\exp(x)$ etc. depending on the size of the input layer considered. Thus, as an example where we have considered x, x^2, $x^{0.5}$ in the case of JT- molecules, the input layer becomes (d, α, d^2, α^2, $d^{0.5}$, $\alpha^{0.5}$). This same procedure has been followed for other input layer sizes.

III. THEORY AND ALGORITHMS

A. Simple Neural Network (SNN)

In data science and data analysis, mathematical models for regression and curve-fitting are required to understand the trends in the data. While machine learning algorithms use brute force to compute the value of a function that is to be fitted, they lack a means of physically interpreting systems. Furthermore, machine learning is often difficult to implement for systems with a large number of degrees of freedom. Deep learning overcomes these limitations by intuitively learning the patterns present in a dataset. It is characterized by the use of neural networks; modelled after the nervous systems of living beings, a generic definition of an artificial neural network has given by Kohonen: "Artificial neural networks are massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations which are intended to interact with objects of the real world in the same way as biological nervous systems do."

Neural networks, as the name itself suggests, are composed of 'neurons'. In general, a neuron is any computational object that can store information. These neurons are organized into layers: an input layer, multiple hidden layers, and an output layer. Inputs, or features, which are tied to their respective outputs in the dataset, are fed into the network via the input layer, with the predicted result being displayed in the output layer. Hidden layers function as an abstract tool in recognizing the patterns present in the dataset. In this study, the physical parameters of molecules, i.e. bond lengths and bond angles have been taken to be the features, while the energies of each discrete geometry have been considered as the outputs.

A network learns the patterns present in a dataset by means of tunable parameters called weights and biases. For multi-dimensional networks, these are represented as matrices, and by performing dot product and matrix algebra operations between the weights, biases and inputs associated with each layer, a network eventually learns the connections between the inputs and outputs it has been given. At every step in this process, the inputs to a layer are passed through an 'activation function' whcih provides a source of non-linearity to the network and allows for the fitting of arbitrary functions.

At the output layer, the network evaluates the accuracy of prediction using what is known as a cost function. Minimizing the value of this cost function is of importance; a lower value implies that a network has better learnt the connections between the inputs and outputs. This is done using the principle of back propagation: by iteratively modifying the weights and biases, the value of the cost function is reduced and the network learns a better fit of the given dataset. One of the most common methods of achieving this is by using a gradient descent algorithm, where the value of the cost function is reduced by altering the weights and biases in such a way that the value of the cost function, also known as the loss, follows the negative gradient of the cost function at that point, which has been numerically calculated for the same.

A mathematical description of these operations has been given by Handley and Popelier, and can be further studied from the references given in their paper.

We have used the sigmoid function as an activation function, which normalizes all inputs to a value between -1 and 1. Formally, it is expressed as follows:

$$\phi(z) = \frac{1}{1 + \exp(-z)}$$

We have considered mean squared error (MSE) loss function as an assessment of the network's accuracy, which is expressed as:

$$MSE = \frac{1}{N} \cdot \sum (y_i - \hat{y}_i)^2$$

Here, $\hat{y}$ is the mean of all outputs y_i of the dataset, with N being the size of the dataset.

Relying heavily on NumPy and its inbuilt methods, we have programmed a simple neural network that can perform basic gradient descent optimization of the MSE cost function. Each run of the network consisted of 10,000 iterations, or epochs through the datasets, with the weights and biases for each individual run being initialized randomly from the standard distribution.

It is important to note that the SNN we have constructed is essentially a trivial neural network, as it includes the bare minimum for what a computational procedure requires to be called a neural network. Even though the results we have obtained using the SNN are not comparable with previously reported accuracies, the trends we have studied with respect to the architecture of the SNN are still valid, and provide valuable insights into the use of neural networks in studying potential energy surfaces.

B. Keras Neural Network (KNN)

Written in Python, Keras is an open-source machine learning library aiming to provide modular, accessible and user-oriented programming of neural networks. Capable of running on top of back-ends such as TensorFlow, Theano or PyTorch, it can be used as an API to ease deep learning studies. While the principles of backpropagation and gradient descent behind neural networks implemented using Keras are the same as those of the SNN, there are some differences in terms of the optimizers and cost functions we have considered.

We have used Keras over a TensorFlow back-end, with network architectures of 2-24-24-1 for JT- and 3-24-24- 1 for JT+ molecules. Layers of the netowrk were constructed by calling on the inbuilt 'Dense' layer, which is functionally similar to the layers of the SNN, with each run of the network consisted of 120 epochs. Once again, All weights and biases were initialized as random values from the standard distribution, using inbuilt functions. We have used the 'relu' activation function for each layer. Extensively used in contemporary deep learning studies and known to provide faster convergence and better results in general, it is expressed as:

$$\phi\left(z\right) = \max\left(0,\, z\right)$$

The following combinations of optimizers and cost functions have been considered for the KNN, using Keras' inbuilt functions. Detailed derivations of the update rules have already been published; here we merely mention the final update rules, using the same notations. Note that the initial values of all hyper-parameters have been taken to be the default values as provided by Keras.

1. Stochastic Gradient Descent + Mean Squared Error (SGD + MSE):

Presented as an improved form of the gradient descent algorithm, the SGD algorithm performs parameter updates for each data point, or a 'batch' of data points. This behaviour gives it an inherent stochastic nature, allowing for faster convergence, though not necessarily better performance compared to the gradient descent algorithm. The parameter update rule for the SGD optimizer is as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta^C \left(\theta_t,\, x^i,\, y^i\right)$$

We have paired the SGD algorithm with the MSE cost function, which has been described in Section III A.

2. ***RMSprop + Mean Squared Logarithmic Error (RMSprop + MSLE):***

RMSprop is an unpublished adaptive learning rate algorithm, seeking to adress the shortcomings of the Adagrad algorithm, which has not been described here. Assosciating a learning rate with every parameter, it performs larger updates for parameters associated with infrequent features and smaller updates for parameters associated with features appearing frequently in the dataset. The RMSprop update rule is defined as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \varepsilon}} \cdot g_t^2$$

Along with the RMSprop optimizer, we have used the Mean Squared Logarithmic Error (MSLE) cost function:

$$MSLE = \frac{1}{N} \cdot \sum \left(\log\left(y_i + 1\right) - \log\left(\hat{y} + 1\right)\right)^2$$

3. ***Adam + Binary Cross-Entropy (Adam + BCE):***

Adam, or Adaptive Moment Estimation, is another adaptive learning rate technique that computes learning rates that can be updated for all weights and biases. Invoking a ball rolling down a slope model, wherein the optimizer 'rolls' down the surface of the cost function in search of local minima, the Adam optimizer can be thought of as introducing friction to the system, thereby giving the optimizer a preference of flat minima on the surface of the cost function.

The Adam update rule is presented as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \cdot \hat{m}_t$$

We have paired the Binary Cross-Entropy (BCE) cost function along with the Adam optimizer, as specified in the Keras documentation. Although it is usually used in cases of binary classification, we have implemented it nonetheless. Such a pairing has produced good results compared to other combinations, despite it not making sense to implement a cost function that is not generally considered in such a use-case. As such, we have not presented the definition of the cost function here.

4. Adamax + Poisson (Adamax + P):

The Adam optimizer makes use of the l_2 norm of the past gradients. By considering the l_∞ norm, the update rule is defined as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} \cdot \hat{m}_t$$

Along with the Adamax optimizer, we have considered the Poisson cost function, which gives a measure of how much a distribution varies from the well-known Poisson distribution. Formally, it is expressed as follows:

$$P = \frac{1}{N} \sum \left(\hat{y} - y_i \cdot \log\left(\hat{y}\right) \right)$$

5. Nadam + Log-cosh (Nadam + LC):

The Nadam (Nesterov-Accelerated Adaptive Moment Estimation) optimizer, as the name suggests, combines the benefits of the Adam optimizer and the Nesterov Accelerated Gradient. The update rule is expressed as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon} \cdot \left(\beta_1 \hat{m}_t + \frac{(1-\beta_1) \cdot g_t}{(1-\beta_1^t)} \right)$$

The log-cosh cost funcion has been used along with the Nadam optimizer. In most cases it performs like the MSE cost function, but is less susceptible to the occasional wildly inaccurate prediction. It is expressed as:

$$LC = \sum \log\left(\cosh\left(y_i - \hat{y}\right)\right)$$

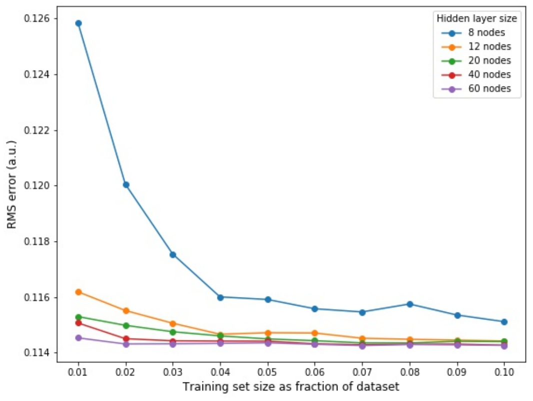
Fraction of dataset as training set	B_3	B_3^+	B_3^-	Li_3	Li_3^+	Be_3	H_2O
0.01	0.12585	0.12175	0.03289	0.02469	0.02718	0.02465	0.06313
0.02	0.12005	0.11713	0.02402	0.02214	0.01737	0.01494	0.05549
0.03	0.11754	0.11632	0.02282	0.01555	0.02056	0.01321	0.05189
0.04	0.11600	0.11531	0.02193	0.01453	0.01531	0.01039	0.04929
0.05	0.11591	0.11520	0.02045	0.01478	0.01410	0.00843	0.04969
0.06	0.11557	0.11507	0.02063	0.01060	0.01152	0.00578	0.04993
0.07	0.11546	0.11495	0.02041	0.01007	0.01092	0.00741	0.04929
0.08	0.11575	0.11451	0.02068	0.01101	0.01068	0.00621	0.04824
0.09	0.11535	0.11493	0.01971	0.00921	0.01020	0.00638	0.04816
0.10	0.11511	0.11469	0.01906	0.00824	0.00996	0.00583	0.04782

Table I: RMS errors between predicted and test sets for various molecules using the SNN

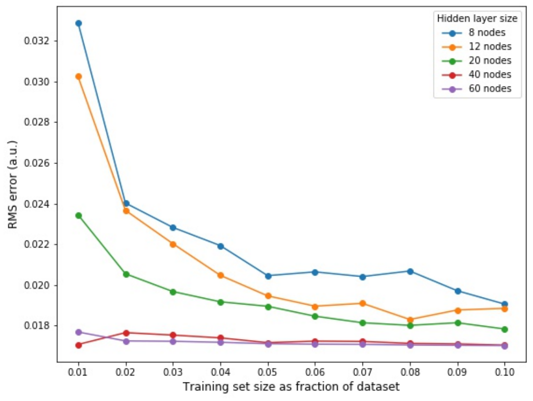
IV. RESULTS AND DISCUSSION

Table I summarizes the results of using the SNN to study of the datasets we have constructed, for fixed network architectures of 2-8-8-1 and 3-8-8-1 for JT- and JT+ molecules respectively. Errors have been presented as RMS error between the predicted and test output sets in atomic units. In general, we see a decrease in the error as the size of the training set increases, though it is not a significant improvement in most cases. The variation in the errors across molecules can be best attributed to the qualities of the individual datasets, i.e. the SNN performs much better on the Be_3 dataset than on the others, and this can be explained by the high level of smoothness of the Be_3 potential energy surface on the small scale.

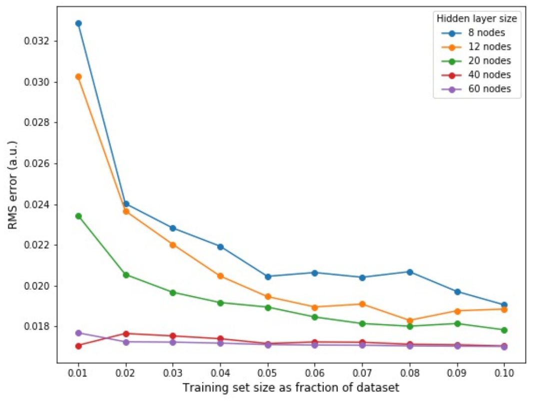
The errors presented in I act as a marker-point from which we study further aspects of the SNN. In sections IV A and IV B we discuss how the SNN performs as the network architecture varies. Further, in Section IV C, using the KNN, we implement various combinations of optimizers and cost functions and compare their relative performances for each dataset.



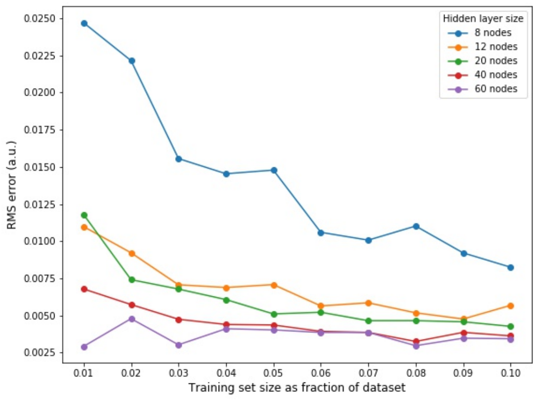
(a) B_3



(b) B_3^+



(c) B_3^-

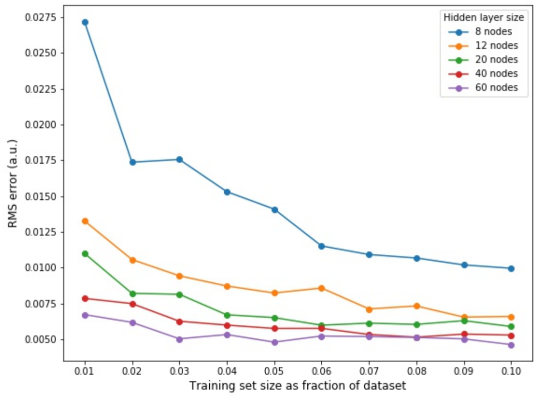


(d) Li_3

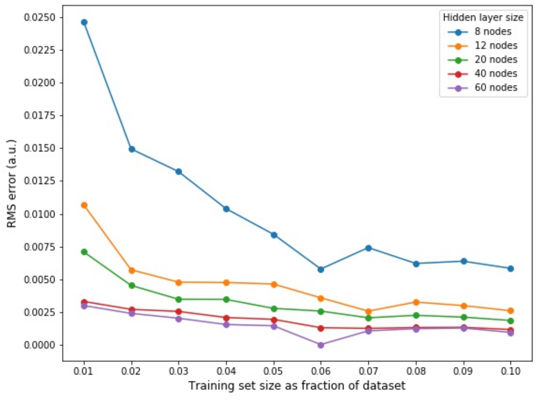
Figure 2: RMS errors between predicted and test sets for various molecules using the SNN while varying the size of hidden layers. Numbers provided in the legend boxes are to be interpreted as follows: if x is the number of nodes in the hidden layers, the network architecture is 2-x-x-1 and 3-x-x-1 for JT- and JT+ molecules respectively.

A. SNN: varying hidden layer size

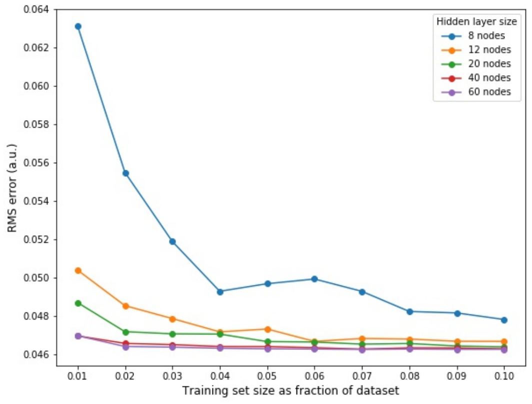
In this section we look at how the results predicted by the SNN vary as the number of nodes in the hidden layers increases. Figures 2a to 2g detail the results of varying hidden layers sizes and the outcome of this change in network architecture with respect to the accuracy of prediction, with errors being presented in atomic units.



(e) Li_3^+



(f) Be_3



(g) H_2O

Figure 2: (contd.) RMS errors between predicted and test sets for various molecules using the SNN while varying the size of hidden layers. Numbers provided in the legend boxes are to be interpreted as follows: if x is the number of nodes in the hidden layers, the network architecture is 2-x-x-1 and 3-x-x-1 for JT- and JT+ molecules respectively.

As one would expect from such a study, in general, the error of prediction decreases as we increase the size of the hidden layers. This can be understood by looking at this result from an abstract perspective: if the first hidden layer is thought to pick up on large scale patterns in the dataset, while the second hidden layer identifies patters on the smaller scale, then increasing the number of nodes in the hidden layers will certainly increase the accuracy of prediction.

In most cases, there is not much change in the error when the size of the hidden layers in 40 and 60 nodes. This suggests that for every dataset there could be some optimal size of hidden layers (and indeed number of hidden layers, which we have not studied here) for which the accuracy of prediction is maximized.

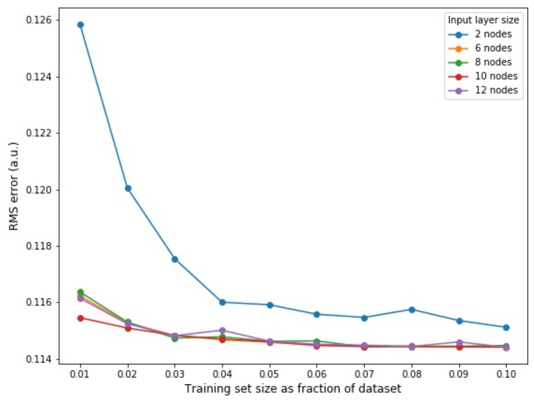
Further, we see that as the hidden layer size increases, convergence of the errors with respect to increasing training set size is much faster for larger neural networks. In most cases, for larger networks, the error reaches some near-constant value around the 4% - 7% training set size range, while it appears that the smallest network (i.e. 8 nodes in both hidden layers) does not reach some limiting error value even at 10% training set size. This, along with the previous observation of a possible optimal network architecture, may imply that there is also an optimal training set size beyond which computational cost supercedes the need for more accurcate predictions.

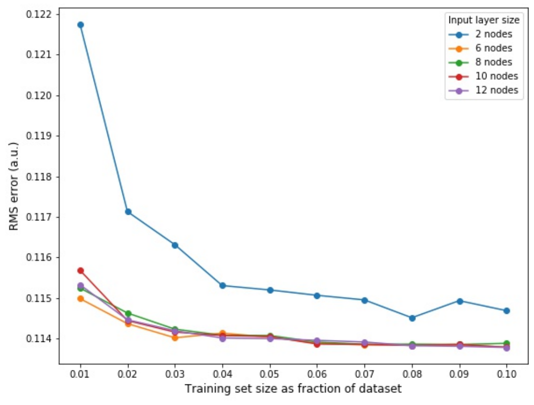
B. SNN: varying input layer size

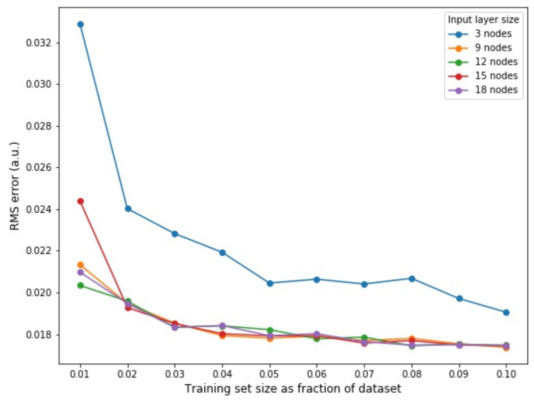
Here we look at how the accuracy of prediction changes as the number of nodes in the input layer changes. Figures 3a to 3g show the same, for the datasets we have used. Again, errors have been presented in atomic units. The procedure involved in changing the number of inputs has been described in Section II.

For every dataset, increasing the number of input nodes drastically reduces the error. However, according to the way we have selected the inputs, there does not seem to be any general preference for a certain number of inputs. This can be explained by the quality and properties of the datasets itself; depending on the dataset, certain types of functional forms of the physical parameters will provide better results.

We also see that for the smallest networks, the errors typically do not converge to some value, whereas for larger networks and for some datasets, the errors converge to some value around 5% - 8% training set size. This shows that networks with larger input layers tend to perform better than smaller networks, both in terms of error and error convergence.


(a) B_3


(b) B_3^+


(c) B_3^-

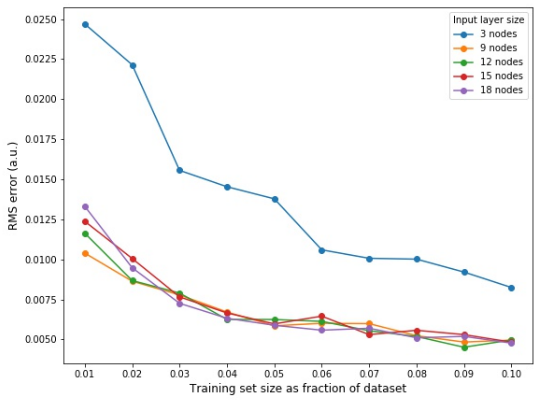
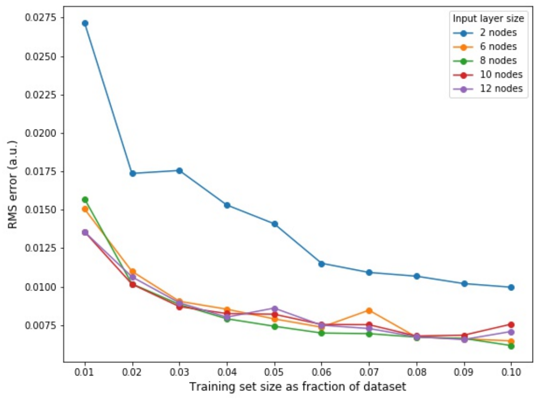

(d) Li_3

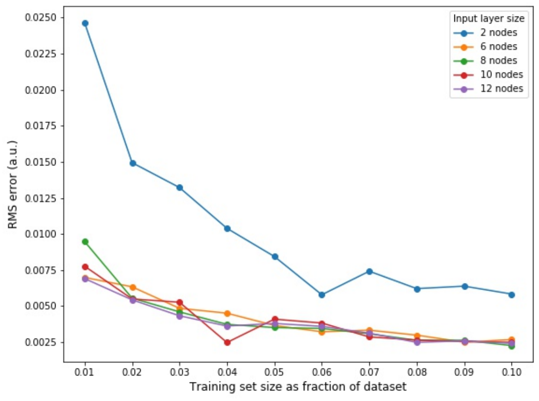
Figure 3: RMS errors between predicted and test sets for various molecules using the SNN while varying the size of hidden layers. Numbers provided in the legend boxes are to be interpreted as follows: if x is the number of nodes in the input layer, the network architecture is x-8-8-1 for both JT- and JT+ molecules.

C. KNN: optimizers and cost functions

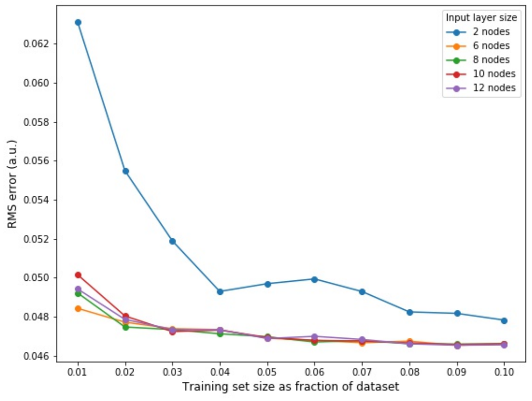
Using the KNN, we have studied the constructed datasets using the combinations of optimizers and cost functions described in SectionIII B. The KNN performs better than the SNN in every aspect, due to the high level of underlying software and optimized deep learning algorithms. In some cases the errors provided are better than the errors provided



(e) Li_3^+



(f) Be_3



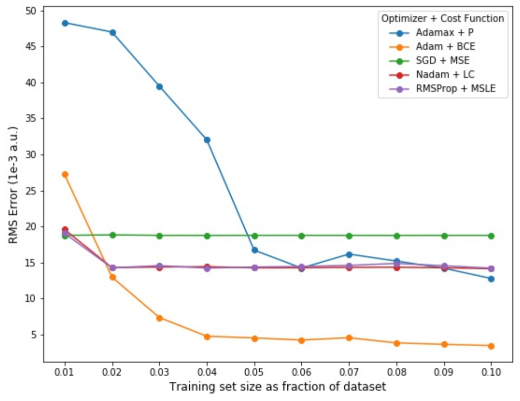
(g) H_2O

Figure 3: (contd.) RMS errors between predicted and test sets for various molecules using the SNN while varying the size of hidden layers. Numbers provided in the legend boxes are to be interpreted as follows: if x is the number of nodes in the input layer, the network architecture is x-8-8-1 for both JT- and JT+ molecules.

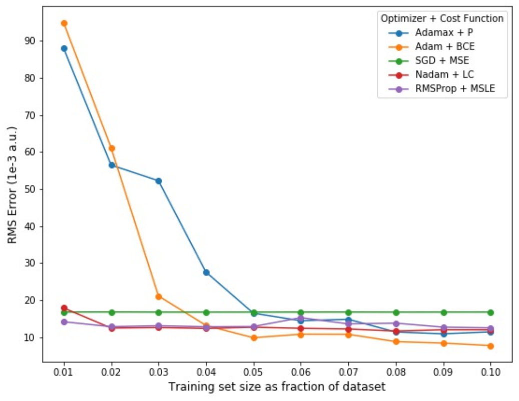
by the SNN by 4 orders of magnitude.

Figures 4a to 4g show the errors of prediction, in 10^{-3} atomic units, for all combinations of optimizers, cost functions and datasets.

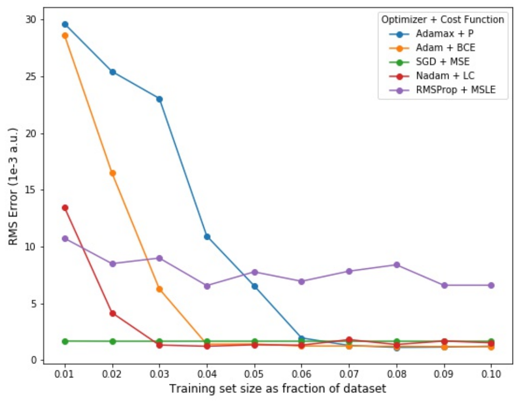
In terms of improvement of accuracy of prediction as the size of the training set increases, the SGD + MSE combination is the worst performing throughout all datasets. While other combinations might start at a higher error, they improve as the training set size increases,



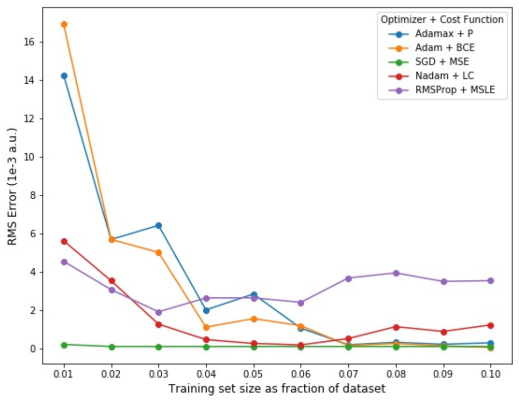
(a) B_3



(b) B_3^+



(c) B_3^-

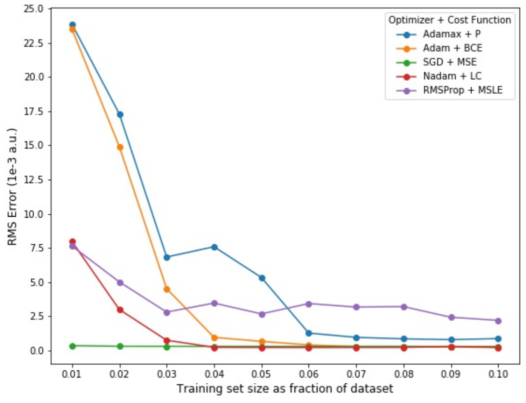


(d) Li_3

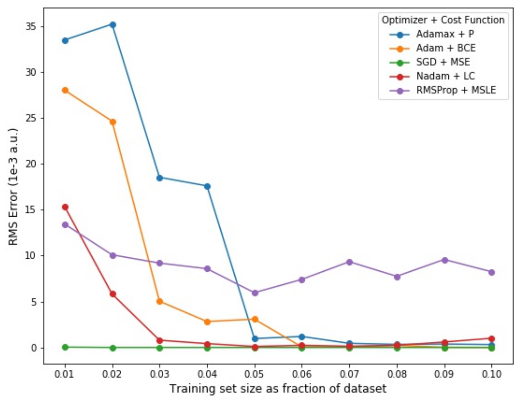
Figure 4: RMS errors between predicted and test sets various molecules using the KNN, for several combinations of optimizers and cost functions

and eventually overtake the SGD + MSE combination after 3% - 6% training set size.

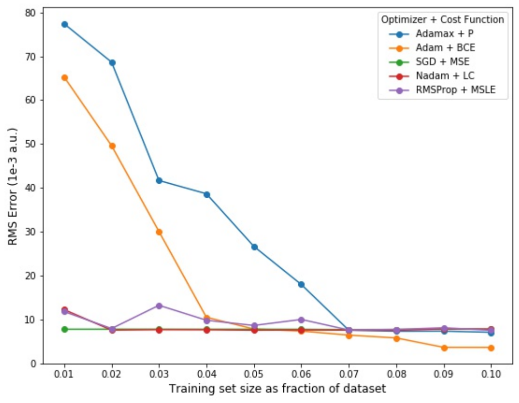
Among the optmizers coming from the adaptive learning rate (ALR) family, the Adam + BCE and Adamax + P combinations generally perform better than the Nadam + LC combination for training sets greater than 4% of the parent datasets. For much larger train-test splits, the Adam + BCE combination performs better than all other combinations. While the Nadam + LC combination performs better for smaller training sets, the accuracy converges faster than the other combinations, allowing for better relative performace for the other combinations. The RMSprop + MSLE combination, though performing better for



(e) Li_3^+



(f) Be_3



(g) H_2O

Figure 4: (contd.) RMS errors between predicted and test sets various molecules using the KNN, for several combinations of optimizers and cost functions

lower training set sizes, does not improve and appears to fluctuate around some mean value as training set size increases.

It is interesting to note that the BCE cost function performs so well compared to other combinations in this use case. The BCE cost function is typically used for binary classification, and it is surprising to see it give good results in a prediction environment. Although the exact theoretical and mathematical reasoning behind this behaviour is beyond the scope of this paper, this property of the Adam + BCE combination can be further studied in the context of quantum chemistry and in the study of potential energy surfaces.

V. CONCLUSION

In this paper we have studied the various aspects of neural networks applied to the potential energy surface of small triangular molecules. Using a self constructed neural network, we show that the accuracy of prediction increases as the size of both input and hidden layers increases. Further, using neural networks programmed using Keras, we show that certain combinations of optimizers and cost functions perform better than others, for various ranges of training set sizes.

While we have used the SNN to demonstrate concepts in neural networks, the KNN is a more practical network in terms of real life modelling of potential energy surfaces. We suggest that for the study of PESs using neural networks, it is ideal to implement the Adam+BCE combination if one wishes to use larger training sizes, whereas the SGD+MSE combination is optimal for smaller training sizes, even though it does not improve as the size of the training set increases. However, it is possible that there are better performing network architectures, activation functions and combinations of optimizers and cost functions that can be implemented. Thus, In the ideal situation, it is best to explore which type of neural network, optimizer, cost function etc. best suits the dataset, as deep learning and its related fields depend highly on data and the properties of the dataset.

There are several ways in which neural networks can give better results while studying potential energy surfaces. Ground zero for optimizing results is the use of better datasets; using higher levels of theory while constructing PESs gives a smoother topology of the PES hypersurface, and in general will improve predictive results. One can also use different types of neural networks, such as convolution or recurrent networks. Using different activation functions, optimizers and cost functions can also lead to better results; again, the results given by neural networks are data-dependent, and it is necessary to explore which combination of hyper-parameters is optimal for a dataset.

For the purpose of optimizing results, one can also consider other higher algebraic forms of the physical parameters, such as powers of the bond lengths, trigonometric ratios of the bond angles, exponents, log functions etc. If neural networks are thought as tools that learn pseudo-analytical functional relationships between the inputs and outputs they are given, introducing higher order terms into the dataset can increase the accuracy of prediction. The exact choice of which terms to use, once again has to be explored before presenting results.

It is important to note that the results we have obtained are at a very low computational level, and they can certainly be improved if more complex hardware and software are implemented. Regardless, the the trends we have studied and observed still hold true even for the simple neural networks we have used.

VI. ACKNOWLEDGEMENTS

The authors would like to thank Dr. Alok Shukla, Dept. of Physics, IIT Bombay for his constant support and advice during an internship completed under his guidance. Without his direction and suggestions, this paper would not have taken shape.