Solving Minimum Energy Structures with Neural Networks

Brian Barch
Norman Tubman

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-104
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-104.html

May 12, 2017

Solving Minimum Energy Structures with Neural Networks

by Brian Barch

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Umesh Vazirani
Research Advisor

(Date)

* * * * * * *

Dr. Norman Tubman
Postdoctoral Advisor

(Date)

Solving for Minimum Energy Structures with Neural Networks

Brian Barch and Norman Tubman

Abstract

In this paper, we train a neural network on atomic configurations to predict energy as a function of atom position, then use this neural network to perform optimization to solve for minimum energy atomic configuration. This is a problem of interest because it could potentially provide a boost to both accuracy and speed over traditional numeric methods of solving for the structure of molecules. Previous papers have shown that neural networks trained on the results of numerical simulations can reproduce those results to high accuracy. We construct one such neural network and experiment with new methods of optimizing its parameters, then use it as a function to optimize in order to find the minimum energy configuration for a systems of a few homonuclear atoms. The results are promising, with our neural network accuracy beating that of the baseline neural network for the problem, and the minimization results showing an improvement over the data the neural network was trained on.

1. Introduction

Simulating atomic systems is a problem of interest to physicists because it has applications to many other fields, such as protein folding, solving for crystal structure, and predicting phase transitions. A key tool in atomic simulation is the calculation of system energy as a function of atomic positions, known as a potential energy surface (PES). Forces calculated from the gradient of the PES can be used to simulate molecular dynamics, or the PES can be optimized over to find the minimal energy structures the configuration would realistically settle into.

Classically, mapping from atomic position to energy is a straightforward task, requiring only direct interactions between particles. In quantum mechanics however, the existence of electron wavefunctions complicate the calculation. In order to fully calculate the quantum mechanical energy of an atomic configuration, one would need to fully solve for the three-dimensional wavefunction of each electron from Schrödinger's equation, a task not possible analytically. Quantum chemistry methods such as Quantum Monte Carlo (QMC) approximate these wavefunctions numerically to high accuracy, but are slow and have poor computational complexity scaling in the number of atoms in the system. Classical potentials are a faster but less accurate which are also incapable of predicting purely quantum phenomena, such as photosynthesis. A middle ground between the two extremes is struck by Density functional Theory (DFT). DFT defines a map known as a density functional from electron density to system energy, and uses numerical iterations to approximate both the electron density and the PES.

Though DFT is useful in general, it is often necessary researchers to reproduce QMC and DFT accuracies on larger atomic systems and with faster speeds, to allow larger simulations to complete within realistic timeframes. Neural Networks (NNs) have emerged as a possible solution to this. NNs trained to predict system energy from atomic coordinate have been shown to be able to reproduce the PESs generated by DFT and QMC calculations to high accuracy [1]. Because these NNs are simple and feed-forward, once trained they are capable of quickly predicting for large numbers of atoms without the need for successive iterations. This allows the simulation of processes that would not otherwise be possible, such as determining the effect of Van der Waals interactions on the freezing of water [2]. Furthermore, one can analytically derive the gradient of energy with respect to atomic coordinate from a NN, which can be used to find forces on the atoms or for minimization of energy. While this is also possible with DFT calculations, QMC produces only discrete predictions of energy from atomic position – predictions of forces from QMC data have relatively poor accuracy [3]. Since NNs are capable of reproducing a PES to higher accuracy than DFT, it is believed that NNs trained on QMC data could then be used for accurate calculation of gradients, allowing fast and more accurate simulation and energy minimization than would otherwise be possible.

2. Related Work

An introduction and overview to the use of NNs to construct PESs is provided by J. Behler, in [1]. This paper includes a review of the prediction accuracies of NN models on various atomic systems. While in [1], Behler discusses the use of NNs for a number of different types of systems (e.g. molecule on surface), this present paper is restricted to atoms in a 3D periodic environment. In [4], Behler goes further into detail on the NN structure, as well as providing a full description of how atomic coordinates are featurized through the use of symmetry functions. [4] also describes the particular symmetry functions used in the present paper.

An example of the application of NNs to molecular simulation can be seen in [2], in which Moriavietz et. al. use NNs trained on DFT data from density functionals with and without Van der Waals interactions to compare the accuracies of those functionals in estimating the melting point of water. A method of solving for minimum energy structure from QMC data is demonstrated by L. Wagner and J. Grossman in [3] using a quadratic fit model. It is believed that by using a NN described in [1] and [2] rather than a quadratic fit model, the results of [4] can be improved upon.

Aenet [5-8] provides a NN package that implements the NN structure described in [1] and [2], and allows one to calculate gradients from the predicted PES for simulation and minimization. Aenet served as the baseline to which results in this paper could be compared.

QuantumEspresso [9, 10] provides an environment in which DFT energies of configurations can be calculated and molecular dynamics simulations run. It was used to generate data for minimization and to evaluate the results. Data for the training of symmetry function parameters was generated by N. Tubman, et. al. in [11] using QMC, then augmented with more data generated using Aenet.

3. Neural Network and Symmetry Functions

To train a NN to predict a PES for an atomic system, data consisting of atomic configurations and associated energies must first be generated either by experiment or numerical methods. Though ultimately our goal is to reproduce QMC results with NNs, at present we have only used DFT generated data. Prior to training, atomic configurations are featurized through what are known as symmetry functions [1, 4, 7], which represent the local environment of each atom as a distinct vector. The NN can then be trained on these vectors to predict the PES.

3.1 Atomic Neural Networks

The NN structure used for predicting PESs is known as the atomic neural network structure [1,4]. While it is feasible to train a fully connected NN on the interatomic distances for configurations of a few atoms [1], better accuracy is achieved for larger systems by first representing each atom’s local environment as a vector, then training a sub-NN known as an atomic neural network to predict the energy of a single atom at a time given that atom’s vector [1,4]. A full configuration of atoms is reexpressed as a set of atom vectors, individual atomic energies calculated, and atomic energies summed to get the total energy of the

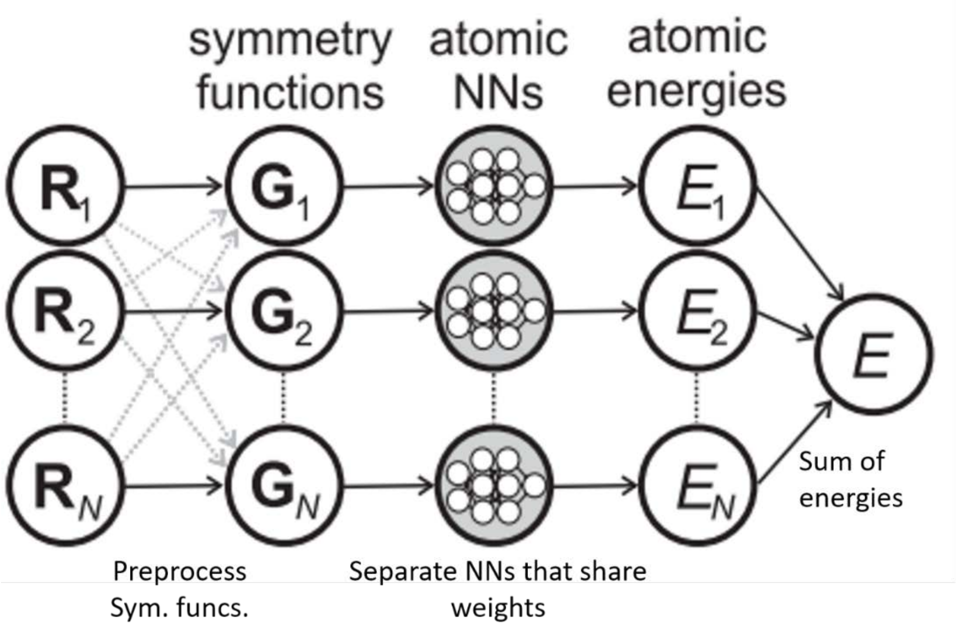


Fig 1. The NN structure described by J. Behler[1, 2]. The initial atomic coordinates R are reexpressed as vectors of symmetry functions G, one for each atom. Each vector is fed into an atomic NN to predict the energy of that atom. These are then summed to get the total energy. Diagram from [4].

system. The total energy is what is used to calculate loss and gradients. This structure is shown conceptually in Fig 1.

Atomic NNs representing the same atom species share weights and are updated each iteration as one. In this way a single sample of 50 atoms will be trained on as if it is 50 samples of single atoms in unique environments, both reducing the dimensionality of the NN input and increasing the effective number of training samples. Furthermore, the structure avoids overfitting to the training data by restricting the NN to treat all atoms of the same species equally, as well as reducing the number of trained parameters of the NN. For the purpose of this paper only a single atom species was used for each dataset, so the entire NN can be thoughts of as many applications of a single atomic NN.

3.2 Symmetry Functions

The vector expression of atomic environment occurs through symmetry functions. Symmetry functions are nonlinear differentiable functions of interatomic distances and angles that represent an atom’s local environment in a form invariant under symmetries of the system. Such symmetries include total system shifts, rotation, index swaps, and other transformations that do not affect total system energy. The form of the symmetry functions used in this paper is the same as that of [2], and displayed in Fig 2 for clarity.

Symmetry functions are nonlinear in both their parameters as well as in input distances and angles, so multiple of the same type of symmetry function with different parameters can provide a more complex description of atomic environment. Though not an orthogonal nor complete representation of the system, symmetry functions as NN input have fared better than either raw atomic coordinates or interatomic distances and angles in our experiments. The symmetry functions used here all include a cutoff function. This ensures that they represent only local atomic environment, which is the primary contribution to the atom’s individual energy. This also makes the computational complexity of calculating symmetry

$$G_i^1 = \sum_j f_c\left(R_{ij}\right). \qquad G_i^2 = \sum_j e^{-\eta\left(R_{ij}-R_s\right)^2} \cdot f_c\left(R_{ij}\right). \qquad G_i^3 = \sum_j \cos\left(\kappa R_{ij}\right) \cdot f_c\left(R_{ij}\right)$$

$$\begin{aligned} G_i^4 &= 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all}} \left(1+\lambda\cos\theta_{ijk}\right)^{\zeta} \cdot e^{-\eta\left(R_{ij}^2+R_{ik}^2+R_{jk}^2\right)} \\ &\quad \cdot f_c\left(R_{ij}\right) \cdot f_c\left(R_{ik}\right) \cdot f_c\left(R_{jk}\right), \end{aligned} \qquad \begin{aligned} G_i^5 &= 2^{1-\zeta} \sum_{j,k\neq i}^{\text{all}} \left(1+\lambda\cos\theta_{ijk}\right)^{\zeta} \cdot e^{-\eta\left(R_{ij}^2+R_{ik}^2\right)} \\ &\quad \cdot f_c\left(R_{ij}\right) \cdot f_c\left(R_{ik}\right). \end{aligned}$$

$$f_c\left(R_{ij}\right) = \begin{cases} 0.5 \cdot \left[\cos\left(\frac{\pi R_{ij}}{R_c}\right)+1\right] & \text{for} \quad R_{ij} \leq R_c \\ 0 & \text{for} \quad R_{ij} > R_c. \end{cases}$$

Fig 2. Symmetry functions used for this problem. Gi means that the function is representing atom i, Rij refers to the interatomic distance between atoms i and j, and Ɵijk refers to the interatomic angle between atoms i, j, and k. The cutoff function fc is not itself a symmetry function, but ensures that they represent *local* atomic environment and reduces overall complexity. Equations from Behler, 2011 [4].

function values scale only with atom density of the system rather than total size, allowing efficient featurization of large systems.

3.3 Implementation

We implemented the atomic NN structure using the Keras [12] wrapper for the Theano [13] deep learning Python library. The model used is shown in Fig 3. This model was implemented as a form of 1D convolutional NN, with atom index as the dimension and each symmetry function input on a separate input channel. The NN used width 1 convolution filters with tanh activations functions, as opposed to having separate atomic NNs constrained to share weights. This allowed the NN to utilize GPU optimization for convolution via the CuDNN package [16], leading to a large speedup in training and prediction. As the atom positions along the 1D input have no spatial significance, using more than width 1 convolutional filters would only lead to potential overfitting to training data. The number of convolutional layers used in our tests ranged from 2 to 4 with varying widths (in this context, number of filters). The final layer is a fully connected layer with weights 1 and bias 0 which effectively sums the atomic energies into a single total system energy.

Rather than inputting preprocessed symmetry function vectors, our model uses a symmetry function layer, which calculates them efficiently using tensor algebra and outputs a vector of values for each atom. The values are then batch normalized before being trained on. Similar to the convolutional layers, the symmetry function layer inputs 1D inputs with multiple channels, where in this case each channel corresponds to a different interatomic distance or angle. Since angles occur between triplets of atoms, the interatomic angle input takes in 2 dimensions of channels. Though the layer could be removed to allow training on preprocessed symmetry functions, it was kept in order to speed up the energy minimization step, which requires the symmetry function values to be recalculated many times.

The symmetry function layer is the most complicated part of the NN. Since we used 5 different types of symmetry function for this project, the symmetry function layer could not be a single operation. Instead, the layer stores a generic form of each symmetry function, as well a set of tensors, each representing a single parameter belonging to a single function. The generic function forms take these parameter tensors as well as the interatomic distances and angles as input. Whereas symmetry function parameters are

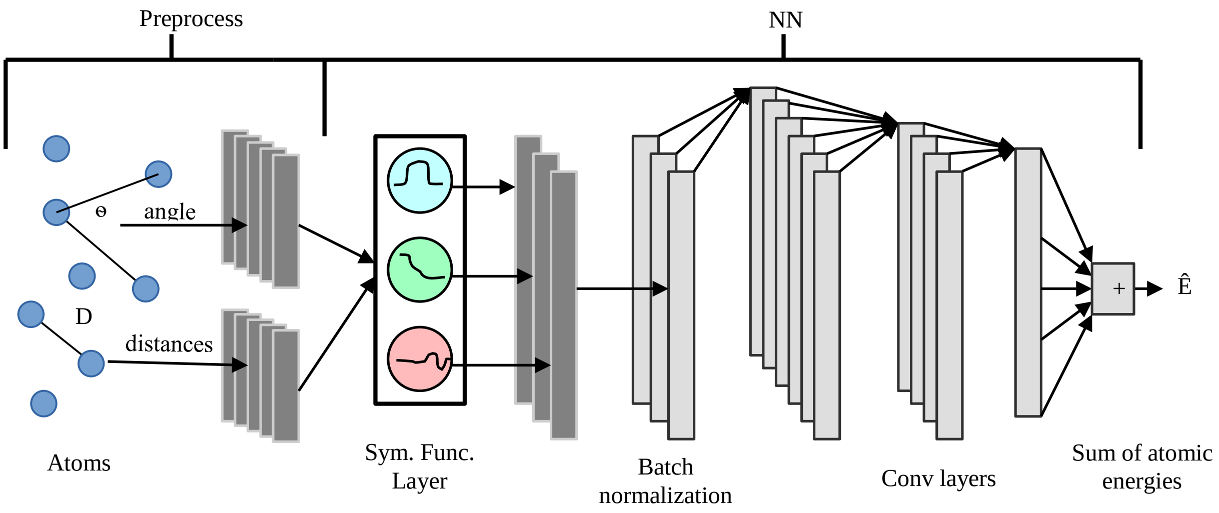


Fig 3. The NN structure used for this project. Vertical bars represent 1D sets of vectors, where each bar is a different channel.

normally taken to be hyperparameters and held constant for the NN, this implementation allows them to be backpropogated to and trained in the same way that weights are.

4. Training Symmetry Functions

Normally, symmetry function parameters are be hand-picked and treated as hyperparameters. As this is unlikely to be optimal, we experimented with optimizing symmetry function parameters using this symmetry function layer.

The dataset used for this part of the project was composed of previously generated configurations of 54 hydrogen atoms. An initial 300 samples were produced using QMC as part of [11], then molecular dynamics simulations run at various temperatures and pressures using Aenet to produce 8 sets of around 3000 samples each. The large number of atoms in each sample is representative of the sort of system in which symmetry functions are most important for the neural network.

When training neural networks, 10% of the training data was set aside as a validation set. Our model was able to beat Aenet on most of the hydrogen datasets when symmetry function parameter was taken to be constant. The validation mean squared error (MSE) results are shown in table 1. The training errors were similar – neither neural network overfit significantly more or less than the other.

Validation MSE in eV across datasets		
dataset	Aenet	My baseline
1	0.007309956	0.09202399804
2	0.01832676	0.01052646236
3	0.01914833	0.01644411056
4	0.01831	0.01276252275
5	0.01335939	0.004068367951
6	0.01230004	0.01672198484
7	0.02461408	0.01878864564
8	0.02553824	0.01870936943
Avg	0.0173633495	0.0237556827
Avg w/o set 1	0.01879954857	0.01400306622
median	0.01831838	0.0165830477

Table 1: MSE in eV of Aenet and our NN without trained symmetry functions on the Hydrogen datasets. Our NN did better on the majority of the datasets but very poorly on the first due to an implementation issue with the data.

The triple sum over angles needed to calculate symmetry functions G^4 and G^5 made these functions exceedingly inefficient to train, to the point where we were unable to fully train them. We were only able to gather data for training the parameters of the first 3 types of symmetry functions. While our calculations with Aenet and untrained symmetry functions all used a set of 36 symmetry functions of types G^2 and G^5, the neural networks with trained functions instead used a set of 24 from G^1, G^2, and G^3.

Since the model with trained symmetry functions was not able to incorporate and angle information from the dataset, its accuracy was worse than our baseline model. However, trained symmetry functions performed better than untrained symmetry functions when given the same initial function types and parameters. Allowing symmetry functions to be trained rather than fixed led to lower MSE, as well as faster convergence and less overfitting. For good initialization, trained symmetry functions achieved a training set MSE of around 10% lower than untrained symmetry functions, with larger decrease on the validation set. However, when initialization or hyperparameters were worse, training symmetry functions led to a larger decrease in MSE. Interestingly, enabling the training of symmetry functions had purely beneficial effects across all cases it was tested. This is as opposed to adding an extra fully connected NN layer, which can often make convergence more difficult or increase overfitting. The fact that we were able to increase the number of trainable parameters while decreasing overfitting implies that the NN is in fact learning a better representation of local atomic environment, as was the goal of training symmetry function parameters. Training symmetry function parameters was slow, and was only implemented on a single hydrogen dataset. Plots of the training on this dataset showing the faster convergence and less overfitting are shown in Fig 4.

Interpretation of what the symmetry function layer learned during training is more difficult. Learned parameters tended to fall into one of two categories. While most parameters stayed very close to their initial value, some drifted far and ended up close to other initial values provided for that parameter, sometimes skipping over intermediate initial values along the way. This implies that training the NN weights creates sorts of local minima in loss near the initial parameter values, causing what parameters

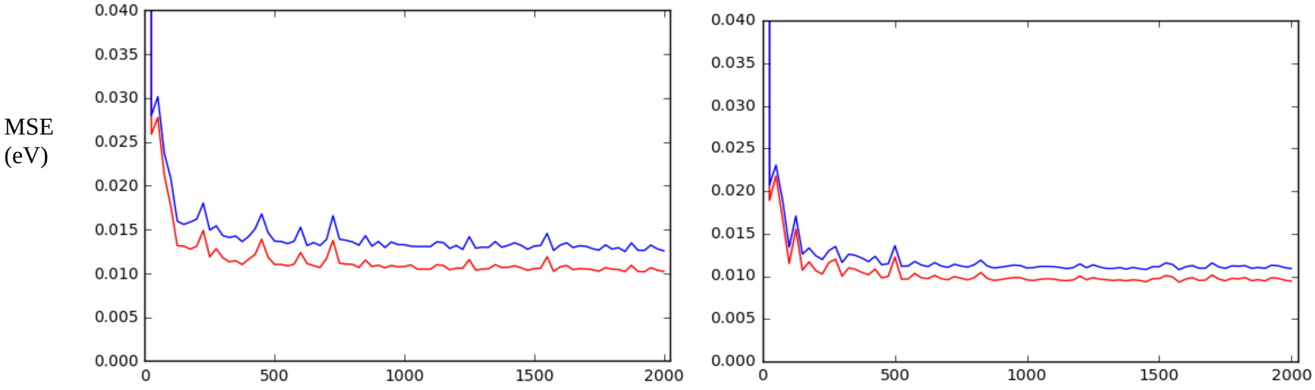


Fig 4: Loss during training for a NN with static symmetry function parameters (left) and with trainable symmetry function parameters (right). Training MSE (red) and validation MSE (blue) are measured in eV. We used 8 each of G^1, G^2, and G^3 for this.

that do escape their own local minima to eventually fall into another one. Having multiple symmetry functions with similar parameters is not inherently harmful, but it is presumably less useful than having a wider variety of symmetry functions would be. Ultimately, NNs with trained symmetry functions restricted to distance were not able to exceed the accuracy of those incorporating angular dependence, so the NNs used for minimization did not incorporate symmetry functions training.

5. Solving for Minimum Energy Structures

Once an NN is trained on DFT data, it can be used as a function to minimize to find the minimum energy structure. We carried out this minimization using the SciPy optimize library, with the lowest energy sample configuration taken as a starting point. Ideally this minimization would use energy gradient information calculated analytically from the NN, but we were unable to extract the gradient from the Keras NN structure. Instead, the optimization used a numerically calculated gradient. This led to numerical limitations in flat areas, which it is possible affected the final results of minimization.

In order to simplify the PES being minimized over, we used different datasets for this part of the project. For this part, we generated new 2, 3, 4, and 5 atom datasets using DFT through the QuantumEspresso Pwscf package on the Bridges supercomputer [14, 15]. The configurations were generated by molecular dynamics simulations run at 1000 F in 3D boxes with periodic boundary conditions. Each time step was taken to be a new configuration. In this way the configurations used to train the NN were representative of the configurations the atoms could potentially take in reality. The alternative method of generating data is randomly generating atomic configurations, which when tested caused the NN to try to fit unrealistic energies and harmed its overall accuracy. The energies calculated for each configuration are total energies of that specific time step assuming stationary atoms. That is, they include the potential and kinetic energy of all relevant electrons and potential energy of nuclei, but not the kinetic energy of nuclei.

Prior to training, the data is preprocessed by calculating interatomic distances and angles from coordinates. In the Aenet model, symmetry function values are also preprocessed since they are taken as constant for a dataset. That was unnecessary for the NN used for this project due to the NN's included symmetry function layer. Configurations with energy two standard deviations above the mean were also removed from the training and validation datasets during preprocessing. These points tended to correspond to the first few time steps of molecular dynamics, where the atoms often began in unrealistic positions (such as too close together) that they quickly were forced out of.

There are a number of ways to evaluate the results, including measuring NN accuracy directly, qualitatively looking at the PES generated by the NN, and evaluating the results of the final minimization process.

5.1 NN accuracy

The mean squared error (MSE) of the NN used in this paper on the 3, 4, and 5 atom datasets was compared to that of Aenet, which was taken to be the baseline. The results are shown in Table 2. The NN used in this paper outperformed Aenet, even with the same numbers of layer and nodes. The reason for this is unclear since the two NN models are fundamentally the same – it is possible the difference is due to difference in optimizer or even normalization. While Aenet uses the L-BFGS-D [8] optimizer, the NN used in this paper was trained with RMSprop. It is also possible that representing the atomic NN as a convolutional NN structure increased the accuracy, potentially because NN packages are well trained to work with convolutional NNs. Indeed, we noticed a large improvement in NN error after switching from separate atomic NNs to a convolutional NN in our own model.

Comparing the accuracy of NNs between papers is difficult, as different systems will lead to different distributions of energy data, and different final NN errors. That being said, the NN used here has lower error than nearly all of the NNs used for similar problems in [1]. A large part of this difference is likely due to the fact that the datasets used here contained only a single atom species, while those listed in [1] generally have at least 2 atom species.

5.2 PES visualization

The PES predicted by a NN can be thought of as a surface in n-dimensional space. In a 2 atom system, the only coordinate that matters is interatomic distance. This makes the PES easy to visualize, since it is a function of a single variable. Such as constructed PES for a 2 atom system is plotted in Fig 5a. For larger numbers of atoms the PES becomes a function of multiple variables, making it more difficult to visualize.

		MSE (eV)					
		3 atom		4 atom		5 atom	
NN structure		train	val	train	val	train	val
15t-15t	This NN	1.13E-04	1.30E-04	5.67E-04	8.30E-04	6.10E-04	7.31E-04
	Aenet	1.76E-04	2.50E-04	8.36E-04	1.20E-03	1.56E-03	2.33E-03
30t-30t	This NN	9.94E-05	1.19E-04	3.76E-04	4.13E-04	4.28E-04	6.28E-04
	Aenet	1.85E-03	2.45E-03	6.37E-04	1.15E-03	1.51E-03	2.40E-03
40t-20t-20t-10t	This NN	6.49E-05	7.15E-05	1.07E-04	4.92E-04	2.23E-04	4.49E-04
	Aenet	3.58E-04	4.15E-04	1.81E-03	2.21E-03	1.65E-03	2.26E-03

Table 2. A comparison of the NN used in this paper and Aenet for various structures and datasets. The structure is represented as layers with dashes between them. The number for each layer is the number of nodes in it, while the t refers to a tanh activation function.

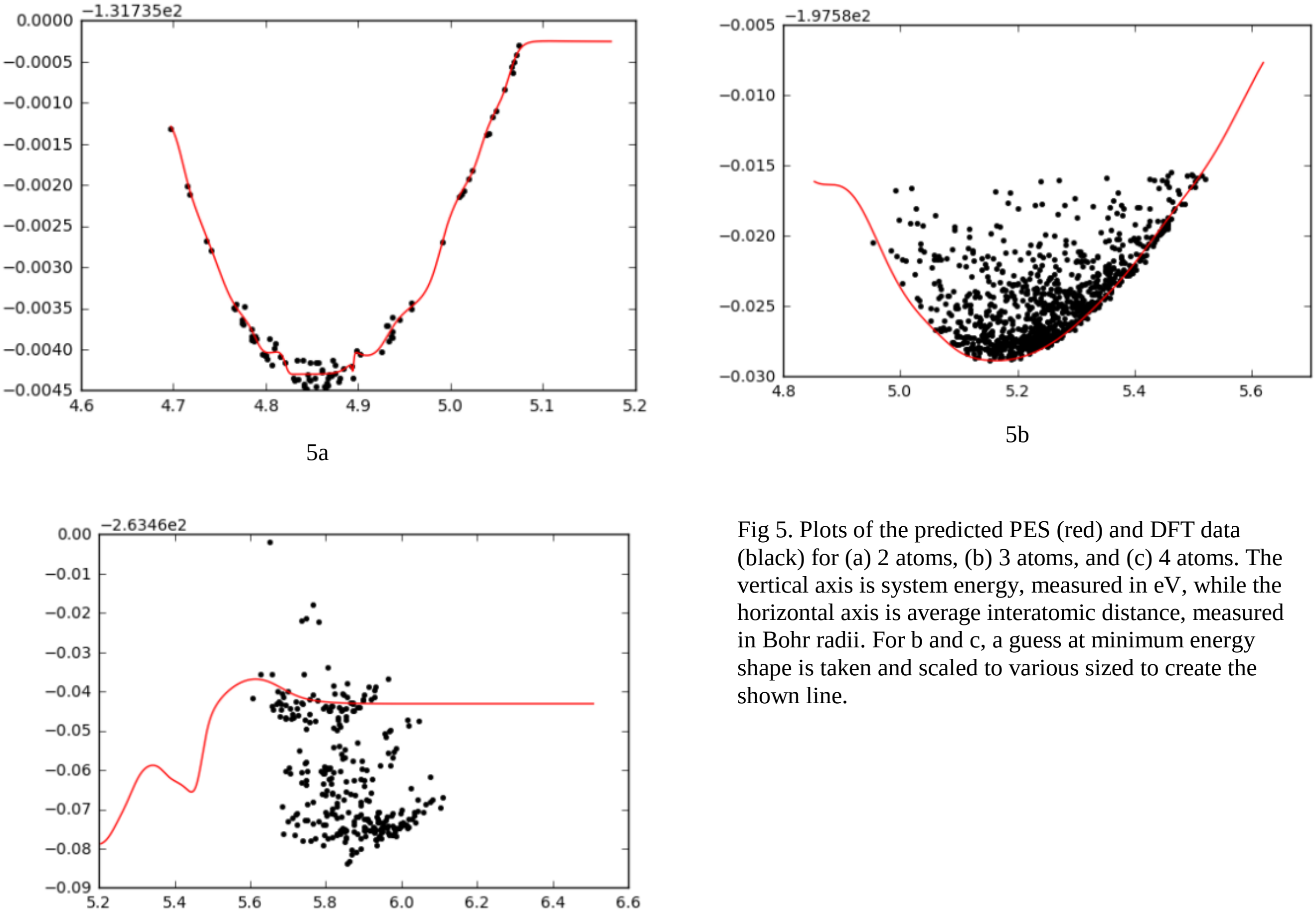
If one chooses a shape for the atoms to take, the PES can be plotted for that shape at various sizes. This is done for a 3 atom equilateral triangle in Fig 5b and a 4 atom tetrahedron in Fig 5c.

The 2 atom PES can be seen to clearly fit the DFT data, though the data's own noise complicates the fit. The reason for the noise is unclear – the problem did not occur in other systems. The 3 atom PES seems to lower bound the DFT data – presumably this is because the equilateral triangle is near the minimum energy shape at any size, making it the lower bound of energy at each size. In the 4 atom system, one gets the impression that the tetrahedron is not the minimum energy shape at all. In this case and in higher dimensions, attempting to visualize the PES becomes less fruitful.

5.3 Minimization results

In order to evaluate the NN PES minimization, the configuration predicted by the minimization process were used as input for a "relax" DFT calculation in QuantumEspresso, which first calculates the energy of those configurations using DFT, then minimizes from there to return the true (to DFT) minimum energy of the system. The results are shown in Table 3.

It would be unrealistic to expect the NN to generate the exact same minimum energy configuration as DFT, but from the results one can see that the NN predicted and DFT predicted configurations are in fact quite close energetically, especially in the 3 atom case, in which the energy difference is around 3E-6 eV. It is also worth noting that the true energy of all predicted configurations is below the lowest energy that



5a

5b

5c

Fig 5. Plots of the predicted PES (red) and DFT data (black) for (a) 2 atoms, (b) 3 atoms, and (c) 4 atoms. The vertical axis is system energy, measured in eV, while the horizontal axis is average interatomic distance, measured in Bohr radii. For b and c, a guess at minimum energy shape is taken and scaled to various sized to create the shown line.

the NN was trained on. This means that the NN has in fact improved upon the DFT data on which it was trained – a task that would not be possible with linear interpolation, for example. This is promising, because it implies that if the NN were then retrained on DFT data closer to the minima, it would then be able to further improve on its results. Our previous tests have shown that such an iterative method does tend to yield improving results each iteration on simpler datasets. If QMC data were used for this rather than DFT, this method could prove a more accurate way of calculating the minimum energy configuration than is currently possible.

6 Conclusion and Future Directions

It has only been applied to homonuclear configurations so far, but the NN used in this paper has shown promising results both in terms of predictive accuracy and for minimization. Though we were unable to train the most useful symmetry functions for this application, it is possible other applications will require different symmetry functions, potentially ones that are better suited for training. The next step in this project will be to train the NN on multiple species of atoms. In particular, the H_2O-OH- is of interest because we will be able to compare the minimization results of our NN to those in [3]. Should the NN minimization process be able to outperform these results, it will be a step forward in the field of solving minimum energy configurations.

	Energy (eV)		
	3 atom	4 atom	5 atom
Lowest energy in training data	-197.60435	-263.53322	-329.45373
Minimum energy predicted by NN	-197.61155	-263.57965	-329.48131
DFT energy of NN predicted minima	-197.609017	-263.53796	-329.45888
DFT energy of DFT predicted minima	-197.60902	-263.55450	-329.49146

Table 3. Energy values associated with the lowest energy configuration in the training data, the lowest energy predicted possible by the NN, the true energy of the NN-predicted minima, and the true energy of the true minimal energy configuration.

Bibliography

1. J. Behler, "Neural network potential-energy surfaces in chemistry: a tool for large-scale simulations" *Phys. Chem. Chem. Phys.* 2011, http://pubs.rsc.org/en/content/articlehtml/2011/cp/c1cp21668f

2. Moravietz et. al., "How van der Waals interactions determine the unique properties of water" *PNAS* 2016

3. L. Wagner, J. Grossman, "Quantum Monte Carlo Calculations for Minimum Energy Structures" *Phys. Rev. Lett.* 2010, http://journals.aps.org/prl/abstract/10.1103/PhysRevLett.104.210201

4. J. Behler, "Atom-centered symmetry functions for constructing high-dimensional neural network potentials" *J. Chem. Phys.* 2011

5. N. Artrith and A. Urban, *Comput. Mater. Sci.* 114 (2016) 135-150

6. J. Behler and M. Parrinello, *Phys. Rev. Lett*. 98 (2007) 146401

7. J. Behler, *J. Chem. Phys.* 134 (2011) 074106

8. R. H. Byrd, P. Lu and J. Nocedal, *SIAM J. Sci. Stat. Comp*. 16 (1995) 1190-1208

9. P. Giannozzi, et al *J.Phys.:Condens.Matter*, 21, 395502 (2009) http://dx.doi.org/10.1088/0953-8984/21/39/395502

10. We used the Au.blyp-d-hgh.UPF pseudopotential from www.quantum-espresso.org for DFT calculations

11. N. Tubman et. al. *Phys. Rev. Lett.* 115 (2015) 045301 https://doi.org/10.1103/PhysRevLett.115.045301

12. Francois Chollet, Keras, 2015, Github Repository: https://github.com/fchollet/keras

13. Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions"

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. Specifically, it used the Bridges system, which is supported by NSF award number ACI-1445606, at the Pittsburgh Supercomputing Center (PSC)."

14. Towns, J., et. al. 2014. XSEDE: Accelerating Scientific Discovery. Computing in Science & Engineering. 16(5):62-74. http://doi.ieeecomputersociety.org/10.1109/MCSE.2014.80

15. Nystrom, N. A., et. al. 2015. *Bridges: A Uniquely Flexible HPC Resource for New Communities and Data Analytics*. In Proceedings of the 2015 Annual Conference on Extreme Science and Engineering Discovery Environment (St. Louis, MO, July 26-30, 2015). XSEDE15. ACM, New York, NY, USA. http://dx.doi.org/10.1145/2792745.2792775

16. S. Chetlur, et. al. "cuDNN: Efficient Primitives for Deep Learning", arXiv:1410.0759