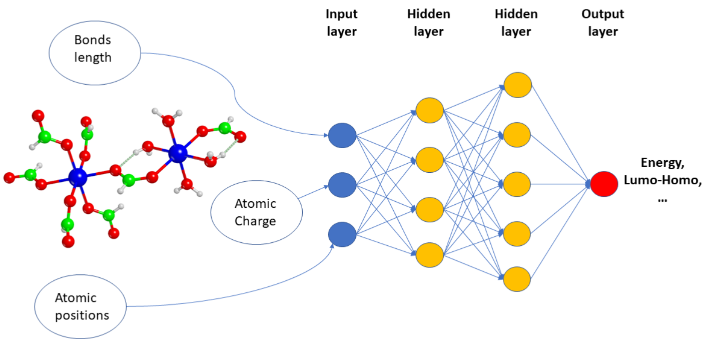
Neural Networks in Quantum Chemistry



Joseph Sleiman & Scott le Roux

The current standard for solving quantum chemical problems scales badly with increasing molecular size. Thus, Neural Networks are proposed as an alternative solution to determine the structure-property relationships of molecules accurately and more efficiently. Using *TensorFlow* and the *DScribe* molecular descriptor library, we implemented Neural Networks that predicted the internal energies of organic molecules in the QM9 dataset, achieving a mean squared error of 0.002 Hartree^2. During descriptor creation and Neural Network training, we found that GPUs achieved significant speedups over single CPU computations, but were slower than parallel CPU computations using 8 cores or more.

Machine Learning (ML) and Artificial Intelligence (AI) have provided comprehensive solutions to various natural and scientific phenomena, ranging from galaxy detection to genome sequencing, and paired with the boom in data and growing computational power, means the applications for ML and AI will continue to stretch further and wider into even the most established and niche sectors.

Likewise, quantum chemistry - a branch of chemistry focused on predicting chemical and physical properties of molecules and materials[1] - has seen an increase in ML applications, particularly through the use of Neural Networks (NNs). This Deep learning (DL) architecture consists of groups of "neurons" (a term inspired by the architecture and functionality of the brain) all densely connected to form a network where the input is typically a vector consisting of your data representation and the output is a value representing your prediction. In between these layers, there are hidden layers consisting of an arbitrary number of neurons optimised by training weights to produce the best predictions possible on the training datasets, whilst also generalising to the new data we wish to make predictions about.

Accordingly, in this project we use NNs to predict quantum mechanical properties of molecules, instead of the traditional method of Density functional Theory (DFT), which uses approximations to the Schrödinger equation. This is a very promising avenue considering that it gets increasingly difficult and computationally expensive to solve the Schrödinger equation for molecules with dozens of atoms or more. Hence, the numerical optimisation approach inherent in ML could result in accurate predictions without the need for large computing power and a lot of time! This is possible because NNs can approximate any function when combined with multiple hidden layers and neurons with nonlinear activation functions.

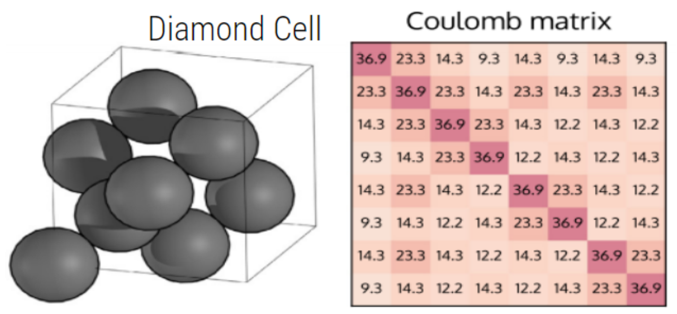
How Did We Replace the Schrödinger Equation?

The main issue with using ML to solve quantum chemical problems is that chemical data does not come in a desirable form readable by NNs. To bridge this, we used “molecular descriptors” which transform raw chemical data into feature vectors which preserve specific chemical properties. These descriptors were implemented via the Python library DScribe[2] which provides molecular descriptors for use in ML applications. More specifically, we used two global descriptors - the Coulomb Matrix (CM) and Many-Body Tensor Representation (MBTR) - and two local descriptors - Smooth Overlap of Atomic Positions (SOAP) and Atom-Centred Symmetry Functions (ACSF) - throughout our project in order to output property predictions such as the internal energy of a molecule at zero Kelvin (0K), and the charges on each atom making up the molecule.

For example, the CM is an $N \times N$ matrix where N is the number of atoms in the molecule. The diagonal elements represent the self-interactions of the i^{th} atom, and the off-diagonal entries represent the coulombic repulsion between the i^{th} and j^{th} nuclei. This produces a compact matrix (see Figure 1) that encodes the positions and electrostatic interactions between all the atoms in the molecule (refer to this link for more details on all the descriptors used).

In order to attain sufficiently accurate NN models, we require a lot of data during training. Hence, we utilised the QM9 dataset[3] which contains 134,000 organic molecules with their corresponding energetic and thermodynamic properties calculated very thoroughly and laboriously via traditional quantum chemical analytical methods. This dataset is readily accessible online and is thus used as a benchmark for developing new quantum chemical tools, as well as for comparative purposes when publishing new research. In terms of data handling, we sorted the QM9 dataset in increasing value of internal energy at 0K, and then reserved every fifth data point for the test set, ensuring that the test was representative of a range of internal energies. The remaining data was randomly separated into an 80:20 splitting of training and validation sets, respectively. The end goal of the project is a supervised learning task, in which we train the algorithm on a labelled dataset (QM9) in order to predict a final output label (internal energy at 0K); this is the standard regression or classification ML procedure.

We implemented our NNs via the DL Python library *TensorFlow* with *Keras* 2.0 back-end, and used the software package KerasTuner to optimise the hyperparameters of the network in order to minimise the mean squared error (MSE) of our predicted results versus the true values given by QM9. These hyperparameters included the learning rate, the batch size, activation function, weight initialisation, as well as the number of neurons and hidden layers making up the network topology. This approach uses either an iterative random-search optimisation or a Bayesian optimisation that minimises your desired loss function (MSE, in our case) in a systematic manner, as opposed to a very long and laborious brute-force method of optimising the NN hyperparameters. This task was completed separately for each molecular descriptor because there is no universally optimal NN architecture for any given problem (see “No Free Lunch”[4] theorem), and each descriptor has a different number of features (see Table 1), altering the number of neurons needed in the input layer.



36.9	23.3	14.3	9.3	14.3	9.3	14.3	9.3
23.3	36.9	23.3	14.3	23.3	14.3	23.3	14.3
14.3	23.3	36.9	23.3	14.3	12.2	14.3	12.2
9.3	14.3	23.3	36.9	12.2	14.3	12.2	14.3
14.3	23.3	14.3	12.2	36.9	23.3	14.3	12.2
9.3	14.3	12.2	14.3	23.3	36.9	12.2	14.3
14.3	23.3	14.3	12.2	14.3	12.2	36.9	23.3
9.3	14.3	12.2	14.3	12.2	14.3	23.3	36.9

Figure 1: Conversion of the positional and electrostatic information of a diamond cell into its Coulomb Matrix form via *DScribe*.[2]

Finally, for the High-Performance Computing (HPC) part of the project, we benchmarked the speeds of both descriptor creation and NN training using GPUs and CPUs, parallelising the latter with multiple cores. This was achieved using the remote server computing node provided by our host site at the Slovak Academy of Sciences, where we were given access to two Intel Xeon CPUs and an NVIDIA Tesla K20m GPU.

Neural Network Model Results and Benchmarking

Optimization of NNs is the heavy lifting of the project; in order to save a lot of time, it is imperative that educated guesses are made concerning what hyperparameter options to test before running the tuning via *KerasTuner*. Batch size and learning rate took the accepted standard values of $32, 64, 128$ and $.01, .001, .0001$, respectively. For weight initialisation, we tested *He Uniform* and *Xavier Uniform* initialisers, which are considered the state-of-the-art and either is compatible with the majority of activation functions. The activation functions we tested were the *rectified Linear Unit* (ReLU), *exponential Linear Unit* (ELU), *softplus*, and *shifted softplus* (ssp),[5] described by the following equations:

$$ReLU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

$$ELU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ e^x - 1, & \text{if } x < 0 \end{cases}$$

$$softplus(x) = \ln\left(e^x + 1\right)$$

$$ssp(x) = \ln(0.5e^x + 0.5)$$

Finally, we tested different NN topologies selecting from between 1-4 hidden layers each with a variable number of hidden neurons from the set, $\{32, 64, 128, 256\}$.

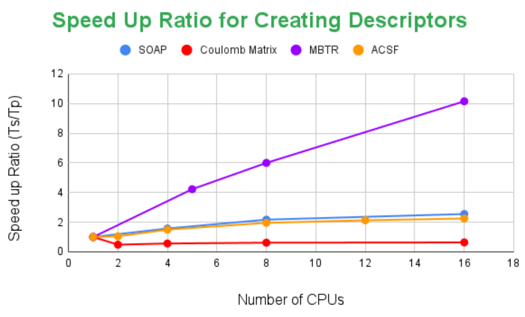
In Table 1, we see the optimised NN structures for each molecular descriptor and their corresponding predictive accuracy. While the best hyperparameters varied across descriptors, a batch size of 32 and an *ADAM* optimiser were universally optimal. To summarise, the CM achieves the best MSE by an order of magnitude over the second best descriptor, MBTR. The latter was a further order of magnitude more accurate than the local descriptors ACSF and SOAP.

In terms of descriptor creation, as we can see from Figure 2, the speed up ratio, $R = \frac{t_{serial}}{t_{parallel}}$, for complex descriptors with thousands of features (ACSF, SOAP, MBTR) consistently exceeds 2 and reached a maximum of $\sim$10 for 16 CPU cores on MBTR. This highlights the importance of parallelising the creation of descriptors with 3000 features or more. Conversely, in the case of simpler descriptors like the CM, invoking parallel implementation produces an overhead greater than the eventual speed up achieved using multiple cores. Hence

Table 1: Summary of Optimised Molecular Descriptors and their Internal Energy Prediction Mean Squared Error (MSE)

Descriptor	# Features	Learning Rate	Activation Function	Hidden Layer Topology	Weight Initialiser	MSE
CM	841	0.0001	*softplus*	[256]	*Xavier Uniform*	0.002
MBTR	9500	0.001	*ELU*	[64,64,32,128]	*Xavier Uniform*	0.01
SOAP	4920	0.001	*ELU*	[256,32,128]	*He Uniform*	0.241
ACSF	3190	0.0001	*softplus*	[128,32,16]	*He Uniform*	0.521

with $R < 1$, it is not efficient to parallelise in this scenario.



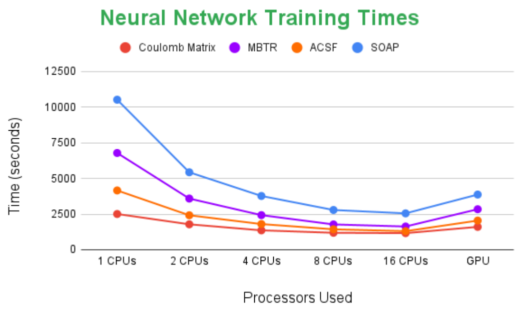


Figure 2: Graphs for computation times

For NN training, we found that there is an inverse relationship between time and number of CPU cores used in the training process, as seen in Figure 2. This is an expected result and highlights that one should aim to utilise multiple cores in order to achieve the best performance. It is also clear that a single GPU core outperforms a single CPU core by $\sim$ 2x. However, if we use 8 CPU cores or more, the training process is consistently faster than a single GPU.

Final Thoughts

Ultimately, the CM was the best molecular descriptor in capturing the salient features of molecules. This is a surprising yet satisfying result when one considers that the CM is the simplest descriptor with the fewest features by a significant margin (see Table 1) and is therefore the most computationally efficient. It was also expected that descriptors that focus on global features like the CM and MBTR would produce the best predictions on global quantities like the internal energy of a molecule at 0K and this proved to be the case.

On the HPC front, as expected, GPUs provided significant speedup to single CPU computation. Reducing computation time during NN training is extremely important and the improvements in hardware such as GPUs and more recently Tensor Processing Units (TPUs) are the primary reason that DL is a leading method in modern computing. These improvements have allowed researchers and programmers to effectively use large DL models to solve problems which previously took many years, in only a few hours.

However, contrary to general computing consensus, we found that parallelised CPU cores were faster than even the GPU speeds. We speculate that this unexpected result was either due to a malfunction with the NVIDIA GPU used, or perhaps *TensorFlow* has optimized GPU calculations for newer GPU hardware. In future work, we hope to solve this GPU issue and validate the widely accepted fact, that GPUs are superior execution platform for NNs to CPUs.

Additionally, we hope to use NNs with a multi-neuron output layer in order to predict other molecular properties such as the charge or spin of each atom that makes up the molecule and we postulate that descriptors like ACSF, focusing on local molecular features, will produce the best results.

References

[1] T. A. Profitt and J. K. Pearson, “A shared-weight neural network architecture for predicting molecular properties,” *Phys. Chem. Chem. Phys.*, vol. 21, pp. 26175–26183, 2019.

[2] L. Himanen, M. O. Jäger, E. V. Morooka, F. Federici Canova, Y. S. Ranawat, D. Z. Gao, P. Rinke, and A. S. Foster, “Dscribe: Library of descriptors for machine learning in materials science,” *Computer Physics Communications*, vol. 247, p. 106949, 2020.

[3] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, “Quantum chemistry structures and properties of 134 kilo molecules,” *Scientific Data*, vol. 1, 2014.

[4] D. Wolpert and W. Macready, “No free lunch theorems for optimization,” *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[5] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller, “Schnet: A continuous-filter convolutional neural network for modeling quantum interactions,” 2017.

PRACE SoHPC Project Title
Neural Networks in Quantum Chemistry

PRACE SoHPC Site
Slovak University of Technology, Bratislava, Slovakia

PRACE SoHPC Authors
Scott le Roux, Trinity College Dublin, Ireland
Joseph Sleiman, University of Edinburgh, UK

PRACE SoHPC Mentor
Marian Gall, Slovak University of Technology, Bratislava, Slovakia

PRACE SoHPC Contact
Scott le Roux, Trinity College Dublin
E-mail: sleroux@tcd.ie
LinkedIn: scottleroux
Joseph Sleiman, University of Edinburgh
E-mail: s1766037@sms.ed.ac.uk
LinkedIn: josephsleiman

Scott le Roux

Joseph Sleiman

PRACE SoHPC Software Applied
Python, TensorFlow, DScribe, Keras-Tuner

PRACE SoHPC More Information
Joseph's PRACE GitHub Repository
Scott's PRACE GitHub Repository

PRACE SoHPC Acknowledgment
Special thanks to Marián Gall for his support and guidance during this project, to everyone at PRACE for the opportunity, and the team at ICHEC for their insightful training week programme.

PRACE SoHPC Project ID
2107