2017

# Application of Artificial Neural Networks to Identify Equilibration in Computer Simulations

Michael H. Leibowitz
*Boise State University*

Evan D. Miller
*Boise State University*

Michael M. Henry
*Boise State University*

Eric Jankowski
*Boise State University*

## Publication Information

Leibowitz, Michael H.; Miller, Evan D.; Henry, Michael M.; and Jankowski, Eric. (2017). "Application of Artificial Neural Networks to Identify Equilibration in Computer Simulations". *Journal of Physics: Conference Series, 921*, 012013-1 - 012013-8. https://doi.org/10.1088/1742-6596/921/1/012013

IOP Conf. Series: Journal of Physics: Conf. Series **921** (2017) 012013 doi:10.1088/1742-6596/921/1/012013

# Application of artificial neural networks to identify equilibration in computer simulations

**Mitchell H Leibowitz, Evan D Miller, Michael M Henry, Eric Jankowski**

Micron School of Materials Science and Engineering, Boise State University, 1910 University Dr., Boise, Idaho 83725, United States

E-mail: ericjankowski@boisestate.edu

**Abstract.** Determining which microstates generated by a thermodynamic simulation are representative of the ensemble for which sampling is desired is a ubiquitous, underspecified problem. Artificial neural networks are one type of machine learning algorithm that can provide a reproducible way to apply pattern recognition heuristics to underspecified problems. Here we use the open-source TensorFlow machine learning library and apply it to the problem of identifying which hypothetical observation sequences from a computer simulation are "equilibrated" and which are not. We generate training populations and test populations of observation sequences with embedded linear and exponential correlations. We train a two-neuron artificial network to distinguish the correlated and uncorrelated sequences. We find that this simple network is good enough for $> 98\%$ accuracy in identifying exponentially-decaying energy trajectories from molecular simulations.

## 1. Introduction

Computer simulations that model physical systems in equilibrium are tools that have been essential to understanding an enormous variety of phenomena ranging from liquid argon[1], the self-assembly of colloids[2], criticality in spin glasses[3, 4], lipid self-assembly[5], DNA-origami[6], protein folding[7, 8], and the segregation of neighborhoods[9]. For the modeling of material systems, Metropolis Monte Carlo (MC)[10] and molecular dynamics (MD)[11] are the two most popular tools for sampling equilibrium ensembles of microstates. Both techniques generate sequences of microstates and the main cost of performing an MD or MC simulation is waiting for this sequence to converge to one that is representative of the thermodynamic ensemble at equilibrium, and then sampling enough microstates to perform ensemble averages with high precision. Sampling the equilibrium distribution of microstates at a particular thermodynamic state point is challenging because both MD and MC techniques are inefficient at overcoming free energy barriers that separate local minima in the multidimensional configurational landscape[12]. Consequently, the observables for a simulation (e.g. potential energy, order parameters) may appear to converge about a stable average for "long" times before relaxing to another stable average associated with lower free energy (Fig. 1). Every researcher performing MD or MC simulations has encountered this equilibration problem when asking themselves "Have I run my simulation for long enough?"

There are two correct answers to this question that are both unsatisfying. The first answer is "No." It is unsatisfying to know that the only way to be sure that the sampled microstates are

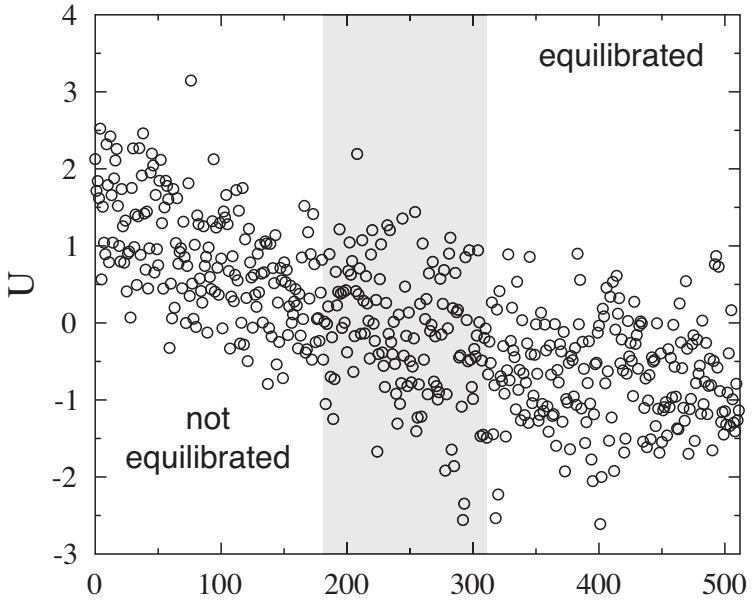


**Figure 1.** An exponential decay to a stable average is a frequently observed trend in thermodynamic simulations. It is difficult to decide where the exponential decay ends, and where the "equilibrated" region begins. Different heuristics could justify equilibrium sampling beginning anywhere in the shaded grey region.

representative of the free energy minimum is to perform a longer simulation that does not relax to a more stable distribution. That is, only in the limit of an infinitely long simulation can one really be sure that is has been run for long enough. The second answer is "Yes, when there are enough measurements to give a sufficiently precise answer." This answer is unsatisfying because it raises a new question "How shall I quantify when there are enough measurements?" and because it ignores the problem of whether the global free energy minimum has been sampled.

Recent advances in sampling schemes and computational hardware help to ameliorate, but not solve, the problem of generating infinitely many microstates. For MD simulations, the application of transition state theory[13] and histogram reweighting[14] has informed acceleration schemes that better sample rough energy landscapes. In MC simulations, enhanced sampling with Wang-Landau algorithms[3], cluster moves[15], and population annealing[16] are among the techniques used to extract more information per microstate. For both MD and MC simulations, accelerated sampling has been enabled with graphics processing units (GPUs) that facilitate single instruction-multiple data (SIMD) parallelization[17, 18].

Here we work towards addressing the second question "How shall I quantify when there are enough measurements." In principle, answering this question can be addressed by performing autocorrelation analysis on the system observables. In practice, measuring the autocorrelation time of system observables relies upon heuristics for what data to ignore or include. Even when the global free energy minimum is achieved after beginning with an initial random guess, which is a best-case scenario, the observed system energy decays exponentially from the initial guess to the correct average. In this case the problem becomes identifying the first microstate $u^*$ at time step $t^*$ after which the sequence of microstates $u_{t>t^*}$ represents the equilibrium distribution, using the distribution of observables such as system energy $U_t$ as a proxy for the distribution of microstates. In order to be reproducible, the process of identifying $u^*$ should be well-defined so that the same identification algorithm can be applied to all of the simulation trajectories generated for a scientific study, and to compare trajectories between studies. The strategy we

use here to construct a well-defined heuristic for identifying $u^*$ is to train an artificial neural network for automating the decision of whether a sequence of observables is representative of an equilibrium distribution or not.

## 2. Methods

### *2.1. Measuring equilibration*

The $N$ microstates sampled during MC and MD simulations represent observations of the underlying, unknown equilibrium distribution. Averages computed from these independent observations should converge to a normal distribution, according to the central limit theorem. Therefore the three key metrics used to determine equilibration are the average $\langle U \rangle = \frac{1}{N}\sum_{i=1}^{N} U_i$, standard deviation $\sigma_U = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(U_i - \langle U \rangle)^2}$, and decorrelation time $\delta t_0$ of a sequence of measurements. The autocorrelation function $a_U(\delta t)$ describes how correlated a sequence of measurements is with itself over a measurement lag $\delta t$ (steps in the case of MC, time in the case of MD), and the decorrelation time $\delta t_0$ where $a_U(\delta t_0) = 0$ describes how long one must wait, on average, for two measurements from the sequence $U_t$ to be decorrelated (independent) of each other. The ubiquity of fast numerical libraries for most programming languages makes computational implementations of the autocorrelation function relatively straightforward through the use of fast Fourier transforms:

$$a_U(\delta t) = \frac{\mathrm{IFT}[\mathrm{FT}[U_t - \langle U \rangle]\mathrm{FT}[U_t - \langle U \rangle]^*]}{N\sigma_U^2} \tag{1}$$

where $\mathrm{FT}[A]$ is the discrete Fourier transform of arbitrary sequence $A$, $\mathrm{IFT}[A]$ is the corresponding inverse Fourier transform, and $\mathrm{FT}[A]^*$ is the complex conjugate of the Fourier transform of $A$. In python, this can be accomplished in a few lines of code:

```
import numpy
FT = numpy.fft.rfft #discrete Fourier transform on real input
IFT = numpy.fft.irfft #inverse discrete Fourier transform on real input
def autocorr1D(array):
    ft = FT(array-numpy.average(array))
    acorr = IFT(ft*numpy.conjugate(ft))/(len(array)*numpy.var(array))
    return acorr[0:len(acorr)//2]
```

where only the first half of the autocorrelation array is returned because of its symmetry with the second half. The above normalization by the variance $\sigma_U^2$ and sample size $N$ sets $a_U(0) = 1$, which indicates the fact that every measurement is perfectly correlated with itself (lag $\delta t = 0$). Common heuristics for determining $\delta t_0$ from $a_U(\delta t)$ include finding the smallest $\delta t$ where $a_U(\delta t) < 0$ (this avoids having to interpolate between values of $a_U(\delta t)$ to find the precise zero), or finding the fist $\delta t$ within some tolerance of 0.

Using $\langle U \rangle$, $\sigma_U^2$, and $\delta t_0$, the process of measuring equilibration can be thought of as finding the largest subset of the sequence $U_t$ that will give "enough" independent samples $N/\delta t_0$, specified by the researcher. Herein lies the challenge of specifying a well-defined heuristic, because it can depend on what one considers to be "enough", the frequency with which $U_t$ is written out by a simulation engine, and the relaxation kinetics for the system being studied, and the details of the scheme used to advance from $U_i$ to $U_{i+1}$. An example of one heuristic is to split $U_t$ into subsets of size $n$, measuring $\langle U \rangle$, $\sigma_U^2$, and $\delta t_0$ for each subset to determine if they are significantly different (e.g. with the student's t-test). The final equilibrated ensemble in this case is determined by concatenating the subsets moving backwards in time until one is found to be significantly different.

### *2.2. Machine Learning*

Another way to think about the problem of measuring equilibrium is to frame it as a pattern recognition problem, for which machine learning algorithms are well-suited[19]. A recent application of deep neural networks to pattern recognition in the board game *go* represents a breakthrough in computational *go* strategy[20]. Neural networks trained to recognize "good" moves were used to simplify the search space of Monte Carlo tree estimates of move quality[21] to beat over fifty professional go players in 2016 and early 2017. In the field of molecular simulation, machine learning techniques have been applied to identify assembly pathways in colloidal systems[2, 22], optimizing coarse-graining strategies[23], structural analysis[24], and design rules for organic photovoltaic ingredients[25]. Here we investigate the automated evaluation of equilibrated simulation trajectories using a single artificial neural network.

An artificial neural network is described by a set of "neurons" $n_i$ and "connections" $c_{i,j}$ between them. Each neuron "fires", generating an output signal that depends on the sum of its input connections, and the output signals can be used as input to more neurons or interpreted as decisions for pattern recognition problems. Artificial neural networks can be "trained" by determining the connection weights that minimize the error for a pattern recognition problem.

Here we specify a single "layer" of a neural network with two neurons. The first neuron is the "equilibrated" neuron and should fire when its input connections correspond to signal from an equilibrated sequence of observations. The second neuron is the "not equilibrated" neuron and should fire when its input connections correspond to signal from a non-equilibrated sequence of observations. Each neuron has one input connection for each observation and only one output connection. We can therefore specify this neural network as a matrix multiplication problem:

$$\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,N} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,N} \end{bmatrix} \cdot \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_N \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \tag{2}$$

where the weights $w_{i,j}$ and biases $b_{i,j}$ are scalars that will be determined through a training process. The weights $w_{1,1\cdots N}$ correspond to the connection weights into the first neuron and $w_{2,1\cdots N}$ for the second neuron. After being trained, the artificial neural net defined by $w_{i,j}$ and $b_i$ can be used to determine if a sequence of $N$ observations is equilibrated by using $U_{1\cdots N}$ as input to Eqn. 2, resulting elements of a decision vector $n_1$ and $n_2$. During training we impose constraints to ensure $n_1 + n_2 = 1$, so the values returned by the artificial neural net can be interpreted as the certainty with which it thinks a data set is equilibrated or not.

### *2.3. Training the Neural Net*

To determine the $w_{i,j}$ and $b_i$ that most accurately distinguish equilibrated from non-equilibrated observation sequences we generate a set of $N_t$ training sequences with equal numbers of equilibrated and non-equilibrated samples. The equilibrated samples are generated using the Mersenne Twister to produce pseudorandom numbers normally distributed about 0. Two types of non-equilibrated samples are generated; linear and exponentials. The linear non-equilibrated samples are generated with slopes varying from -0.1 to -1.1 and with the same type of normally distributed noise about the net slope. The exponential non-equilibrated samples add $\exp(-\tau x)$ to the equilibrated case, for $1.5 \leq \tau \leq 3$. Sample python code for generating these populations in included below:

```
import numpy as np
def eq(n=512):
    return np.random.randn(n)
```

```
def neq_linear(n=512,min_slope=0.1):
    slope = -(np.random.rand()+min_slope)
    return np.arange(n)*slope/n+np.random.randn(n)

def neq_exp(n=512):
    e = np.exp(-3*(np.random.rand()+1.)*np.arange(n)/2./n)
    return e+np.random.randn(n)

%Generate training data with both types of non-equilibrium functions
def training_data(n_samples=2000,size=512):
    plots = eq(size)
    labels = np.array([1.0,0.0])
    for i in range(n_samples//2-1):
        plots = np.vstack((plots,eq(size)))
        labels = np.vstack((labels,np.array([1.0,0.0])))
    for i in range(n_samples//2):
        func = neq_linear
        if i%2==0:
            func = neq_exp
        plots = np.vstack((plots,func(size)))
        labels = np.vstack((labels,np.array([0.0,1.0])))
    return plots, labels
```

Before training the neural net on these sample observable sequences, we normalize them by subtracting out the sequence's average and dividing by its standard deviation. This normalization helps to ensure the neural net is trained to recognize relative changes in sequence for sequences of arbitrary magnitude. Each data set is produced with 64, 128, 256, or 512 x-values on the interval $[0, 1)$. The exponential decay constants and linear slopes used here are justified by being representative of simulation data and impossible for the authors to distinguish by eye (Fig. 2). Autocorrelation analysis of the equilibrated data sets gives $\delta t_0 = 1$ (neighboring samples are perfectly decorrelated, as expected), while the linear (slope $m = -0.1$) and exponential (decay time $\tau = 3$) nonequilibrated samples have $\delta t_0 = 3 \pm 1$. Here we train the neural net and charachterize the degree to which an it can distinguish the differences between these representative sequences.

We perform experiments with training populations of $N_t \in \{1600, 3200, 8000\}$, and generate independent test populations that are $\frac{1}{4}$ the size of the training population. We employ the TensorFlow python package for machine learning to facilitate training and testing[26]. TensorFlow can also take advantage of installed GPUs to accelerate its internal routines. Explaining the framework and syntactical details of TensorFlow is beyond the scope of this work, and can be found in online tutorials, including `www.tensorflow.org`. The following 18 lines of python use TensorFlow to define the network inputs and outputs as in Eqn. 2, the objective function that is minimized during training, perform the training, perform testing, and then returns the degree of accuracy ($[0, 1]$) with which the trained network correctly predicted the test data:

```
import tensorflow as tf
def single(train_plots, train_labels,test_plots,test_labelsn=512,rounds=2000):
    x = tf.placeholder(tf.float32,[None,n])
    W = tf.Variable(tf.zeros([n,2]))
    b = tf.Variable(tf.zeros([2]))
    y = tf.matmul(x,W)+b #Eqn. 2
    y_ = tf.placeholder(tf.float32,[None,2])
```

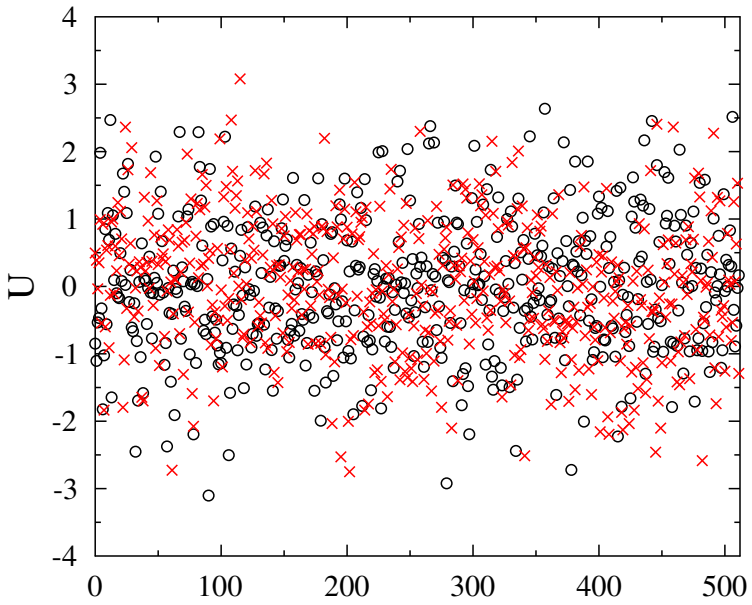


**Figure 2.** Normally-distributed observations (black) and normally-distributed observations with $\exp(-3x/512)$ added (red). The artificial neural network tested here can distinguish these data sets over 96% of the time.

```
obj = tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y)
cross_entropy = tf.reduce_mean(err)
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
for _ in range(rounds): #do the training
    sess.run(train_step, feed_dict={x:train_plots,y_:train_labels})
correct = tf.equal(tf.argmax(y,1),tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct,tf.float32))
return sess.run(accuracy,feed_dict={x:test_plots,y_:test_labels})
```

Here we investigate how information quality in the form of sample size $N$, number of training samples $N_t$, number of training rounds $r$ for a single layer artificial neural net influences is predictive accuracy $A$ for sample equilibrated and non-equilibrated observation sequences.

## 3. Results

Table 1 summarizes the predictive accuracy of the neural network as a function of information quality, for the training case where linear and exponential non-equilibrated data sets are included in equal parts, and where the non-equilibrated data has randomly generated slopes and decay rates, respectively. As is expected, predictive accuracy increases with the size of the input sequence $N$ and the number of training samples $N_t$. The number of training rounds (1000,2000,4000) is found to weakly influence accuracy. To better understand which observation sequences this neural net is best suited to distinguish, we restrict the types of non-equilibrium sequences are included in the training and testing sets and perform additional experiments. In *small* experiments we use $N = 256$, $N_t = 1600$, $r = 1000$, and for *large* experiments we use $N = 512$, $N_t = 8000$, $r = 2000$. The small experiments take less than thirty seconds on an

**Table 1.** Prediction accuracy when randomly sloped linear and randomly decaying exponential plots are included in the training and test population samples.

| $N$ | $N_t$ | $r$ | $A$ |
|---|---|---|---|
| 64 | 1600 | 2000 | .79 |
| 128 | 1600 | 2000 | .82 |
| 256 | 1600 | 2000 | .84 |
| 512 | 3200 | 4000 | .86 |
| 512 | 3200 | 1000 | .88 |
| 512 | 1600 | 2000 | .89 |
| 512 | 3200 | 2000 | .90 |
| 512 | 8000 | 1000 | .91 |

NVIDIA GeForce GT 750M GPU, while the large experiments take up to three minutes. We distinguish the accuracies that correspond to these tests with $A_s$ and $A_l$ respectively.

In the case of only linear sequences with constant slope $m = -0.1$, we find the network is unable to reliably distinguish these from equilibrated data ($A_s = .566$ and $A_l = .599$). If the slope of the linear sequences is increased to $m = -.2$ we find the network does relatively better ($A_s = .664$ and $A_l = .716$), suggesting that the network can detect steeper slopes more easily. When the slope of the linear sequences is uniformly chosen on $[0.2, 2.2)$ we find the network does even better ($A_s = .862$ and $A_l = .912$).

We find the network is more effective at distinguishing randomized exponentially decaying observations than equilibrium observations, with $A_s = .968$ and $A_l = .996$ for $exp(-\tau x)$ and $\tau = -3$. When $\tau$ is uniformly chosen on $[1.5, 3)$, we find the network is approximately as predictive as in the constant $\tau$ case, with $A_s = .968$ and $A_l = .990$.

## 4. Conclusions

Artificial neural networks are a promising advance for automating the identification of "equilibrated" measurements from thermodynamic simulations. A few minutes of training time on a laptop GPU is sufficient to train a two-neuron artificial network for identifying an exponential decay hidden in normally-distributed data more than 96% of the time. As this is the simplest possible artificial neural net that can be applied to the equilibrium-or-not decision problem, this suggest that neural networks with multiple layers of neurons can be applied to this problem with greater efficacy. The high accuracy of identifying random sequences with exponential decay is encouraging, because these sequences are characteristic of relaxations to equilibrium in thermodynamic simulations. However, our artificial neural network was not able to reliably distinguish random sequences with a small, constant slope embedded within them. Improvements to the two-neuron network studied here are needed for more robust pattern recognition. Future work applying multilayered artificial neural networks to automatically identify the subset of observations from a sequence that best represent equilibration are warranted, and seem accessible given the successes of deep (thousands of layers) networks in more computationally demanding pattern recognition problems. Open source machine learning libraries including TensorFlow provide a way to apply pattern recognition to problems in computer simulation, making these future studies more accessible, or even routine.

## References

[1] Rahman A 1964 *Phys. Rev. A* **136** A405–A411

[2] Long A W and Ferguson A L 2014 *J. Phys. Chem. B* **118** 4228–4244
[3] Wang F and Landau D P 2001 *Phys. Rev. Lett.* **86** 2050–2053
[4] Perera D, Vogel T and Landau D P 2016 *Phys. Rev. E* **94** 043308 2470-0045
[5] Gai L, Vogel T, Maerzke K A, Iacovella C R, Landau D P, Cummings P T and McCabe C 2013 *J. Chem. Phys.* **139** 54505 1089-7690
[6] Reinhardt A and Frenkel D 2014 *Phys. Rev. Lett.* **112** 238103 0031-9007
[7] Karplus M and McCammon J A 2002 *Nat. Struct. Biol.* **9** 646–652
[8] Liu Y, Prigozhin M B, Schulten K and Gruebele M 2014 *J. Am. Chem. Soc.* **136** 4265–4272
[9] Schelling T C 1971 *J. Math. Sociol.*
[10] Metropolis N, Rosenbluth A W, Rosenbluth M N, Teller A H and Teller E 1953 *J. Chem. Phys.* **21** 1087–1092
[11] Alder B and Wainbright T 1957 *J. Chem. Phys.* **27** 1208
[12] Frenkel D 2013 *Eur. Phys. J. Plus* **128** 10 2190-5444
[13] Perez D, Uberuaga B P, Shim Y, Amar J G and Voter A F 2009 *Chapter 4 Accelerated Molecular Dynamics Methods: Introduction and Recent Developments* vol 5 (Elsevier) ISBN 9780444533593
[14] Sugita Y and Okamoto Y 1999 *Chem. Phys. Lett.* **314** 141–151
[15] Jankowski E and Glotzer S C 2009 *J. Chem. Phys.* **131**
[16] Borovský M, Weigel M, Barash L Y and Žukovič M 2016 *EPJ Web Conf.* **108** 02016 2100-014X
[17] Glaser J, Nguyen T D, Anderson J A, Lui P, Spiga F, Millan J A, Morse D C and Glotzer S C 2014 *Comput. Phys. Commun.* **192** 97–107
[18] Anderson J A, Jankowski E, Grubb T L, Engel M and Glotzer S C 2013 *J. Comput. Phys.* **254** 27–38
[19] Bishop C M 2013 *Pattern Recognition and Machine Learning* vol 53 ISBN 9788578110796 (*Preprint*
[20] Silver D, Huang A, Maddison C J, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T and Hassabis D 2016 *Nature* **529** 484–489 ISSN 0028-0836
[21] Gelly S and Wang Y 2006 URL `http://eprints.pascal-network.org/archive/00002713/`
[22] Long R, Li B, Liu Z and Liu W 2016 *Chem. Eng. J.* **284** 325–332
[23] Ruff K M, Harmon T S and Pappu R V 2015 *J. Chem. Phys.* **143** 243123
[24] Phillips C L and Voth G a 2013 *Soft Matter* **9** 8552 1744-683X
[25] Hachmann J, Olivares-Amaya R, Atahan-Evrenk S, Amador-Bedolla C, Sánchez-Carrera R S, Gold-Parker A, Vogt L, Brockway A M and Aspuru-Guzik A 2011 *J. Phys. Chem. Lett.* **2** 2241–2251
[26] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, et al., 2015 TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (`arXiv:1603.04467v2`)