TorchANI: A Free and Open Source PyTorch Based Deep Learning Implementation of the ANI Neural Network Potentials

Xiang Gao,[†] Farhad Ramezanghorbani,[‡] Olexander Isayev,[¶] Justin S. Smith,[§] and Adrian E. Roitberg[*,‡]

[†] *Work done at: Department of Chemistry, University of Florida, Gainesville, FL 32611, USA; Now at: NVIDIA Corporation, 2788 San Tomas Expressway, Santa Clara, CA 95051*

[‡] *Department of Chemistry, University of Florida, Gainesville, FL 32611, USA*

[¶] *Department of Chemistry, Carnegie Mellon University, Pittsburgh PA, 15213, USA*

[§] *Center for Nonlinear Studies and Theoretical Division, Los Alamos National Laboratory, Los Alamos, NM 87545, USA*

E-mail: roitberg@ufl.edu

Abstract

This paper presents TorchANI, a PyTorch based software for training/inference of ANI (ANAKIN-ME) deep learning models to obtain potential energy surfaces and other physical properties of molecular systems. ANI is an accurate neural network potential originally implemented using C++/CUDA in a program called NeuroChem. Compared with NeuroChem, TorchANI has a design emphasis on being light weight, user friendly, cross platform, and easy to read and modify for fast prototyping, while allowing acceptable sacrifice on running performance. Because the computation of atomic environmental vectors (AEVs) and atomic neural networks are all implemented using PyTorch operators, TorchANI is able to use PyTorch's autograd engine to automatically compute analytical forces and Hessian matrices, as well as do force training without additional codes required. TorchANI is open-source and freely available on GitHub: `https://github.com/aiqm/torchani`

Introduction

The potential energy surface (PES) of atomistic systems plays a major role in physical chemistry: it is a core concept in molecular geometries, transition states, vibrational frequencies, and much more. Existing approaches for obtaining a molecular PES can be categorized into two general classes: quantum mechanics (QM) and molecular mechanics[1] (MM). The correct physics for obtaining the PES of molecules is given by QM, or more specifically by solving the many-body Schrödinger equation (MBSE), which takes the interaction of electrons and nuclei into account. However, solving the MBSE is QMA-Hard[2]. That is to say, on any computer that humans have theorized, including quantum computers, obtaining an exact solution of the MBSE is intractable[3]. In practice, numerous approximations have been developed to obtain solutions to the MBSE. Depending on the accuracy of the approximation, the computational cost varies drastically across methods. Kohn-Sham density functional theory (DFT)[4] and coupled cluster theory[5] are two popular approximations. These methods tend to be accurate compared to MM methods but computationally very expensive; for example, DFT scales as $\Theta\left(N^3\right)$ and CCSD(T) scales as $\Theta\left(N^7\right)$, where N is the number of electrons in the molecule. A general trend is, the better the accuracy, the worse the computational scaling with system size.

The molecular mechanics (MM) approach does not directly account for electrons. It obtains an approximate PES by defining bonds, angles, dihedrals, non-bonded interactions, etc. and then parameterizing specific functions for describing these interations. The obtained potential are called force fields. Due to a restrictive functional form and limited parameterization, force fields often yield non-physical results when molecules are far from equilibrium geometry, or when applied to molecules outside their fitting set. For example, in most force fields, bonds cannot break due to the use of a harmonic functional form for bonding. Despite these problems, force fields have the advantage of scaling as $O\left(N^2\right)$ with respect to the number of atoms in the system N, which leads to their wide use in the study of large systems like proteins and DNA.

Recent deep learning developments in many fields[6] have shown that an artificial neural network is generally a good approximator of functions[7]. Being aware of this fact, researchers in the field of computational chemistry have been deploying neural networks and other machine learning-based models for the prediction of QM computed properties.[8–31] These models aim to bypass solving the many-body Schrödinger equation by directly predicting QM properties. In recent years, a few of these models have been released as open source codes, many in machine learning frameworks such as TensorFlow or PyTorch.

In this article, we introduce an open source implementation of the ANI[32] style neural network potential in PyTorch. ANI is a general-purpose neural network-based atomistic potential for organic molecules. To date, three ANI models have been published, the ANI-1,[32] ANI-1x,[33] and ANI-1ccx[34] potentials. The ANI-1 model was developed by random sampling conformational space of 57k organic molecules with up to 8 heavy atoms, C, N, and O, plus H atoms to have proper chemistry, then running DFT calculations to obtain potential energies for training. ANI-1x was trained to a data set of molecular conformations sampled through an active learning scheme. Active learning is where the model itself is iteratively used to decide what new data should be included in the next iteration. Finally, ANI-1ccx was trained to the ANI-1x data set, then retrained to a 10% smaller data set of accurate coupled cluster calculations, resulting in a potential that outperformed DFT in test cases. We include the ANI-1x and ANI-1ccx potentials with our framework. The general philosophy of our software is to provide the public with an easily accessible and modifiable version of the ANI model for deploying our existing ANI-1x, ANI-1ccx, and future models, for training new models to new data sets, or for fast prototyping of new ideas and concepts.

Similar to traditional force fields, ANI does not explicitly treat electrons and defines the potential energy directly as an explicit function of coordinates of atoms. But, unlike force fields, ANI does not predefine concepts like bonds, and the functional form of potential energy in ANI is an artificial neural network. Since ANI does not solve the Schrödinger equation, the computational cost of ANI comparable to force fields, which makes ANI able to scale to large molecules like proteins. Being trained on synthesized data computed by quantum chemical methods, such as DFT[32,33,35] and CCSD(T)/CBS[34], ANI can predict most parts of the potential energy surface at a quantum level. Since the level of accuracy is at quantum chemical level, it should be able to capture important properties such as bond breaking that traditional force fields cannot model.

As discussed by Behler,[36] there are symmetries that the predicted potential energy has to obey: it has to be invariant under the transformations of translation, rotation, and permutation of the same type of atoms. Behler et al presented an architecture that satisfies this type of symmetry.[10] In that work, for each atom, a fixed-size representation of its chemical environment called an atomic environmental vector (AEV) is computed. AEVs are invariant under translation and rotation. The AEV of each atom is further passed through a neural network to get a scalar, the atomic contribution of this total energy. The total molecular energy is obtained by adding up these atomic energies. If the neural networks applied to the AEVs of the same type of atoms are the same as each other, the permutation symmetry is also satisfied.

The AEVs in ANI are modified from those in Behler and Parrinello[10]. The structure of AEV in ANI which is composed of radial and angular parts is shown in Figure 1. The radial AEV is further divided into subAEVs according to atom species. Similarly, angular AEV is further divided into subAEVs according to pairs of atom species. Each subAEV only cares about neighbor atoms of its corresponding species/pair of species. Loosely speaking, we can think of AEV as counting the number of atoms for different species/pair of species, at different distances and angles. Interested readers are referred to[32] for more detail.

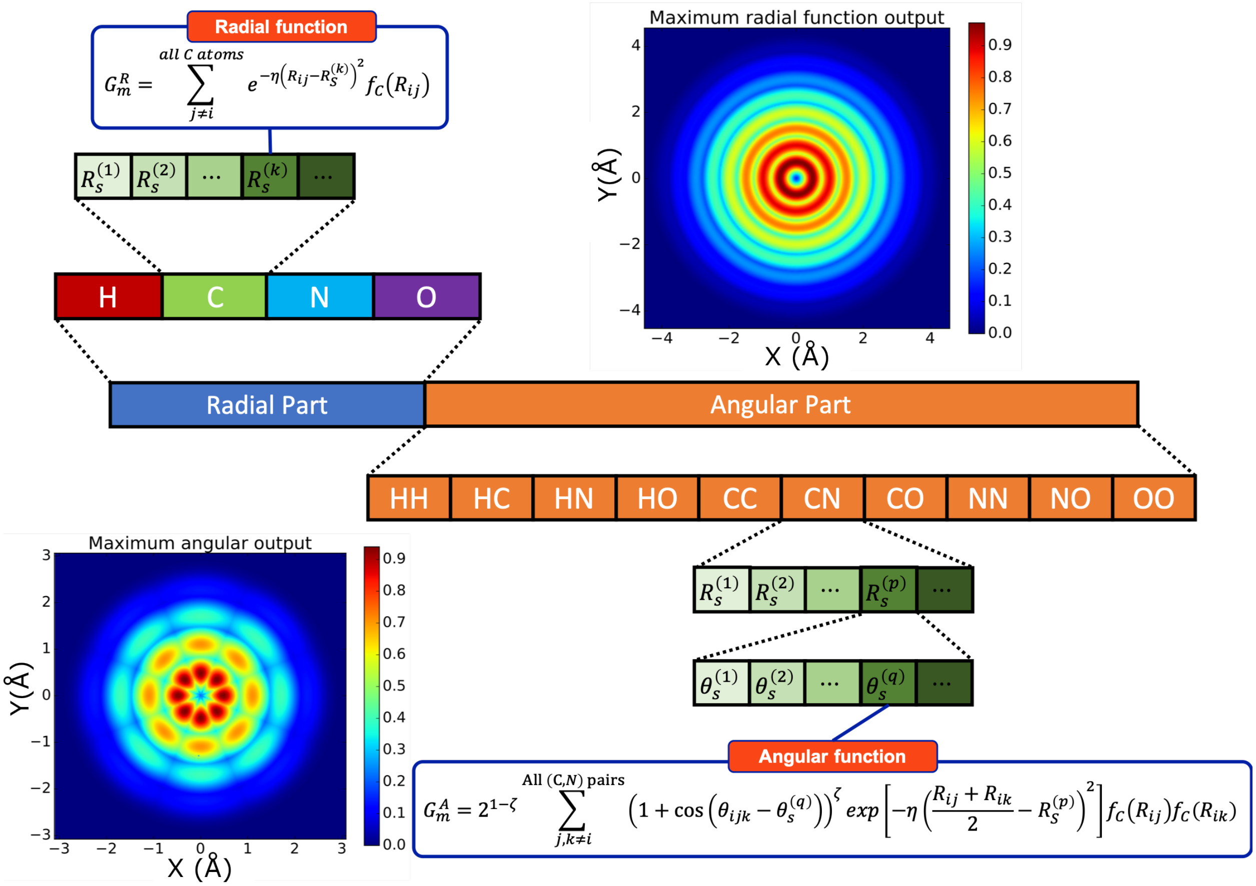
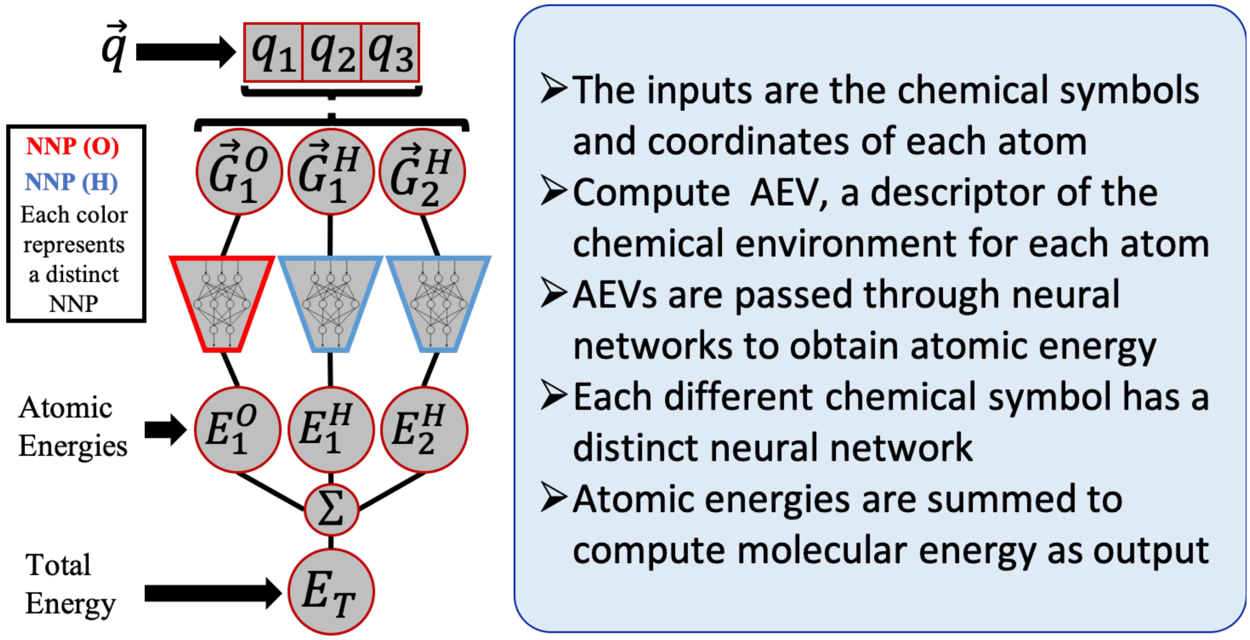


Figure 1: The Structure of the ANI AEVs.
The sum of j and k is on all neighbor atoms of selected species/pair of species. R_s and θ_s are hyper-parameters called radial/angular shifts. The f_C is called cutoff cosine function, defined as $f_C(R) = \frac{1}{2}\left[\cos\left(\frac{\pi R}{R_C}\right)+1\right]$ for $R \leq R_C$ and 0 otherwise, where R_C is called cutoff radius, a hyperparameter that defines how far we should reach when investigating chemical environments.

As shown in Figure 2, after computing the AEV for each atom, these AEVs are further passed forward through the neural network to obtain atomic energies, which will be further summed together for each molecule to obtain the total energy. The AEVs of the atoms with the same atomic numbers are passed through the same neural network.



- ➢The inputs are the chemical symbols and coordinates of each atom
- ➢Compute AEV, a descriptor of the chemical environment for each atom
- ➢AEVs are passed through neural networks to obtain atomic energy
- ➢Each different chemical symbol has a distinct neural network
- ➢Atomic energies are summed to compute molecular energy as output

Figure 2: From AEV to Molecule Energy
Figure reproduced from Ref.[32] with permission from the Royal Society of Chemistry.

In the first version of ANI, aka ANI-1, the training data is a set of synthesized data, called the ANI-1 dataset[35], coming from DFT ωB97X/$6-31$G(d) computations of energies of near equilibrium structures of small organic molecules using normal mode sampling. Only elements H, C, N, and O are supported.

ANI was originally implemented in C++/CUDA in a program called NeuroChem, which allows us to do lighting fast training and inference on modern NVIDIA GPGPUs. High performance of the NeuroChem code is obtained as a trade-off with fast prototyping, lossy maintenance, simple installation and cross platform. This motivate us to implement a light weight and easy to use version, i.e. TorchANI. TorchANI is not designed to replace NeuroChem. But instead, it is a complement to NeuroChem with different design emphasis and expected use case.

Methods

PyTorch based implementation

In terms of software for neural network potential researches, both performance and flexibility are important. But unfortunately, performance and flexibility usually can not be achieved together. Trade-offs has to be made when designing a software.

On the one hand, there are researchers seeking for using neural network potentials to study large bio-molecules like proteins at a highly accurate level, which has a high demand on inference performance of the software. Besides, the quality of a neural network potential highly depend on the quality of the dataset on which the potentional is trained. Researches on improving dataset quality is to use accurate synthesized data to cover the chemical space more complete and balanced. To achieve this goal, we have proposed to use active learning[33] to incrementally expand the dataset, from HCNO only to HCNOSFCl,[37] and from near equilibrium structures only to reaction pathways, and from DFT to coupled-cluster.[34] The fact that active learning requires a large number of training makes the training performance also critical in such kind of researches.

On the other hand, for researches prototyping neural networks of different architectures, loss functions, optimizers, the software should be highly flexible. It should also be cross-platform so that researchers could try their idea both on a GPU server and on a laptop. The best technology selection for this purpose is to use a popular deep learning framework, which allows employing the implementations of the most modern methods in the rapidly growing field of machine learning. Since its release, PyTorch[38] has gained a great reputation on its flexibility and ease to use, and has become the most popular deep learning framework among researchers. TorchANI is an implementation of ANI on PyTorch, aimed to be light weight, user-friendly, cross-platform, and easy to read and modify.

Major deep learning frameworks could be categorized as layer-based frameworks like Caffe[39] and compute graph-based frameworks like PyTorch,[38] TensorFlow,[40] and MXNet.[41]

Layer-based frameworks consider a neural network as several layers of neurons stacked together. The software usually allocates memory buffers to store inputs and outputs, as well as the gradients obtained during back-propagation, for each layer. The core of the software is a CPU code and CUDA kernels that fill in these buffers. Frameworks of this type are simple in design and fast in performance. However, considering deep learning models as a stack of layers is a very restrictive assumption. As a result, not all deep learning models fit into the framework of layers. Also, the lack of data structure to store the computation history makes it very hard to implement higher order derivatives.

Compute graph-based deep learning frameworks, such as PyTorch, usually contain automatic differentiation engine.[42,43] The engine stores the data dependency as a graph and contains API that allows users to invoke algorithms to investigate the mathematical operations of the history and compute the derivatives in one line of code. NeuroChem is coded as a layer-based program.

Unlike most deep learning researches in the field of computer vision and natural language processing, etc., in which the automatic differentiation engine is only used in computing the derivatives of the loss function with respect to model parameters, the automatic differentiation engine could be more useful in chemistry: many physical properties are defined as derivative of two other properties, say $C = \frac{\partial A}{\partial B}$. Due to this nature of science, higher order derivatives are also more important than in the general artificial intelligence community. By using the automatic differentiation engine of PyTorch, people can write down the code that computes A from B, and the framework provides tools to automatically compute C. Higher order derivatives could also be computed within a few lines of code. We will show some example on how the automatic differentiation engine could be used with TorchANI:

Example 1: For a periodic system, the stress tensor is defined as the per area force pulling the system on surfaces of different directions. It can be computed as $\sigma_{ij} = \frac{1}{V} \cdot \frac{\partial E(\lambda_{ij})}{\partial \lambda_{ij}}$, where $E(\lambda_{ij})$ is the energy as a function of the factor λ_{ij} of shearing the system and cell simultaneously in a direction defined by i while keeping the direction of the surfaces defined by j unchanged. On PyTorch, the pseudo-code of implementing stress can be as simple as shown in Listing 1.

Listing 1: Compute Stress

See the source code of the stress implementation in the Atomic Simulation Environment (ASE)[44] interface of TorchANI for more detail.

```
displacement = torch.zeros(...)
scaling_factor = 1 + displacement
new_cell, new_coordinates = scale_system_and_unit_cell(
    cell, coordinates, scaling_factor)
# Numerically new_cell and new_coordinates has the same values as
# old values, i.e. cell, coordinates because the are just distorting
# the system by zero. But the new values contain compute graph on
# how they are related to displacement, so that the autograd engine
# can compute the gradient from the graph.
energy = compute_energy(new_cell, new_coordinates)
stress = torch.autograd.grad(energy, displacement)[0] / volume
```

Example 2: An important task in computational chemistry is the analysis of molecular vibrations. To compute the normal modes and frequencies of vibrations, we need to compute the Hessian matrix first and then compute the eigenvalues and eigenvectors of the mass scaled Hessian matrix. In TorchANI, thanks to the autograd engine of PyTorch, achieving such a task is as simple as shown in Listing 2.

Listing 2: Vibrational Analysis

See also `https://aiqm.github.io/torchani/examples/vibration_analysis.html`

```
_, energies = model((species, coordinates))
hessian = torchani.utils.hessian(coordinates, energies=energies)
element_masses = torch.tensor([
    1.008,  # H
    12.011,  # C
    14.007,  # N
    15.999,  # O
], dtype=torch.double)
masses = element_masses[species]
freq, modes = torchani.utils.vibrational_analysis(masses, hessian)
```

In the above code, the torchani.utils.hessian is a short function that first computes forces using torch.autograd, and then loop on every element of the forces to compute the Jacobian matrix of forces with respect to coordinates. The torchani.utils.vibrational_analysis scales the hessian with mass, and diagonalize to obtain the frequencies and normal modes.

Example 3: Compared with energy, force is more critical in molecular dynamics because energy is just an observer (print its value at each step), but the force is a player of the game (velocities are updated according to force). Training to energy solely does not necessarily lead to good forces (see the experiment in Section Benchmark). A straightforward solution to make the model predicting good forces is training to force, which requires taking the second derivative of the predicted energies. Implementing force training is trivial in PyTorch: we just need to add a few lines of code to our energy trainer, as shown in Listing 3.

Listing 3: Train to Force

See also: `https://aiqm.github.io/torchani/examples/nnp_training_force.html`

```
forces = -torch.autograd.grad(energies.sum(), coordinates,
                              create_graph=True, retain_graph=True)[0]
force_loss = mse(true_forces, forces).sum(dim=(1, 2)) / num_atoms
loss = energy_loss + alpha * force_loss
```

Example 4: The infrared (IR) intensity is computed as

$$A = \frac{1}{4\pi\epsilon_0} \cdot \frac{N_A \pi}{3c} \cdot \left(\frac{\partial \bar{\mu}}{\partial \vec{R}_k^{(q)}} \right)^2$$

Where $\bar{\mu}$ is the dipole moment, and $\vec{R}_k^{(q)}$ is the vibrational coordinates. As long as we could train a neural network predicting dipoles, the computation of IR intensity using PyTorch would also be straightforward. Starting from the normal modes, which could be computed as shown in Listing 2, the pseudo-code to obtain its IR intensity is shown in Listing 4.

Listing 4: Compute IR Intensity

```
cartesian_coordinates = to_cartesian(normal_coordinates)
dipole_moment = dipole_model(cartesian_coordinates)
grad_dipole = torch.autograd.grad(dipole_moment, normal_coordinates)[0]
ir_intensity = coefficient * grad_dipole ** 2
```

Design

TorchANI is composed of the following major parts:

- The core library, including AEV computer, species-differentiated atomic neural network, and some other utilities.
- The dataset utilities to prepare datasets and add necessary padding to be used in the training and evaluation of ANI models.
- The NeuroChem compatibility module that can: 1) read networks trained on NeuroChem, and 2) read NeuroChem's training configuration files and train on PyTorch with precisely the same procedure.
- The Atomic Simulation Environment (ASE)[44] interface with full periodic boundary condition and analytical stress support that allows users to run structure optimization, molecular dynamics, and etc., with ANI using ASE.
- The ANI model zoo that stores public ANI models

The major part of the core library consists of three classes, `AEVComputer`, `ANIModel`, and

`EnergyShifter`, which are used to build the coordinate-AEV-energy pipeline:

$$\text{Coordinates} \xrightarrow{\texttt{AEVComputer}} \text{AEVs} \xrightarrow{\texttt{ANIModel}} \text{Raw energies} \xrightarrow{\texttt{EnergyShifter}} \text{Molecular energies}$$

All three of these classes are subclasses of `torch.nn.Module`. The inputs of all these classes are tuples of size 2, where the first elements of the tuple are always `species`, a `LongTensor` storing the species of each atom in each molecule. The `species` are passed through to the output unchanged, which allows us to pipeline objects of these classes using `torch.nn.Sequential`. The energies computed by `ANIModel` (called raw energies) are different from the real molecular energy by a number that scales linearly with the number of atoms of each species. `EnergyShifter` is the class responsible for shifting the raw energies to real molecular energies.

The dataset utilities provide tools to read the published dataset of the same format as in[35] and prepare it for training in TorchANI. The trick here is padding. Training of ANI models uses stochastic gradient descent, which requires creating mini-batches containing different molecules. A natural way to design this is to make the model have an input that is a tensor of shape (molecules, atoms, 3) as coordinates and (molecules, atoms) to store the type of elements of atoms. However, each minibatch contains molecules with a different number of atoms. The nature of a tensor being an n-dimensional array makes it impossible to make the whole batch a single tensor. Our solution was to “invent” a new ghost element type -1, which does not exist on the periodic table. When batching, we pad all molecules by adding atoms of the ghost element to make all molecules in the batch have the same number of total atoms.

The code in Listing 5 shows how to use TorchANI to compute the energy and force of a methane molecule, using an ensemble of 8 different ANI-1ccx[34] models. From the example, we can see that the whole coordinate→energy pipeline is part of the computational graph so the gradients and higher-order derivatives can be computed using PyTorch’s automatic differentiation engine.

Listing 5: Compute Energy and Force Using ANI-1ccx Model

See also: `https://aiqm.github.io/torchani/examples/energy_force.html`

```
import torch
import torchani

model = torchani.models.ANI1ccx(periodic_table_index=True)
# To use a single model instead of an ensemble,
# replace the above line with:
#    model = torchani.models.ANI1ccx(
#          periodic_table_index=True)[0]

coordinates = torch.tensor([[[0.03,   0.006,   0.01],
                             [-0.8,   0.4,  -0.3],
                             [-0.7,  -0.8,   0.2],
                             [0.5,    0.5,   0.8],
                             [0.7,   -0.2,  -0.9]]],
                           requires_grad=True)
# In periodic table, C = 6 and H = 1
species = torch.tensor([[6, 1, 1, 1, 1]])

_, energy = model((species, coordinates))
force = -torch.autograd.grad(energy.sum(), coordinates)[0]

print('Energy:', energy.item())
print('Force:', force.squeeze())
```

Taking advantage of the power of PyTorch's autograd engine, training to force becomes trivial. Listing 3 shows the additional code added to the energy training script to train an ANI model to force. We can see training a network to forces requires only a few additional lines of code, as demonstrated in Listing 3.

TorchANI provides tools to compute the analytical Hessian using autograd engine and to perform vibrational analysis, as shown in Listing 2. TorchANI also provides analytical stress support, and it will be automatically used when the user is using the ASE interface to do a NPT simulation with periodic boundary conditions. A set of detailed example files and documentations for training and inference using TorchANI is available at `https:`

`//aiqm.github.io/torchani/`.

Results and discussion

Benchmark

All benchmarks are done on a workstation with NVIDIA GeForce RTX 2080 GPU and Intel i9-9900K CPU. Training on the whole ANI-1x dataset[33] with network architecture identical to the one used by NeuroChem takes 54 seconds per epoch. Within the 54 seconds, 16 seconds are spent on computing AEV, 28 seconds are spent on neural networks, and the rest are on backpropagation. In comparison, NeuroChem takes 18 seconds for each epoch using the same GPU/CPU architecture.

We also measured the number of seconds it takes to do 1000 steps of molecular dynamics. All models are run in double data type on GPU. We tested both periodic and non-periodic systems. We use water boxes with densities between 0.94g/mL and 1.17g/mL (except for the very small system with only eight waters, which has density 0.72g/mL) of different size for all periodic tests. The time vs. size of the system for both the periodic system and the non-periodic system, as well as for both single ANI model and ANI model ensembles, are shown in Table 1.

We also report the training behavior on the ANI-1x[33] dataset. The whole ANI-1x dataset is split into $80\% + 10\% + 10\%$ where 80% of the data are used as the training set, 10% are used as validation, and the other 10% are used as testing. We compare the results of training only to energy and to both energy and force. When training with force, the loss function is defined as $loss = (\text{energy loss}) + \alpha \times (\text{force loss})$, with different α values. For the training to energy experiment, the MSE loss is scaled by the square root of the number of atoms per each molecule, as described in.[32] The performance on the COMP6 benchmark[33] for the resulting models of these training are shown at table 2. Energies are in kcal/mol, forces are in $\text{kcal}/\left(\text{mol} \cdot \text{Å}\right)$. Error keys are MAE/RMSE. From the table, we can see that although

Table 1: Seconds for 1000 molecular dynamics steps

System	PBC	Total Atoms	Single Network	Ensemble of 8 Networks
benzene	No	12	5.80	14.80
PHE-GLU-ILE tripeptide	No	58	7.14	22.69
ALA14	No	143	7.00	23.83
ALA28	No	283	7.24	24.52
ALA42	No	423	7.32	24.32
ALA56	No	563	7.98	27.00
ALA84	No	843	8.58	27.12
ALA126	No	1263	9.36	27.45
ALA252	No	2523	14.87	35.14
ALA504	No	5043	30.87	53.37
water box	Yes	24	13.27	21.30
water box	Yes	51	12.53	22.02
water box	Yes	150	13.54	22.38
water box	Yes	300	16.10	24.39
water box	Yes	501	19.65	28.28
water box	Yes	801	32.19	41.47
water box	Yes	1200	56.10	65.58

enabling force training night hurt the RMSE of absolute energies, the prediction of the relative energies always improve. The relative energy is a more important quantity than the absolute energy because it is related to reaction barriers and conformational changes.

Table 2: COMP6 benchmark result for different models. MAE/RMSE (kcal/mol)

Model	Energy ($\alpha = 0$)	$\alpha = 0.5$	$\alpha = 0.25$	$\alpha = 0.1$
Energy	2.27/3.62	3.10/33.93	2.73/4.50	2.43/3.93
Relative Energy	2.29/3.51	1.95/3.09	1.93/3.07	1.90/3.00
Forces	4.41/6.96	2.33/3.75	2.30/3.75	2.35/3.88

Application

In addition to its mentioned training/inference capabilities, we use TorchANI to train a fully connected neural network to predict the NMR chemical shift of α and β carbons of proteins on the the dataset used by SHIFTX2.[45] NMR chemical shifts in proteins are used to determine the protein structure. It is an atomic property that depends on many factors, including the local protein structure as well as environmental factors such as hydrogen bonding and pH.[45] SHIFTX2 is a program that predicts chemical shifts by combining different methods, including machine learning. The dataset used to train SHIFTX2 is publicly available and can be downloaded at `http://www.shiftx2.ca`.

Protein chemical shift databases usually contain chemical shift data of different types of hydrogens, carbons, and nitrogens. Among these atoms, α and β carbons are mostly related to structural information of the protein itself, rather than environments like hydrogen bonding of nearby water molecules,[45] making them an excellent choice for a simple application of predicting a property solely based on local structure.

Since NMR chemical shifts are atomic properties, they are well suited to the ANI architecture. We build a fully connected neural network with only one hidden layer, which contains 256 neurons. We use Exponential Linear Unit (ELU)[46] activation function to add non-linearity to this network. The input of the network is solely the AEV for the atom to be predicted, and the output is the chemical shift we are predicting. The AEV computer only supports five elements: HCNOS. The length of each AEV is 560. Ligands and ions in the protein structures are deleted so that each atom of interest only contains these five elements in its neighborhood.

SHIFTX2 dataset contains a training set, which we use to train our models, and a testing set which we use to evaluate our trained models. We train two different neural networks, one for α carbons and the other for β carbons. After training, the resulting models can predict the chemical shift of ${}^{\alpha}C$ with a coefficient of determination $R^2 = 0.96$ on the testing set, which for ${}^{\beta}C$ this number is $R^2 = 0.99$. The 2D histogram in logarithm scale for the true values vs. predicted values is shown in Figure 3.

Acknowledgements

TorchANI has been public as a free and open source software at GitHub since Oct 2018. The authors would like to thank all the users of TorchANI for using and providing feedback to us.

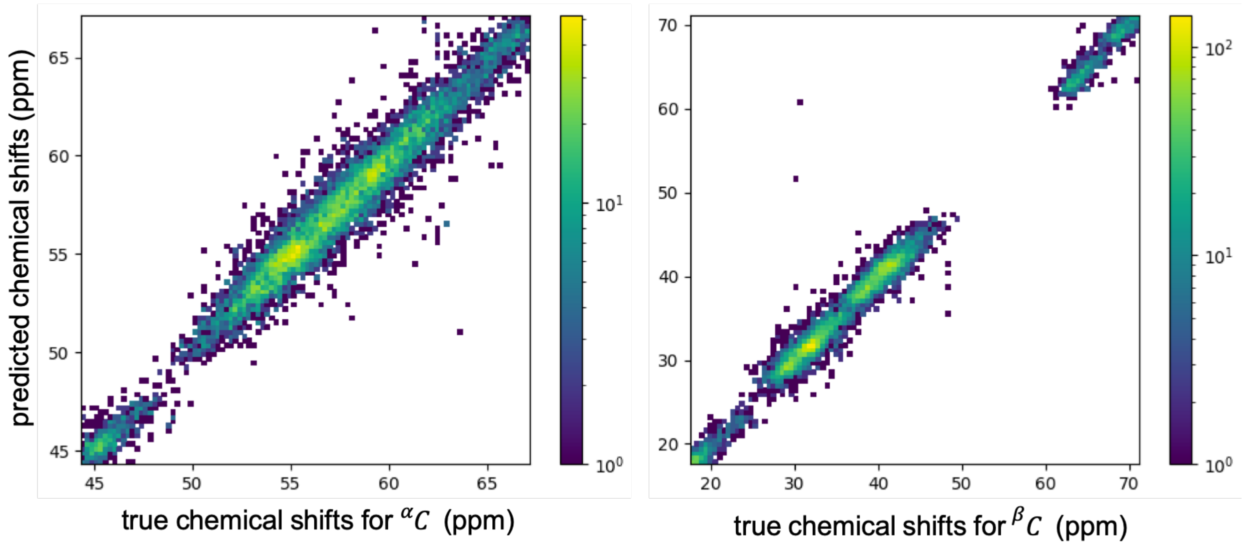


Figure 3: The 2D Histogram for the Prediction of Chemical Shift
Note that the color scale is logarithmic, the yellow means 100x more populated than the deep blue.

Contributions of code improvements from Ignacio J. Pickering and Jinze Xue's improvements on ANI data loader is also worth mentioning.

Farhad Ramezanghorbani would like to thank the Molecular Sciences Software Institute (MolSSI) for a fellowship award under NSF grant ACI-1547580. Adrian E. Roitberg would like to thank National Science Foundation for supporting this research with NSF CHE-1802831 award. Justin S. Smith was supported by LDRD program and the Center for Nonlinear Studies (CNLS) at Los Alamos National Laboratory (LANL).

References

(1) Leach, A. R.; Leach, A. R. *Molecular modelling: principles and applications*; Pearson Education, 2001.

(2) Aaronson, S. Computational complexity: why quantum chemistry is hard. *Nature Physics* **2009**, *5*, 707.

(3) Watrous, J. Quantum computational complexity. *Encyclopedia of Complexity and Systems Science* **2009**, 7174–7201.

(4) Kohn, W.; Sham, L. J. Self-consistent equations including exchange and correlation effects. *Physical Review* **1965**, *140*, A1133.

(5) Bartlett, R. J.; Musiał, M. Coupled-cluster theory in quantum chemistry. *Reviews of Modern Physics* **2007**, *79*, 291.

(6) Alom, M. Z.; Taha, T. M.; Yakopcic, C.; Westberg, S.; Sidike, P.; Nasrin, M. S.; Van Esesn, B. C.; Awwal, A. A. S.; Asari, V. K. The history began from AlexNet: a comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164* **2018**,

(7) Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural Networks* **1991**, *4*, 251–257.

(8) Blank, T. B.; Brown, S. D.; Calhoun, A. W.; Doren, D. J. Neural network models of potential energy surfaces. *The Journal of Chemical Physics* **1995**, *103*, 4129–4137.

(9) Hobday, S.; Smith, R.; Belbruno, J. Applications of neural networks to fitting interatomic potential functions. *Modelling and Simulation in Materials Science and Engineering* **1999**, *7*, 397–412.

(10) Behler, J.; Parrinello, M. Generalized neural-network representation of high-dimensional potential-energy surfaces. *Physical Review Letters* **2007**, *98*, 146401.

(11) Han, J.; Zhang, L.; Car, R.; E, W. Deep Potential: a general representation of a many-body potential energy surface. *arXiv* **2017**, Preprint at https://arxiv.org/abs/1707.01478.

(12) Lubbers, N.; Smith, J. S.; Barros, K. Hierarchical modeling of molecular energies using a deep neural network. *The Journal of Chemical Physics* **2018**, *148*, 241715.

(13) Schütt, K. T.; Sauceda, H. E.; Kindermans, P. J.; Tkatchenko, A.; Müller, K. R. SchNet - A deep learning architecture for molecules and materials. *Journal of Chemical Physics* **2018**, *148*, 241722.

(14) Gastegger, M.; Schwiedrzik, L.; Bittermann, M.; Berzsenyi, F.; Marquetand, P. wACSFWeighted atom-centered symmetry functions as descriptors in machine learning potentials. *The Journal of Chemical Physics* **2018**, *148*, 241709.

(15) Zubatyuk, R.; Smith, J. S.; Leszczynski, J.; Isayev, O. Accurate and transferable multitask prediction of chemical properties with an atoms-in-molecules neural network. *Science Advances* **2019**, *5*, eaav6490.

(16) Rupp, M.; Tkatchenko, A.; Muller, K.-R.; von Lilienfeld, O. A. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical review letters* **2012**, *108*, 58301.

(17) Thompson, A. P.; Swiler, L. P.; Trott, C. R.; Foiles, S. M.; Tucker, G. J. Spectral neighbor analysis method for automated generation of quantum-accurate interatomic potentials. *Journal of Computational Physics* **2015**, *285*, 316–330.

(18) Faber, F. A.; Hutchison, L.; Huang, B.; Gilmer, J.; Schoenholz, S. S.; Dahl, G. E.; Vinyals, O.; Kearnes, S.; Riley, P. F.; Von Lilienfeld, O. A. Prediction errors of molecular machine learning models lower than hybrid DFT error. *Journal of chemical theory and computation* **2017**, *13*, 5255–5264.

(19) Glielmo, A.; Sollich, P.; De Vita, A. Accurate interatomic force fields via machine learning with covariant kernels. *Physical Review B* **2017**, *95*, 214302.

(20) Botu, V.; Batra, R.; Chapman, J.; Ramprasad, R. Machine learning force fields: construction, validation, and outlook. *The Journal of Physical Chemistry C* **2017**, *121*, 511–522.

(21) Kruglov, I.; Sergeev, O.; Yanilkin, A.; Oganov, A. R. Energy-free machine learning force field for aluminum. *Scientific reports* **2017**, *7*, 1–7.

(22) Jiang, B.; Li, J.; Guo, H. Potential energy surfaces from high fidelity fitting of ab initio points: the permutation invariant polynomial-neural network approach. *International Reviews in Physical Chemistry* **2016**, *35*, 479–506.

(23) Gassner, H.; Probst, M.; Lauenstein, A.; Hermansson, K. Representation of intermolecular potential functions by neural networks. *The Journal of Physical Chemistry A* **1998**, *102*, 4596–4605.

(24) Morawietz, T.; Sharma, V.; Behler, J. A neural network potential-energy surface for the water dimer based on environment-dependent atomic energies and charges. *The Journal of chemical physics* **2012**, *136*, 064103.

(25) Kolb, B.; Zhao, B.; Li, J.; Jiang, B.; Guo, H. Permutation invariant potential energy surfaces for polyatomic reactions using atomistic neural networks. *The Journal of chemical physics* **2016**, *144*, 224103.

(26) Handley, C. M.; Popelier, P. L. Potential energy surfaces fitted by artificial neural networks. *The Journal of Physical Chemistry A* **2010**, *114*, 3371–3383.

(27) Yao, K.; Herr, J. E.; Toth, D. W.; Mckintyre, R.; Parkhill, J. The TensorMol-0.1 model chemistry: a neural network augmented with long-range physics. *Chemical science* **2018**, *9*, 2261–2269.

(28) Bleiziffer, P.; Schaller, K.; Riniker, S. Machine learning of partial charges derived from high-quality quantum-mechanical calculations. *Journal of chemical information and modeling* **2018**, *58*, 579–590.

(29) Nebgen, B.; Lubbers, N.; Smith, J. S.; Sifain, A. E.; Lokhov, A.; Isayev, O.; Roitberg, A. E.; Barros, K.; Tretiak, S. Transferable dynamic molecular charge assignment

using deep neural networks. *Journal of chemical theory and computation* **2018**, *14*, 4687–4698.

(30) Gastegger, M.; Behler, J.; Marquetand, P. Machine learning molecular dynamics for the simulation of infrared spectra. *Chemical science* **2017**, *8*, 6924–6935.

(31) Chmiela, S.; Tkatchenko, A.; Sauceda, H. E.; Poltavsky, I.; Schütt, K. T.; Müller, K.-R. Machine learning of accurate energy-conserving molecular force fields. *Science advances* **2017**, *3*, e1603015.

(32) Smith, J. S.; Isayev, O.; Roitberg, A. E. ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost. *Chemical science* **2017**, *8*, 3192–3203.

(33) Smith, J. S.; Nebgen, B.; Lubbers, N.; Isayev, O.; Roitberg, A. E. Less is more: Sampling chemical space with active learning. *The Journal of Chemical Physics* **2018**, *148*, 241733.

(34) Smith, J. S.; Nebgen, B. T.; Zubatyuk, R.; Lubbers, N.; Devereux, C.; Barros, K.; Tretiak, S.; Isayev, O.; Roitberg, A. E. Approaching coupled cluster accuracy with a general-purpose neural network potential through transfer learning. *Nature Communications* **2019**, *10*, 2903.

(35) Smith, J. S.; Isayev, O.; Roitberg, A. E. ANI-1, A data set of 20 million calculated off-equilibrium conformations for organic molecules. *Scientific data* **2017**, *4*, 170193.

(36) Behler, J. Constructing high-dimensional neural network potentials: a tutorial review. *International Journal of Quantum Chemistry* **2015**, *115*, 1032–1050.

(37) Devereux, C.; Smith, J.; Davis, K.; Barros, K.; Zubatyuk, R.; Isayev, O.; Roitberg, A. Extending the Applicability of the ANI Deep Learning Molecular Potential to Sulfur and Halogens. **2020**,

(38) Paszke, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32. 2019; pp 8024–8035.

(39) Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093* **2014**,

(40) Abadi, M. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015; `https://www.tensorflow.org/`, Software available from tensorflow.org.

(41) Chen, T.; Li, M.; Li, Y.; Lin, M.; Wang, N.; Wang, M.; Xiao, T.; Xu, B.; Zhang, C.; Zhang, Z. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. In Neural Information Processing Systems, Workshop on Machine Learning Systems. 2015.

(42) Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; Lerer, A. Automatic Differentiation in PyTorch. NeurIPS Autodiff Workshop. 2017.

(43) Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; Siskind, J. M. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research* **2018**, *18*, 1–43.

(44) Larsen, A. H.; Mortensen, J. J.; Blomqvist, J.; Castelli, I. E.; Christensen, R.; Dułak, M.; Friis, J.; Groves, M. N.; Hammer, B.; Hargus, C. The atomic simulation environmenta Python library for working with atoms. *Journal of Physics: Condensed Matter* **2017**, *29*, 273002.

(45) Han, B.; Liu, Y.; Ginzinger, S. W.; Wishart, D. S. SHIFTX2: significantly improved protein chemical shift prediction. *Journal of biomolecular NMR* **2011**, *50*, 43.

(46) Clevert, D.; Unterthiner, T.; Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 2016.

Figure 4: Table of Contents graphic