Analytic derivatives of the trajectory with respect to forcefield parameters.

An exact solution for fitting parameters to ensemble observables.

Yutong Zhao (proteneer@gmail.com), unaffiliated

1 Introduction

When we run molecular dynamics to generate conformations along a trajectory, we almost always generate observables derived from the ensemble. Example observables span from simple quantities such as pseudo-electron density in XRD/ED refinement, density, radii of gyration, Helmholtz free energies, distance information arising from NMR, etc. Nearly always, the experimentally measured observable and the predicted observable don't agree with each other. This results in the painstaking effort of identifying new forcefield functional forms and/or fitting their parameters. Note that this paper is not concerned with the former (discovery of functional forms), but rather, on how to fit the parameters used by these functional forms to condensed phase properties that arise from a simulation (which often span from simple fixed-charged models, all the way to more complicated polarizable forcefields).

In stark contrast to training techniques of modern machine learning, forcefield parameter fitting in MD has been sort of a black-art. At a high level, the parameter tuning procedure is typically separated into two parts: 1) fitting to gas phase QM energies across a varying level of theories either by energy or force-matching, which initializes parameters into a sane region of parameter space, 2) followed by a difficult procedure of fitting to condensed phase properties in a variety of solvent and/or crystallographic conditions. Typically, the second step requires repeated simulations to generate predicted observables from the ensemble. However, there lacks an analytic derivative of the observables to parameters partially owing to complexity of generating necessary derivatives.

Fortunately, with the advent of sophisticated auto differentiation systems in modern machine learning frameworks, it has become not only possible, but also practical to generate these derivatives. We introduce a new molecular dynamics tool with two primary aims:

First, the ability to quickly test new functional forms, as well as its derivatives. Ideally, the user should only need to specify the functional form of the energy function, and all subsequent higher order derivatives should be automatically generated. As an example, OpenMM uses symbolic differentiation (lepton) that's then JIT compiled into the respective kernels. However, symbolic differentiation has severe drawbacks in that it scales poorly with increased complexity of functional forms. In contrast, forward/reverse-mode autograd systems uses dual-algebra, which tends to scale better with complexity as well as naturally lending itself to higher order derivatives. This should enable us to rapidly prototype novel functional forms, ranging from polarizable models to neural-network potentials.

Second, the ability to analytically fit these parameters to observables derived from a simulation trajectory. We show that one can, without too much difficulty, implement analytic derivatives through Langevin dynamics provided one has access to analytic hessians and second-order mixed partial derivatives of the energy function. This allows for optimization of not only the location of the minimas, but also the curvature along the path to reach said minimias. This approach is in contrast to other approaches such as thermodynamic gradients (assuming the ergodic limit has been reached), which uses only the first order derivative of the energy with respect to the parameters. In practice, the implementation is realized in two parts through the chain rule, the first being the derivative of the observable with respect to each conformation in the ensemble, the second being the derivative of a conformation in the trajectory with respect to the forcefield parameters.

2 Mathematical Formalism

Define the loss function $L(O(\boldsymbol{X}_{label}), O(\boldsymbol{X}_{md}))$, where O is a function that calculates some observable from an ensemble. $\boldsymbol{X}_{md}(\boldsymbol{x}_0; \theta)$ is a set of conformations derived from a molecular dynamics simulation, starting from some initial state $\boldsymbol{x}_0$ with some set of forcefield parameters θ. We can also write $\boldsymbol{X}_{md}$ as a collection of individual states $\boldsymbol{x}_t$ at various times t. The goal is to analytically derive $\dfrac{\partial L}{\partial \theta}$ so we can train θ to some experimental observable $O(\boldsymbol{X}_{label})$ by using a standard MSE loss:

$$L(O(\boldsymbol{x}_{label}), O(\boldsymbol{x}_t)) = [O(\boldsymbol{x}_{label}) - O(\boldsymbol{X}(\boldsymbol{x}_0; \theta))]^2 \tag{1}$$

Training the parameters means we need the gradient of L with respect to the parameters θ, which we compute using the chain rule:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \boldsymbol{X}_{md}} \frac{\partial \boldsymbol{X}_{md}}{\partial \theta} \tag{2}$$

In an unbiased sample of conformations, this is equal to a sum of the derivatives of individual states from the trajectory:

$$\frac{\partial L}{\partial \theta} = \sum_t \frac{\partial L}{\partial \boldsymbol{x}_t} \frac{\partial \boldsymbol{x}_t}{\partial \theta} \tag{3}$$

Each summand is a component-wise matrix-multiply, where both $\partial L/\partial \boldsymbol{x}_t$ and $\partial \boldsymbol{x}_t/\partial \theta$ are $(N, 3)$ matrices. The LHS is the derivative of the loss w.r.t. to a conformation, and the RHS is the derivative of a conformation w.r.t. to the model parameters. The LHS is trivial to compute, so the RHS is the main quantity of interest.

If we were to run Langevin dynamics, the velocity is $\boldsymbol{v}_t$ is updated according to:

$$\boldsymbol{v}_t = a\boldsymbol{v}_{t-1} - b\boldsymbol{\nabla} E(\boldsymbol{x}_{t-1}; \theta) + c\boldsymbol{n}_{t-1} \tag{4}$$

Where $\boldsymbol{\nabla}_x$ is the gradient operator (w.r.t. coordinates), and a, b, c are the coefficents derived from the temperature and the friction, and $\boldsymbol{n}$ is noise sampled from an independent gaussian.

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{v}_t \tau \tag{5}$$

Expanding out the first few terms we get:

$$\begin{aligned}
\boldsymbol{x}_1 &= \boldsymbol{x}_0 + \big(a\boldsymbol{v}_0 - b\boldsymbol{\nabla} E(\boldsymbol{x}_0) + c\boldsymbol{n}_0\big)\tau \\
\boldsymbol{x}_2 &= \boldsymbol{x}_1 + \bigg(a\Big(a\boldsymbol{v}_0 - b\boldsymbol{\nabla} E(\boldsymbol{x}_0) + c\boldsymbol{n}_0\Big) - b\boldsymbol{\nabla} E(\boldsymbol{x}_1) + c\boldsymbol{n}_1\bigg)\tau \\
\boldsymbol{x}_3 &= \boldsymbol{x}_2 + \bigg(a\Big(a\Big(a\boldsymbol{v}_0 - b\boldsymbol{\nabla} E(\boldsymbol{x}_0) + c\boldsymbol{n}_0\Big) - b\boldsymbol{\nabla} E(\boldsymbol{x}_1) + c\boldsymbol{n}_1\Big) - b\boldsymbol{\nabla} E(\boldsymbol{x}_2) + c\boldsymbol{n}_2\bigg)\tau
\end{aligned} \tag{6}$$

Recall we can compute the derivatives with respect to a parameter θ using the total derivative:

$$D_\theta f \equiv \frac{df(x_0, \ldots, x_n; \theta)}{d\theta} = \frac{\partial f}{\partial x_0}\frac{\partial x_0}{\partial \theta} + \ldots + \frac{\partial f}{\partial x_n}\frac{\partial x_n}{\partial \theta} + \frac{\partial f}{\partial \theta} \tag{7}$$

We can then compute the equations of motion:

$$\begin{aligned}
\frac{\partial \boldsymbol{x}_1}{\partial \theta} &= -b\tau D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_0) \\
\frac{\partial \boldsymbol{x}_2}{\partial \theta} &= -b\tau\Big(D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_0)(1 + a) + D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_1)\Big) \\
\frac{\partial \boldsymbol{x}_3}{\partial \theta} &= -b\tau\Big(D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_0)(1 + a + a^2) + D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_1)(1 + a) + D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_2)\Big)
\end{aligned} \tag{8}$$

Where,

$$D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_t) = \frac{\partial \boldsymbol{\nabla} E}{\partial \boldsymbol{x}_t}\frac{\partial \boldsymbol{x}_t}{\partial \theta} + \frac{\partial \boldsymbol{\nabla} E}{\partial \theta} \tag{9}$$

Which we can tidy this up into the following expression:

$$\frac{\partial \boldsymbol{x}_t}{\partial \theta} = -b\tau\Big(D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_0;\theta)S^a_{t-1} + D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_1;\theta)S^a_{t-2} + \ldots + D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_{t-1};\theta)\Big) \tag{10}$$

S^a_n is the geometric series of n terms of a. Observe that we only need to do a single forward pass, which can be implemented using forward-mode auto-differentiation. Naively, this procedure needs $O(t)$ memory and $O(t^2)$ runtime since we need to store $\partial \boldsymbol{x}_i/\partial\theta$ and traverse back in time as they're re-weighted after each step. But this would catastrophic for all practical purposes. Fortunately, we can immediately leverage the geometric series, whose rapid convergence implies we need to store the ζ terms for only a small numbers of the most recent steps, since for large t:

$$D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_0)(1 + a + a^2 + ... + a^t) \approx D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}_0)(1 + a + a^2 + ... + a^{t-1}) \tag{11}$$

Suppose it takes k steps for the geometric series converge to numerical accuracy. This allows us to maintain a rolling window of only size $O(k)$ for unconverged ζ values. The converged ζ terms can be reduced and thus require only $O(1)$ extra space. In practice, $k \approx 4000$, so the over all algorithm becomes becomes $O(1)$ in memory and $O(t)$ in time.

3 Periodic Boundary Conditions

If our energy function also depends on a set of box vectors $\boldsymbol{b}$ then we need to augment our equations of motion with:

$$\boldsymbol{b}_t = \boldsymbol{b}_{t-1} - \boldsymbol{\nabla}_{\boldsymbol{b}} E\tau \tag{12}$$

Here $\boldsymbol{\nabla}_{\boldsymbol{b}}$ is the gradient operator with respect to the box vectors $\boldsymbol{b}$. Similar to the above, we can also compute the parameter derivatives of the box vectors through time:

$$\frac{\partial \boldsymbol{b}_t}{\partial \theta} = -\tau\Big(D_\theta \boldsymbol{\nabla}_{\boldsymbol{b}} E(\boldsymbol{x}_0, \boldsymbol{b}_0;\theta) + D_\theta \boldsymbol{\nabla}_{\boldsymbol{b}} E(\boldsymbol{x}_1, \boldsymbol{b}_1;\theta) + \ldots + D_\theta \boldsymbol{\nabla}_{\boldsymbol{b}} E(\boldsymbol{x}_{t-1}, \boldsymbol{b}_{t-1};\theta)\Big) \tag{13}$$

We should also take care to update all the total derivatives to account for the box vectors:

$$D_\theta \boldsymbol{\nabla} E(\boldsymbol{x}, \boldsymbol{b}) = \frac{\partial \boldsymbol{\nabla} E}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \theta} + \frac{\partial \boldsymbol{\nabla} E}{\partial \boldsymbol{b}}\frac{\partial \boldsymbol{b}}{\partial \theta} + \frac{\partial \boldsymbol{\nabla} E}{\partial \theta} \tag{14}$$

Similarly,

$$D_\theta \boldsymbol{\nabla}_{\boldsymbol{b}} E(\boldsymbol{x}, \boldsymbol{b}) = \frac{\partial \boldsymbol{\nabla}_{\boldsymbol{b}} E}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \theta} + \frac{\partial \boldsymbol{\nabla}_{\boldsymbol{b}} E}{\partial \boldsymbol{b}}\frac{\partial \boldsymbol{b}}{\partial \theta} + \frac{\partial \boldsymbol{\nabla}_{\boldsymbol{b}} E}{\partial \theta} \tag{15}$$

Note that:

$$\frac{\partial \boldsymbol{\nabla} E}{\partial \boldsymbol{b}} \equiv \frac{\partial \boldsymbol{\nabla}_{\boldsymbol{b}} E}{\partial \boldsymbol{x}} \tag{16}$$

4 Issues and Caveats

The first major issue has to do how training proceeds. The above procedure needs to be repeated multiple times for the parameters to converge. That is, each iteration would require us to re-run an entire MD simulation since we can't easily re-use the intermediate values. For some test systems it can take anywhere from 50 to 500 rounds of simulations for a single system to be fully optimized (internal data).

A second more practical issue deals with the computational complexity and speed. The analytically dense Hessian is an inherently $O(N^2)$, which is a major increase from the standard $O(NlogN)$ in MD. While the bonded terms have sparse diagonal Hessians, it is unclear if the nonbonded hessians can be sparsified. Furthermore, because the derivatives are automatically generated, and that it's using tensorflow underneath the hood, the simulation itself is significantly slower than a highly optimized MD package. However, with recent advances in XLA-JIT and the flexibility of implementing custom ops, we hope to reconcile this to within an order of magnitude.

5 Code

The code is publicly available at www.github.com/proteneer/timemachine and licensed under the Apache V2.