The Time Machine

A new molecular dynamics engine

Yutong Zhao (proteneer@gmail.com)

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, Helmholtz free energies, distance information due to NOEs from NMR experiments, and even very simple ones such as the radius of gyration. Nearly always, the observed observable and the predicted observable don't agree with each other. This results in the painstaking effort of fitting simulation parameters, which includes both forcefield parameters such as torsional terms, as well as hyperparameters such as the lambda schedule. Despite known pathologies, the functional forms and their parameters in classic MD forcefields have barely changed in decades, partially due to the immense difficulty in fitting their parameters.

In stark contrast to training techniques of modern machine learning, MD looks sclerotic. Fortunately, with highly sophisticated auto differentiation systems, it has become not only possible, but practical to implement an MD engine from scratch. As with any large-scale engineering project, undertaking a rewrite is no small feat, and must be fully justifiable in its time and effort spent. Thus, the two major goals of the Time Machine rewrite are as follows:

First, the ability to quickly test new functional forms, as well as its derivatives. This should be a major improvement over all existing systems. For example, OpenMM uses symbolic differentiation (lepton) that's then JIT compiled into the respective kernels, which has very different performance characteristics than modern forward/reverse-mode autograd systems using dual-algebra. Furthermore, we want the ability to not only derive forces, but also hessians and second order mixed partials for reasons soon to be explained. This will enable us to play with far more sophisticated functional forms, such as neural network-potentials, semi-empirical QM/MM methods, and even simpler replacements such as Morse-typed bonds.

Second, the ability to rapidly fit the parameters of our functional forms, new and old, to observables derived from the simulation trajectory. It turns out that one can, without too much difficulty, implement analytic derivatives of the trajectory with respect to the parameters, provided one has access to hessian and mixed partial derivatives. One can essentially leverage forwardpropagation techniques (opposite of backpropagation) to carry the derivatives over.

Nevertheless, there are major trade-offs we'd have to consider carefully:

First, it will be significantly slower on a per time step basis than all existing MD packages. This could be anywhere from 10x to 100x slower since it will be extremely difficult to beat hand-tuned GPU kernels. We hope to mitigate this by incorporating more advanced XLA-JIT techniques to generate fused kernels (similar to OpenMM) down the road, as well as being able to fit a set of parameters that allows for much bigger time steps.

Second, if we start changing the functional forms, we need to bootstrap parameters that allow the system to be somewhat stable to start with. This will require us to implement something akin to force-matching against large-scale gas phase QM energies and forces. Fortunately we already have the mixed partial derivatives for free, so we can do this in the context of our framework.

2 Mathematical Formalism

Define the loss function $L(O(\vec{x}_{label}), O(\vec{x}_t))$, where O is an observable, $\vec{x}_t$ is a single conformation at some time t of a molecular dynamics simulation, starting from some initial state $\vec{x}_0$. Denote the set of forcefield parameters to be θ. The goal is to analytically derive a gradient for $\dfrac{\partial L}{\partial \theta}$ so we can train our parameters θ to some observable. Note that in practice, observables are really functions of an ensemble of structures, but for the time being, and without loss of generality, we define them to be a function of a single conformation.

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

We want to compute the gradient of L with respect to the parameters θ using the chain rule:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \vec{x}_t} \frac{\partial \vec{x}_t}{\partial \theta} \tag{2}$$

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

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

$$\vec{v}_t = a\vec{v}_{t-1} - b\nabla E(\vec{x}_{t-1};\theta) + c\vec{n}_{t-1} \tag{3}$$

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

$$\vec{x}_t = \vec{x}_{t-1} + \vec{v}_t\delta t \tag{4}$$

Expanding out the first few terms we get:

$$\begin{aligned}
\vec{x}_1 &= \vec{x}_0 + \big(a\vec{v}_0 - b\nabla E(\vec{x}_0) + c\vec{n}_0\big)\delta t \\
\vec{x}_2 &= \vec{x}_1 + \Big(a\Big(a\vec{v}_0 - b\nabla E(\vec{x}_0) + c\vec{n}_0\Big) - b\nabla E(\vec{x}_1) + c\vec{n}_1\Big)\delta t \\
\vec{x}_3 &= \vec{x}_2 + \Bigg(a\Big(a\Big(a\vec{v}_0 - b\nabla E(\vec{x}_0) + c\vec{n}_0\Big) - b\nabla E(\vec{x}_1) + c\vec{n}_1\Big) - b\nabla E(\vec{x}_2) + c\vec{n}_2\Bigg)\delta t
\end{aligned} \tag{5}$$

We can then immediately differentiate with respect to the parameters, if first, for convenience, we define the ζ operator:

$$\zeta E(\vec{x}_i;\theta) \equiv \frac{\partial^2 E}{\partial \vec{x}_i^2}\frac{\partial \vec{x}_i}{\partial \theta} + \frac{\partial^2 E}{\partial \vec{x}_i \partial \theta} \tag{6}$$

To demystify this expression, the first summand is a tensor contraction of a rank-4 hessian (N,3,N,3) with a rank-2 derivative (N,3). The second summand is a rank-2 mixed partial derivative (N,3). We can write out the first few terms of the derivative using the ζ operator:

$$\begin{aligned}
\frac{\partial \vec{x}_1}{\partial \theta} &= -b\delta t\zeta E(\vec{x}_0) \\
\frac{\partial \vec{x}_2}{\partial \theta} &= -b\delta t\Big(\zeta E(\vec{x}_0)(1+a) + \zeta E(\vec{x}_1)\Big) \\
\frac{\partial \vec{x}_3}{\partial \theta} &= -b\delta t\Big(\zeta E(\vec{x}_0)(1+a+a^2) + \zeta E(\vec{x}_1)(1+a) + \zeta E(\vec{x}_2)\Big)
\end{aligned} \tag{7}$$

Finally, we can tidy this up into the following expression:

$$\frac{\partial x_t}{\partial \theta} = -b\delta t\Big(\zeta E(\vec{x}_0;\theta)S^a_{t-1} + \zeta E(\vec{x}_1;\theta)S^a_{t-2} + \ldots + \zeta E(\vec{x}_{t-1};\theta)\Big) \tag{8}$$

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 since we need to store $\partial\vec{x}_i/\partial\theta$ as they're re-weighted after each step. But this would catastrophic for all practical purposes. Fortunately, the beauty of the Langevin dynamics reveals itself via a 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:

$$\zeta E(\vec{x}_0)(1 + a + a^2 + ... + a^t) \approx \zeta E(\vec{x}_0)(1 + a + a^2 + ... + a^{t-1}) \tag{9}$$

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 while using an extra $O(1)$ space for the converged ζ terms. In practice, $k \approx 4000$, so the over all algorithm becomes far more practical.

With this machinery in place, we can now use some additional tricks to regularize the fitting process, such as enforcing that for all ergodic systems, the time averaged observable must be equal to the ensemble averaged observable. The exact method is that we use reservoir sampling to draw from the time average, and the last time slice of a replica simulation to generate the ensemble average.