BT5420: Computer Simulations of Biomolecular Systems
Advances in MD Force Field, Simulation or Analysis Technique
Project-II Report

Sapna R
BS20B032

1. Title of the Paper:
sGDML: Constructing accurate and data efficient molecular force fields using machine learning
https://doi.org/10.1016/j.cpc.2019.02.007

The objective of the paper

- This paper aims to present an optimized implementation of the **Symmetric Gradient Domain Machine Learning interfaced ForceField** model incorporating spatio-temporal physical symmetries onto the model to describe any complex physical interaction system
- With respect to the implementation of the **sGDML** model, the main focus has been in providing compact working examples for molecular systems with a few dozen atoms(paracetamol) in an accessible programming language achieving high performance in terms of speed using state of art technologies including parallel processing support for **multi-Core CPUs** and **multi-GPU** supporting environments .

Reason for choosing this Paper

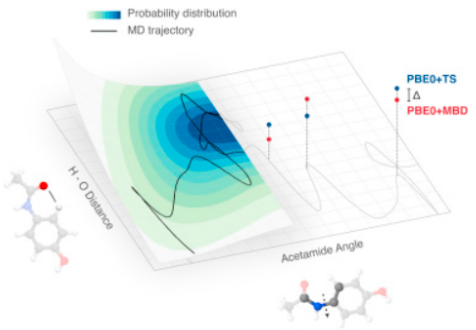
- In atomistic MD simulations, it is often challenging to choose between highly accurate yet time consuming **Ab-initio methods** derived from solving Schrodinger's equations and relatively faster **classical MD force fields** which display limited accuracy .
- This is where ML based forcefields play a crucial role in bridging the gap between the 2 methods by using data analysis approaches to find the statistical correlation between Potential Energy and a particular conformation of the molecular system without any preconceived notion of the **PES** .
- In particular, the paper I had chosen implements the sGDML model which functions at a speed only 3 orders lower than that of conventional Forcefield methods.
- While searching papers on advanced techniques used in MD simulations , the main factors I had considered was the ease with which some results can be

reproduced and how much relevance the chosen paper had in providing future directions to take forward the presented research.

- The paper I had chosen used state of art implementations of a highly data efficient model that could resemble polarisable force fields much better than classical force fields like **AMBER,CHARM** and **GROMACS** in terms of speed.
- The paper provides a user-friendly python package with easy to implement, reconstruct or query the sGDML model where the only input provided by the user is a small set of reference geometries with the corresponding Forcefields and energy.
- Another user-friendly feature includes a **command line interface sgdml** that guides the user through the entire process of reconstructing sGDML model. Thus the software provides a good starting point for anyone with not much domain knowledge in sGDML model as well.

iii. Describe the technique

Sampling Training Set



- The first step in ff construction involves generating a minimal training set capturing required geometrical configurations of the molecular system.
- The planned simulations are confined only in the configurational space spanned by the training data to ensure that any unreliable predictions are avoided.
- A sufficiently long MD trajectory analysis is done at temperature > 500k to provide an optimal coverage of the configurational space.
- The actual training set sampled from the trajectory have their energies closely following the **Maxwell-Boltzman** distribution.
- The **DFT+PBE+TS** trajectory is used to perform geometric sampling which is computationally much less expensive than the **DBT+PBE0+MBD** which requires a longer time run for MD simulation with a larger basis set.
- The force and energy labels only for the sampled points are recomputed using the **DBT+PBE+MBD** level of theory.

- The sampling scheme mentioned above produces a PES which is a good approximation of the PES that would be generated had the **DBT+PBE0+MBD** surface as overfitting approximations would result in losing out important features
- The next important step involves designating the appropriate time step for the MD simulations so that the configurational space is sampled with the correct probability.
- Typically, a time step equal to one tenth of the oscillator with highest frequency is chosen.In the provided example (paracetamol) the highest frequency observed was 3600 cm^{-1} equivalent to 9.3 fs. Thus a time step of 1fs was selected in the working example.
- The simulated trajectory is obtained as a dataset file with the sampled data points and other columns corresponding to force and energy in an extended xyz format
- While composing the above mentioned input data files, it is important to ensure consistency in the units of force & energy as the sGDML model is unit agnostic.
- The xyz format file is converted to an **sGDML binary format file** which typically is the default numpy binary file using the command given below.

```
$ sgdml_dataset_from_xyz.py paracetamol.xyz
```

- The user can then operate the program using CLI or Command Line interface or directly work with Training and Predict Modules in the python interface

Using Command Line Interface

- With the resulting **data file .npz** ,the automated **sgdml** training assistants run to obtain a completely trained and tested model

```
$ sgdml all d_paracetamol.npz 1000 500
```

```
$ sgdml all <dataset_file> <n_train> <n_validate>\
          [<n_test>] [--sig <list_or_range>]
```

- The arguments 1000 and 500 denote the number of training data points and no of geometries to be used for validation. The range of values for the hyperparameter sigma can be specified using the argument - -sig
- The number of test data points need not be mentioned explicitly.The training assistant can automatically split the bulk dataset into training module, validation module & test module ,train the models for a number of **hyperparameter candidates** ,validate without overlap unless individual datasets for overlap are specified and pick the most accurate model followed by testing the same .
- In the provided working example the output file after the above mentioned steps are stored in m_paracetomol.npz.

```
import numpy as np
from sgdml.predict import GDMLPredict

model = np.load('m_paracetamol.npz')
gdml = GDMLPredict(model)
```

The obtained file in the previous step can be used to interface the ff with existing applications.

```
e,f = gdml.predict(r)
```

- Predictions are made using the above command

Using Training and Predict Modules

- Reconstruction of the Forcefield and evaluation of the same is organized into 2 independent modules in the python interface namely; train & predict
- The training module is packaged with required routines for sampling reference data,model parameterization and recovery of symmetry.
- Every sGDML model comes with a batch of training tasks packaging the configurations of the training run including the range of indices for training and validation points,the permutational symmetries of the molecule and the range of hyper parameter values. This is generated using the below command.

```
$ sgdml create <dataset_file> <n_train> <n_valid> \
             [-sig <list_or_range>]
```

A directory containing the corresponding task files is set up. The parameters/arguments used here are the same as the ones used in the CLI.

- The **train** command is used to invoke training in the set up directory.

```
$ sgdml train <task_dir_or_file>
```

The model candidates corresponding to each task are stored in the tasks directory.

- Post training, each model is validated for performance. The validation process is invoked by the following command.

```
$ sgdml validate <model_dir_or_file> <dataset_file>
```

- The RMSE is the metric used for evaluating the best performing model candidate.The best model from the full set of candidates is kept using the below command.

```
$ sgdml select <model_dir>
```

- Finally, the performance of the model on the test set is carried out using the below command

```
$ sgdml test <model_file_or_dir> <dataset_file> \
            [<n_test>]
```

- The light weight model files generated by the training module contains preprocessed requisites for evaluating forcefield.
- The results obtained from the training module are then independently instantiated and queried using the predict module to integrate into external packages.

```
gdml_predict = GDMLPredict(model,\ [chunk size],\ [num_processes])
```

- Further, the parameters chunk size and num_processes can be automatically tuned to achieve optimal performance in parallel processing using the below command

```
gdml_predict.set_opt_parallelism()
```

- The force and energy predictions for a given geometry are then calculated using the below command.

```
r,_ = io.read_xyz(geometry_path)
e,f = gdml_predict.predict(r)
```

- The predict module is designed minimalistic to contain the only logic to evaluate the force fields and energies for a given set of reference geometries.
- Such a modularisation enables the training to be centralized on a high performance computer and the full fledged model can be efficiently integrated into any forcefield requiring application .

Here's an example of a how to train a sgdml model without validation or testing via python API

```
import sys
import numpy as np
from sgdml.train import GDMLTrain

n_train = 500
train_dataset = np.load(<train_dataset_path>)

n_train = 1000
test_dataset = np.load(<test_dataset_path>)

gdml = GDMLTrain(model)

task = gdml.create_task(train_dataset, n_train,\
                        test_dataset, n_test,\
                        sig=10, lam=1e-15)

try:
  model = gdml.train(task)
except Exception, err:
  sys.exit(err)
else:
  np.savez_compressed("model.npz", **model)
```

- The python software is interfaced with 2 commonly used forcefield simulation engines;a **Calculator** for ASE and **i-PI** interface

ASE

- The **A**tomic **S**imulation **E**nvironment is a collection of python tools that aid in atomistic simulations including classical MD, Structure optimization and nudged elastic band calculations.sGDML is integrated into ASE as a calculator object.The sGDML calculator object is imported from the SGDMLCalculator package and the initialized with the path to the model file

```
from sgdml.intf.ase import SGDMLCalculator
sgdml = SGDMLCalculator(m_paracetamol.npz)
```

- It is then attached to any **Atoms** object that matches the molecular geometry represented by the model file.

```
mol = read_xyz(paracetamol.xyz).next()
mol.set_calculator(sgdml)
```

i-PI interface

- The i-PI software provides a user-friendly implementation of the Path Integral Molecular Dynamics Simulation with the advanced integrators and thermostats. To interface the sGDML model as a forcefield object, it is necessary to make some changes in the forcefield.py present in the directory ipi/engine to which sGDML is added as a new interface class. It is also necessary to make changes in the ipi/inputs/forcefield.py and ipi/inputs/simulation.py to include FFsGDML as an additional parameter.
- The sGDML force field is specified as an input in the .xml format.

```
<ffsgdml name="sgdml">
</ffsgdml>
<system>
...
  <forces>
    <force forcefield="sgdml"> </force>
    sGDMLmodel="sGDML_model_name.npz"
  </forces>
...
</system>
```

  Where sgdml is the name of the appropriate model file.
- The input files input.xml, <file_name>_input_geometry.xyz and the m_<filename>.xyz is required for MD simulation which is invoked by the below command.

```
$ python i-pi input.xml
```

The **Summary** of the entire process is shown in the below figure.

iv. Explain the background theory

- Training a **GDML** model requires the following objective function to be minimized over a training set comprising M points.

$$\mathcal{L}(\mathbf{\Omega}) = \sum_{i}^{M} (\mathbf{J}_{\Phi_i}\vec{\omega}_i - \mathbf{F}_i)^2 + \lambda \|\mathbf{\Omega}\|^2$$

- Here, the $J_{\Phi i}$ = $J_{\Phi(\sim xi)}$ denotes the the (3N x F) jacobian matrices of a non linear transform of the training geometries xi into the F dimensional space weighted by

vectors $\vec{w}_i$.$\mathbf{F}_i$ denotes the vector containing the force acting on each geometry. The objective function also includes a regularization term with hyperparameter λ to penalize the norms of **Ω** containing the coefficients $[\vec{\omega}_1^\top, \ldots, \vec{\omega}_M^\top]^\top$

- The minima of the loss function is obtained by setting the derivative to 0.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{\Omega}} = 2\sum_i^M \mathbf{J}_{\Phi_i}^\top (\mathbf{J}_{\Phi_i}\vec{\omega}_i - \mathbf{F}_i) + 2\lambda\mathbf{\Omega} = \vec{0}$$

- Thus, we obtain the below solution

$$\mathbf{\Omega} = \left(\lambda \mathbb{I}_F + \sum_i^M \mathbf{J}_{\Phi_i}^\top \mathbf{J}_{\Phi_i}\right)^{-1} \sum_j^M \mathbf{J}_{\Phi_j}^\top \mathbf{F}_j$$

- Now the individual Jacobian matrices are combined to obtain J_Φ as [$J_{\Phi 1}$, . . . , $J_{\Phi M}$] of dimensions 3NM x F simplifying the solution to the below equations using the woodbury matrix identity.

$$\begin{aligned}\mathbf{\Omega} &= (\mathbf{J}_\mathbf{\Phi}^\top \mathbf{J}_\mathbf{\Phi} + \lambda \mathbb{I}_F)^{-1} \mathbf{J}_\mathbf{\Phi}^\top \mathbf{F} \\ &= \mathbf{J}_\mathbf{\Phi}^\top \underbrace{((\mathbf{J}_\mathbf{\Phi}\mathbf{J}_\mathbf{\Phi}^\top + \lambda \mathbb{I}_{3NM})^{-1}\mathbf{F})}\end{aligned}$$

- The above simplification is done so as to solve the linear system much faster(3NM<<F)
- The F_{new} is then calculated by the below equation

$$\mathbf{F}_{\text{new}} = \mathbf{J}_{\Phi_{\text{new}}} \mathbf{J}_\mathbf{\Phi}^\top \mathbf{A}.$$

$$\mathbf{J}_\mathbf{\Phi} = \nabla \mathbf{\Phi}^\top$$

- The jacobian is rewritten as the outer product of the derivative operator & feature transform.

$$\begin{aligned}\mathbf{J}_\mathbf{\Phi}\mathbf{J}_\mathbf{\Phi}^\top &= \nabla\mathbf{\Phi}^\top(\nabla\mathbf{\Phi}^\top)^\top \\ &= \nabla \underbrace{\mathbf{\Phi}^\top\mathbf{\Phi}}_{\kappa} \nabla^\top\end{aligned}$$

- The inner product of the feature transform is substituted by a scalar valued kernel function k.
- Thus the elements of the forcefield kernel in GDML are given by the below entries.

$$(\kappa)_{ij} = \partial^2\kappa / \partial\vec{x}_i \partial\vec{x}'_j.$$

- If k is stationery , we have

$$\frac{\partial^2\tilde{\kappa}}{\partial\vec{x}\partial\vec{x}'} = \frac{\partial^2\tilde{\kappa}}{\partial\tau^2}\frac{\partial\tau}{\partial\vec{x}}\frac{\partial\tau}{\partial\vec{x}'} = -\frac{\partial^2\tilde{\kappa}}{\partial\tau^2}\left(\frac{\partial\tau}{\partial\vec{x}}\right)^2 = \frac{\partial^2\tilde{\kappa}}{\partial^2\vec{x}}.$$

- Finally the forcefield and energy equations are given by

$$\hat{\mathbf{f}}_\mathbf{F}(\vec{x}) = \sum_i^M \sum_j^{3N} (\vec{\alpha}_i)_j \frac{\partial}{\partial x_j} \nabla_{\vec{x}} \kappa(\vec{x}, \vec{x}_i) \qquad \hat{f}_E(\vec{x}) = \sum_i^M \sum_j^{3N} (\vec{\alpha}_i)_j \frac{\partial}{\partial x_j} \kappa(\vec{x}, \vec{x}_i) + c.$$

where $\mathbf{A} = [\vec{\alpha}_1^\top, \ldots, \vec{\alpha}_M^\top]^\top$

- The best approximate to the integration constant c is given by

$$c = \frac{\sum_i^M e_i + \hat{f}_E(\vec{x}_i)}{M}$$

obtained by minimizing the loss function,

$$\mathcal{L}(c) = \sum_i^M (\int \hat{\mathbf{f}}_{\mathbf{F}}(\vec{x}_i)\,\mathrm{d}x - e_i)^2$$

- The **sgdml** is an extension of the gdml model which automatically incorporates the **rigid space group symmetries** (eg rotation translation etc) and **fluxional space group symmetries** (methyl group symmetries) from the training set of molecular geometries and exercises them within the kernel function.

,

$$\mathrm{Hess}(\kappa_{\mathrm{sym}})(\vec{x},\vec{x}') = \frac{1}{S}\sum_{pq}^{S} \mathbf{P}_p^\top \mathrm{Hess}(\kappa)(\mathbf{P}_p\vec{x}, \mathbf{P}_q\vec{x}')\mathbf{P}_q$$

where the S symmetries are represented by the permutation matrices P acting on the atoms of the molecule.

- The above Hessian kernel matrix can be obtained wrt to polar coordinate using the kernel description **x=** D(**r).** The bold denotes the vectors here.

$$\mathrm{Hess}(\kappa_{\mathrm{sym}})_{\mathbf{D}} = \frac{1}{S}\sum_{pq}^{S} (\mathbf{J}_{\mathbf{D}_{\mathbf{P}_p}}\mathbf{P}_p)^\top \mathrm{Hess}(\kappa)(\mathbf{D}_{\mathbf{P}_p}, \mathbf{D}'_{\mathbf{P}_q})\mathbf{J}_{\mathbf{D}'_{\mathbf{P}_q}}\mathbf{P}_q$$

- In GDML and sGDML, matrices of pairwise inverse distance between atoms are constructed to describe molecular graphs in a translation & rotation invariant way.

$$(\mathbf{D})_{ij} = \begin{cases} \|r_i - r_j\|^{-1} & \text{for } i > j \\ 0 & \text{for } i \leq j \end{cases}$$

- The Jacobian J_D is composed of a vectorised descriptor of derivatives of **r** with respect to the cartesian coordinates is given by

$$(\nabla_{r_i}\mathbf{D})_{ij/ji} = \begin{cases} (r_i - r_j)\|r_i - r_j\|^{-3} & \text{for } i > j \\ 0 & \text{for } i \leq j \end{cases}.$$

v. The advantages and disadvantages of the technique

Advantages:

- Highly Data efficient
- Unlike traditional ffs, the proposed model imposes no hypothesized interaction pattern and can model any complex interaction
- sGDML model can reach spectroscopic accuracy in predicting energy for a small molecular system like benzene or toluene.

- Energies and Forces are calculated at nearly 4-8 orders of magnitude greater than (Coupled Cluster method) CCSD & (Density Functional Theory) DFT respectively. sGDML is only 3 orders of magnitude slower than conventional force fields in comparison to classical force fields like AMBER,CHARM, GROMACS etc .
- The sGDML model implementations enables ad hoc reconstruction of the Potential Energy Surface based on the training data set provided giving new insights in studying complex patterns of physical interactions wherein the ab-initio implementation is computationally too expensive
- With the sGDML model behaving like traditional FF,a variety of PES sampling tasks can now be performed including molecular dynamics, computation of transition paths, vibrational analysis etc.

Disadvantages:

- The sGDML model works well only for molecules with a few dozen atoms. Thus, there is a long way to go in updating the model for biomolecular systems with a few 1000 atoms.
- The cost of transferability is high. The model trained on one particular conformer cannot be used to obtain insights on another conformer.
- Memory Consumption is high. Analytic Solvers require the kernel matrix comprising of $(3MxN)^2$ double precision entries of 8 bytes each to be stored at once which

vi. Connect the involved concepts to the what you have learnt during this course

- Gradient Vectors
- Linear algebra concepts including Hessian matrix computation and Jacobian Transform in evaluating the kernel matrix and non linear transform of the training geometries xi into the F dimensional space.
- Choosing time step for MD simulation
- Intuitions about conventional force fields AMBER, CHARM and GROMACS
- Forcefield parameterisations and variations