ML-Waterbox Notes

Introduction

Contact Information

For any questions on the content of this report or of the repository, please contact Owen Lockwood at lockwo@rpi.edu. Issues with the repository can be made, feel free to recommend any changes, or to fork the repository and make your own (please ensure you follow the guidelines set forth by the license).

Contents of this Report

This report contains information concerning the work done in and around the ML-Waterbox repository. This report is up to date as of April 2019.

Use of the Work

Overview of Project

The goal of this project (is) (was) to be a first foray into the application of machine learning to the field of molecular simulations for the Garde Group. This project focused on the use of machine learning to provide a new methodology for analyzing simulations of liquid water. This repository contains many of the important files surrounding this project.

Project Details

Now I will detail more of the nitty gritty details of the project. Fundamentally there were two different physical/structural properties being investigated: cavity formation and tetrahedral order parameter.

Cavity Formation

Within the investigation of cavity formation (which has been worked on more as of yet) there are a number of "subprojects" and different problems within this overarching idea. After running a liquid water simulation (in my case 4x4x4 nm box of 2180 SPC/E water molecules), it must then be processed in FORTRAN. To generate the input into the machine learning algorithm there are two different approaches: nearest and box. The first approach, nearest, goes through grid of the water box and finds the nearest number (20) water molecules and writes their coordinates and whether there is a cavity of a specified size (0.2nm) at that grid point. There is clearly a lot of room for customization here. The second approach is the box approach. This approach we believed to be more potentially applicable (and computationally cheaper). In the FORTRAN code for this technique the simulation box is split into a specified number of sub-boxes. Each water molecule is then placed into one of these boxes and the existence of a cavity of a certain size is calculated and the actual cavity size at the center is also calculated. Note: this information is only calculated for the "grid point" at the center of the box. Hence, the data generation of this process is well below the theoretical optimum, and this lost increases with box size. Improvements could easily be implemented into the box FORTRAN code, if one desires to improve data saturation. These two different FORTRAN codes produce files that can be read in the two different types of machine learning algorithms: binary classification and regression. The binary classification

problem was the first thing to be completed and the least likely to be practically useful (in this authors opinion). Both the box and nearest approaches could be fed into this problem. However, they would yield significantly different results. This is because the nearest coordinates had a fixed number of input coordinates, but the box approach has a varying number of water molecules in a given box and thus must be padded and therefore has varying input sizes. As a general statement, the padded data (or variable input data) performs worse universally (although it doesn't necessarily perform bad, just worse). Fundamentally, the binary classification problem asks the algorithm, is there a cavity of X size at this point (yes or no)? The regression problem which only the box code can be fed into (not from a theoretical limitation but the nearest code just doesn't have the output programed to include the actual cavity distance). The regression problem basically asks what is the size of the cavity at this gird point? While the nearest FORTRAN code doesn't include this aspect, in the python code for this, there are conversions available (i.e. fixing the number of coordinates to a nearest number).

Tetrahedral Order Parameter

Work on this front is ongoing. In theory this seems to be more suited for the task of machine learning. The computational expense of the traditional calculations is significantly higher than that of cavity formation, and it relies on a constant number of coordinates (which traditionally perform better). Problems in the FORTRAN code have hindered progress, and at the time of this report, the tetrahedral code is finished but may contain problems. The python code exists that the FORTRAN code can be "plugged in" to, but very limited testing has been done. The tetrahedral order parameter problem represents a regression problem.

Encoding Strategies

For each of these problems there are also 4 different ways to encode the data for the network to train on: no change, categorical, normalized, centered. No change means we feed in the coordinates directly as they are output. Categorical and normalized are standard encoding strategies. Centered involved moving each coordinate as though the starting point of the calculation was the origin (0, 0, 0). The extent to which these encoding strategies have been tested is limited and is room for improvement. For fixed input, categorical and centered have generally been the best (96-99% accuracy) for regression (cavity) centered and normalized have generally been the best. Tetrahedral has not been tested enough to provide any other preliminary notes.

Overview of Repository

The benchmarking folder contains the files used to do some preliminary benchmarking for binary classification of cavities, can be expanded upon. The other files are in the source-code directory. In this directory you will find cavity.f90 (FORTRAN code for nearest), coord.py (the machine learning algorithm to learn from nearest), cv2.pptx (a powerpoint given on this), two makefiles (for the FORTRAN code, they can be adapted for any of the FORTRAN codes), pre.py (a preprocessing data generator for the cavity.f90, should be integrated with the FORTRAN code), test_encode.py (a small testing group for different encodings), tetra.py (python code frontend for the data generated by tetrahedral.f90), and tetrahedral.f90 (the fortran code to general tetrahedral order parameter data). In the subdirectory box, there is box.f90 (that generates the data for the box approach) and box.py (a python code to learn the box approach categorically encoded). Within the different encodings, there are 4 files, onhot.py (categorically encoded, binary classification), regress.py (a sort of testing ground for regression), regresscenter.py (centered encoded regression), and regressnormal.py (normalized encoded regression).

Results and Future

Results for this project were not documented with the detail and care as they should've been. There is not a centralized folder of all results. Due to the ease with which these results can be generated them, I created them on a need to know basis (when I needed to know something, I just tested it). Future work will have a more rigorous standard of documentation. Some results can be seen in the poster. However, if you desire more detailed results I encourage you to run the programs for yourself, it is a quick process to generate results. In terms of the general results, the fixed input cavity formation and regression worked very well, the variable input version of these worked (albeit less well) and the tetrahedral code might work. I do not foresee a direct application of this software, as the benchmarking reveals how much slower it is, but I do foresee potential use of this methodology. See this Group Github page for future work.