PHYS–467 Machine Learning for Physicists

Assignment 3: Predicting Atomization Energies with Neural Nets

Due: December 23, 2022, 23:59

Instructions

- Please, upload a notebook[1] with your code and a report in PDF format with your plots, comments, and answers to the Moodle page of the course. The plots in your report must coincide with the ones appearing in your notebook (you can use a random seed if that may help you with reproducibility). **No delay is accepted**.
- The files should be called `surname_name_{notebook, report}.{ipynb, pdf}`. Use only characters in [a-z] (e.g. à, ç, é $\mapsto$ a, c, e) and use an underscore instead of spaces.
- As for the previous assignments, list whoever you consulted with in your report.

Predicting ground-state molecular properties is a critical task in chemistry. However, solving the Schrödinger equation is computationally intractable even for small systems. Instead, in this homework, you will use a machine learning approach. In particular, you will predict the atomization energies of small organic molecules using a neural network. You will use a subset of the `QM7` dataset. Each molecule in this dataset is represented by an $N \times N$ *Coulomb matrix* M, where N is the number of atoms in the largest molecule. The matrix M is specified by the set of nuclear coordinates $\{R_i\}_{i\leq N}$ and the corresponding charges $\{Z_i\}_{i\leq N}$,

$$M_{ij} = \frac{1}{2} Z_i^{2.4}\, \delta_{ij} + \frac{Z_i Z_j}{\|R_i - R_j\|}\,(1-\delta_{ij}). \tag{1}$$

If a molecule has less than N atoms, the remaining elements of M are set to 0. Notice that the off-diagonal elements correspond to the Coulomb repulsion between different atoms in the molecule, which gives the name to this molecular representation.

1. (1 pt) The dataset is stored as a dictionary in the file `dataset.pt` and is already divided into train, validation, and test sets, with keys `X_train`, `y_train`, `X_val`, `y_val`, and `X_test`, `y_test` respectively. The tensors X's contain the upper triangular parts of the normalized Coulomb matrices, and the tensors y's the atomization energies of the molecules in units of kcal/mol. Open the dataset using `torch.load`.

```
dataset = torch.load("dataset.pt")
```

2. (1 pt) To inspect the labels y's, make a histogram of the atomization energies of the molecules contained in the training set. Then normalize the train, validation, and test labels by subtracting the mean and dividing by the standard deviation of the train labels. This procedure helps to speed up the convergence of the optimization dynamics and to avoid numerical instabilities. Keep track of the mean and the standard deviation that you used to normalize in order to be able to unnormalize the energies once the training is done.

3. (4 pt) Use `torch.utils.data.TensorDataset` to generate the train, validation, and test PyTorch datasets. Then, build the train, validation, and test Pytorch dataloaders, setting the batch size to 100 and activating reshuffling at each epoch for the train data by setting `shuffle=True`.

4. (6 pt) Define a `nn.Module` class for a scalar valued one-hidden-layer fully-connected network $f_{\mathsf{NN}}(x)$ with the appropriate input dimension d, p hidden neurons, and ReLU activation function, i.e.,

$$f_{\mathsf{NN}}(x) = \sum_{i=1}^{p} w_i^{(2)} \sigma\left(x^\top w_i^{(1)} + b_i^{(1)}\right) + b^{(2)}, \tag{2}$$

[1] The notebook format allows you to run your code on Google Colab's GPUs. If you prefer to run the notebook locally, you can add PyTorch to the `ml4phys` environment via `conda install pytorch torchvision torchaudio -c pytorch`.

where $\sigma(x) = \max(0,\, x)$ and $w_i^{(2)},\, b^{(2)} \sim \mathcal{U}\left(-\frac{1}{\sqrt{p}}, \frac{1}{\sqrt{p}}\right)$, $w_{ij}^{(1)}, b_i^{(1)} \sim \mathcal{U}\left(-\frac{1}{\sqrt{d}}, \frac{1}{\sqrt{d}}\right)$. Notice that this is the default initialization used by PyTorch modules.

5. (8 pt) Train the neural network setting $p = 1000$ for 1000 epochs, using the MSE loss (`nn.MSELoss`) and the Adam optimizer (`torch.optim.Adam`). Choose from the set $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}\}$ the learning rate that results in the best generalization performance on the validation set. Averaging over 2-3 different random initializations of the network leads to a better picture and avoids choices based on a poor random initialization, but it is not mandatory. All the following requests refer to the model trained with the chosen learning rate. First, plot and analyze the behavior of the train and validation losses during training[2]. Finally, evaluate the performance of this model on the test set.

6. (2 pt) Plot the energies predicted by this model on the test set vs the true energies in kcal/mol units and compute the unnormalized[3] root mean square error on the test set.

7. (8 pt) Retrain the neural network, varying the number of training examples $n \in \{100, 200, 500, 1000\}$ and using the same architecture, number of epochs, loss, optimizer, and the learning rate you selected in question 5. Plot the learning curve[4], i.e., unnormalized mean squared error on the test set ε_T as a function of the size of the training set n on a log-log scale. What type of curve do you observe? Identify what function fits well $\varepsilon_T(n)$, e.g., $A \exp(-\beta n)$, $A\,(\log_{10} n)^{-\beta}$, $A\, n^{-\beta}$, $-\beta\, n + A$. Find the values of the parameters A, β by manual inspection or using another curve fitting method. Estimate the test error that this model would achieve with $n = 4000$ training points, i.e., compute $\varepsilon_T(n = 4000)$.

8. (Bonus) Another method to improve the generalization of a model, especially when few data are available, is to encode the symmetries of the problem in the algorithm. In this case, the Coulomb matrix representation of the molecules depends on the labeling of the atoms, whereas it would be desirable to be permutation invariant. Indeed, if we label the atoms of the same molecule in two different ways (e.g., we assign the same atom once to the first column and once to the fifth one), we would like our prediction to remain unchanged! Can you propose a way to obtain an algorithm more (or fully) invariant to permutations? You may read and compare your proposition with *Montavon et al. "Learning invariant representations of molecules for atomization energy prediction" (NeurIPS 2012).*

[2] You can track the losses every 5-10 steps in order to avoid slowing down the training too much. As a reference, our training loop over 1000 epochs takes around 30 seconds on Google Colab with hardware acceleration, and twice this time on CPU.

[3] Use the unnormalized predictions and labels.

[4] Again, averaging over 2-3 different random initializations of the network leads to a cleaner plot, but it is not mandatory.