Due date: December 03, 2023
Please submit a pdf with answers (add figures if appropriate)
as well as the notebook used to generate them.

1 Molecular featurization (5 pt)

It's finally time to apply our knowledge of machine learning to molecules. In this exercise, we will train a neural network to learn the total potential energy of an artificial two-dimensional "molecule" depending on the positions of the atoms. There are different ways of how to encode the molecular structure before feeding it to an ML model which is known as *featurization*. We will explore different ways of molecular featurization and discuss their strengths and weaknesses.

As an interaction potential, we use a Coulomb-like pair-potential given by

$$v(\mathbf{r}_i, \mathbf{r}_j) = \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \quad . \tag{1}$$

The total potential energy is accordingly given by the sum of all pairwise interactions:

$$E = \sum_i \sum_{j>i} v(\mathbf{r}_i, \mathbf{r}_j) \quad . \tag{2}$$

1.1 Data generation

In the notebook, you find the "equilibrium" structure (shape `(n_atoms,n_dimensions)`) of the molecule we want to study. First, we want to generate some data to train the model on. In real-life, this data could come from a Molecular Dynamics (MD) or Monte Carlo (MC) simulation, but for this exercise we just noise up the equilibrium structure.

For each of the 4×2 coordinates, generate 500 samples by drawing an array of random samples from a normal distribution (`np.random.normal` with argument `size=(n_samples,n_atoms, n_dimensions)`) with zero mean and a standard deviation of 0.1. Add the equilibrium structure to the generated array to obtain an array of shape `(n_samples,n_atoms,n_dimensions)` containing the positions of all atoms.Visualize the atomic positions of the noised data in a scatter plot (`plt.scatter`).

1.2 Featurization

We want to study two different types of featurization: Absolute positions and pairwise distances.

1.2.1 Absolute atomic positions

We don't need to do much preprocessing before using the absolute positions as features apart from reshaping: Remember that the MLP can only work with rank-1 data, so we need to reshape (`np.reshape`) the last two dimensions of the array containing the noised positions

(create a new array, don't modify the original array with the positions, we will need it later). You should end up with an array of shape `(n_samples,n_atoms*n_dimensions)`.

1.2.2 Pairwise distances

To extract the pairwise distances between all atoms, you can use the function `pdist` imported from `scipy.spatial.distance`. Apply it to the unflattened array of atomic positions to obtain an array of shape `(n_samples,n_pairs)`. Note that `pdist` already returns the condensed distances, i.e., only one distance for each pair of atoms such that each pair is only counted once.

1.3 Labeling

As a final step before we can train a model, we need to assign labels to the generated configurations, i.e., compute the energy of each sample. The easiest way to do this is to first evaluate the pair potential in Eq. (1) for all pairwise distances and then summing all contributions up according to Eq. (2). Plot a histogram of the generated energies (`plt.hist`). After evaluating the energy, create test and training `Datasets` for both feature sets (4 `Datasets` in total) and create two training `DataLoaders`.

1.4 Training

Train two different MLPs on the two different features. Choose a suitable architecture (you don't have to report convergence tests, but make sure both models achieve at least MSE $< 10^{-3}$ on the test set) and train both models for a few hundred epochs. For both models, plot the loss on the training set vs the number of epochs and report the MSE on training and test set and make a plot y_{pred} vs y_{true}.

2 Data augmentation (5 pt)

The energy of the molecule should be invariant under rotations and permutations of equivalent atoms. In this exercise, we want to see which of these symmetries are already incorporated into the features and of which we need to take care ourselves.

2.1 Rotations

Generate a new set of noised atomic positions by rotating each sample by $\pi/2$. Visualize the new set of atomic positions in a scatter plot. For the rotated positions, repeat the steps from 1.2 and 1.3, i.e., recompute the features and create new `Datasets`. Evaluate the models trained in 1.4 on the new datasets (do **not** retrain your model before the evaluation). What do you observe? Do you have an explanation?

2.2 Permutations

Generate a new set of noised atomic positions by permuting the atoms in each sample. You can use any permutation you like, e.g., $[0, 1, 2, 3] \rightarrow [3, 0, 1, 2]$. Visualize the new set of atomic positions in a scatter plot. For the rotated positions, repeat the steps from 1.2 and 1.3, i.e., recompute the features and create new `Datasets`. Evaluate the models trained in 1.4 on the new datasets (do **not** retrain your model before the evaluation). What do you observe? Do you have an explanation?

2.3 Augmentation

Generate two new sets of noised atomic positions by combining a) the original positions with the rotated positions and b) the original positions with the permuted positions. To do so, you can use `np.vstack`. You should end up with two arrays of shape `(2*n_samples,n_atoms*n_dimensions)` and `(2*n_samples,n_pairs)`, respectively. Retrain your models on these two datasets and evaluate them on the test sets of the permuted and rotated positions. What do you observe? Do you have an explanation?

Do you think that data augmentation is a useful technique to treat invariances with respect to rotations and permutations? In your answer, consider that "real-world "simulations may contain several thousands of atoms. How large would the dataset become if you wanted to include all permutations of 1024 atoms?