Machine Learning for Molecular Physics WiSe 24/25

Exercise 10 Machine-learned potentials III December 17, 2024

Due date: January 07, 2025
Please submit a pdf with answers (add figures if appropriate) as well as the notebook used to generate them.

1 SchNet (10 pt)

Over the next two assignements, we will implement one of the most widely used graph neural networks (GNNs) in the molecular sciences: SchNet, a continuous-filter convolutional neural network. Our implementation will follow the original publication very closely, so have a look if you're interested in learning more about the details. Some of the building blocks we will build ourselves, some we will take from the `PyTorch Geometric` library (which you need to install into your environment via `python3 -m pip install torch_geometric torch_cluster`).

As in the previous two assignments, our final goal is to use SchNet as an energy function in our MD simulation. For this sheet, however, we do not need any simulation data and it is enough to create some "fake" data with the correct dimensions which we will create in the notebook.

1.1 Basic building blocks

We first recall all the relevant building blocks and updates that we will need to build our model. You only need to implement layers which are not marked as "already implemented".

1.1.1 Graph (already implemented)

A central step of any GNN is to build the graph representation of the input data for a given cutoff radius. Here, we use the `RadiusInteractionGraph` class to do this. Have a look at the notebook to understand how it works. Select one configuration of the fake MD trajectory and use the function `visualize_graph` to plot a 2D projection of the resulting graph with different cutoff values.

1.1.2 Embedding

Besides the atomic positions from which the graph is built, SchNet only uses the atomic numbers as input parameters. Through the layers of SchNet, the atom i at layer l is described by a feature vector $\mathbf{x}_i^l \in \mathbb{R}^F$ with the feature dimension F. For a system with different types of atoms, every atom type Z_i has the same representation. As we only have one atom type, the initial embedding at the first layer of all atoms is identical:

$$\mathbf{x}_i^0 = \mathbf{a}(Z). \tag{1}$$

$\mathbf{a}$ is a trainable parameter, which we can realize, *e.g.*, by generating $\mathbf{a}$ as output of an MLP or simply a single linear layer that takes as input only the one-dimensional atomic number Z. As we only have one type of atom, we can choose any number - for simplicity, we set $Z = 1.0$.

1.1.3 Radial basis expansion

SchNet uses an expansion in radial basis functions. Every pairwise interatomic distance $d_{ji} = ||\mathbf{r}_j - \mathbf{r}_i||$ is expanded in a basis of Gaussians

$$e_k(\mathbf{r}_j - \mathbf{r}_i) = \exp(-\gamma \cdot (||\mathbf{r}_j - \mathbf{r}_i|| - \mu_k)^2) \tag{2}$$

with centers μ_k chosen on a uniform grid between zero and the distance cutoff. Define a class `RadialExpansion` that is initialized with γ and an array containing all μ_k and a has a method `expand` implementing Eq. (2). For $\gamma = 4.0$ and using ten different basis functions (*i.e.*, $n_k = 10$ and `mu=torch.linspace(0,r_cut,10)`), visualize (`plt.plot`) the resulting expansion for three test distances $d_{ji} = 0.5, 1.0, 1.5$. What is the idea behind using this type of expansion?

1.1.4 Interaction block (already implemented)

The interaction blocks of SchNet apply updates to the individual representations $\mathbf{x}$ according to

$$\mathbf{x}_i^{l+1} = \sum_{j \in \mathcal{N}(i)} \mathbf{x}_j^l \circ W^l(\mathbf{r}_j - \mathbf{r}_j), \tag{3}$$

where $\mathbf{x}_i^l$ and W^l are the representation of atom i and the filter-generating network at layer l, respectively. As this process is somewhat complex in terms of implementation, we use a version that is already implemented in `PyTorch Geometric`.

1.1.5 Atomwise layers and readout (already implemented)

As discussed in the lecture, the output of a graph network is a graph. Therefore, we need to add a readout function that transforms the graph representation to the output of interest. In our case, we are interested in the total potential energy of the system, which is modeled as the sum of all atomic contributions $E = \sum_i^N E_i$. Consequently, SchNet uses a few atom-wise updates followed by a summation readout.

1.2 Bringing it all together

Now we can bring all the relevant layers together and build SchNet. First, we need to initialize the relevant layers. Besides the already implemented layers, we need:

1. An embedding layer (`torch.linear`) which takes as input the atomic number and outputs the embedding vector.
2. A layer to obtain the interaction graph (use `RadiusInteractionGraph`).
3. A layer for the expansions in radial basis functions.

Now, we need to implement the `forward` pass of the model. As input, the model takes a batch of atomic positions (shape `(n_atoms * batch_size,3)`), atomic numbers (shape `(batch_size,1)`), and the batch dimensions assigning each particle to the correct sample (shape `(n_atoms * batch_size)`
The `forward` pass should consist of:

1. Generating the embedding vectors of all atoms (see 1.1.2). Since this is operation is identitical for each atom, you can use `torch.vmap` to map over the first dimension. The output should be of shape `(n_atoms * batch_size, dim_embedding)`.
2. Computing the interaction graph (*i.e.*, get indices and distances between atoms within the cutoff).
3. Expanding the distances into radial basis functions. You can again use `torch.vmap` to map over the batch dimensions.
   The resulting array should have dimension `(n_pairs * batch_size, num_gaussians)`.
4. Residual update of the embedding vector using the interaction blocks (already implemented).
5. Readout (already implemented).