ECFP vs. NGF

Extended-Connectivity Fingerprints (ECFPs) are a type of molecular fingerprint used to represent chemical structures for machine learning and cheminformatics tasks. They are circular fingerprints that capture the local chemical environment around each atom up to a specified radius.

The algorithm works by iteratively encoding atom neighborhoods into unique integer identifiers, which are then hashed into a fixed-length binary vector. The most common variant, **ECFP4**, uses a radius of 2, capturing atom environments within two bond lengths.

Background:

- ECFPs provide an excellent representation of molecules as binary fingerprints, as shown in various benchmarks performing molecular property prediction with a MLP or random forest projection head.
- NGF claims to produce embeddings of molecules using a differentiable network that are comparable to those computed by circular fingerprints.
- In the limit of large weights, the embedding vector of a molecule computed by NGF should be similar to that of circular fingerprints. Yet Figure 3 clearly shows significant deviations between pairwise distances.
- In this project we are limiting our focus solely on the embeddings and leave the performance on downstream tasks to another project.
- We will only look at the embeddings of the NGF model, without any training.
- We will be using the “continuous generalization of the Tanimoto” score to evaluate pairwise distances as outlined in the NGF paper.

Task 1:

- Implement **Algorithm 1** from scratch as described in the pseudocode in Figure 2. Compare the binary fingerprints created using **Algorithm 1** to those from RDKit on the ESOL Dataset. The concatenate operation seems to omit the Bond-Order as well as the iteration number information as described in the paper Extended-Connectivity Fingerprints. This should in theory have an impact on the fingerprint generated and was not taken into account in the NGF **Algorithm 2.**

Task 2:

- Replicate the results of Figure 3 comparing the pairwise circular fingerprint distances to the NGF distances on the ESOL Dataset. The paper claims to have used “large random weights”. Describe these weights, what order of magnitude do they have? Can you increase them to arbitrary sizes? What if you decrease their size or even set them to zero? Take the correlation coefficient “r” as a measure of comparison and create some visual plots.

Task 3: Improve the correlation coefficient “r”

- Set the embedding dimension of **NGF** to the fpSize of the circular fingerprint (1024, 2048). Do you already see any differences?
- We will take a closer look at the individual functions. Keep in mind, we are not actually training the network here, so be sure to take this into account when you are performing your evaluations.

Task 3.1: Sum Function:

- Should replace the concatenate operation of the ECFP.
- Introduces summation as a permutation invariant function.
- Include the Bond-Order information (potentially by multiplying r_i with the Bond-Order)

Task 3.2: Smooth Function:

- Takes the aggregated node embeddings and applies *tanh* piecewise to each element, after multiplication with weights *H*. The authors claim that “in the limit of large input weights, tanh nonlinearities approach step functions, which when concatenated form a simple hash function”.
- Is this actually required? What happens if we remove this?
- Replace the *tanh* function with another differentiable hash function.
- Come up with better ideas here

Task 3.3: Sparsify Function:

- The softmax function effectively magnifies the difference between the largest element and all others, leading to a spare vector i with many small values, and a single large value.
- Replace softmax with Gumbel Softmax. This distribution allows the temperature parameter to be set to 0, creating a one-hot output. Need to look into this more.

Task 3.4: Add to Fingerprint Function:

- This updates the embedding vector at every step with the sparsified vector computed in the previous step. Compared to the circular fingerprint variant, individual elements are added at each step, as opposed to setting a binary value.
- Is there a way to avoid continuous accumulation of individual elements during each layer?

Algorithm 1 Circular fingerprints

Input: molecule, radius R, fingerprint length S
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow [\mathbf{r}_a, \mathbf{r}_1, \dots, \mathbf{r}_N]$ ▷ concatenate
    $\mathbf{r}_a \leftarrow \text{hash}(\mathbf{v})$ ▷ hash function
    $i \leftarrow \text{mod}(r_a, S)$ ▷ convert to index
    $\mathbf{f}_i \leftarrow 1$ ▷ Write 1 at index
Return: binary vector $\mathbf{f}$

Algorithm 2 Neural graph fingerprints

Input: molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$
Initialize: fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
for each atom a in molecule
  $\mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
for $L = 1$ to R ▷ for each layer
  for each atom a in molecule
    $\mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
    $\mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^{N} \mathbf{r}_i$ ▷ sum
    $\mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ ▷ smooth function
    $\mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ ▷ sparsify
    $\mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ ▷ add to fingerprint
Return: real-valued vector $\mathbf{f}$

Figure 2: Pseudocode of circular fingerprints (*left*) and neural graph fingerprints (*right*). Differences are highlighted in blue. Every non-differentiable operation is replaced with a differentiable analog.

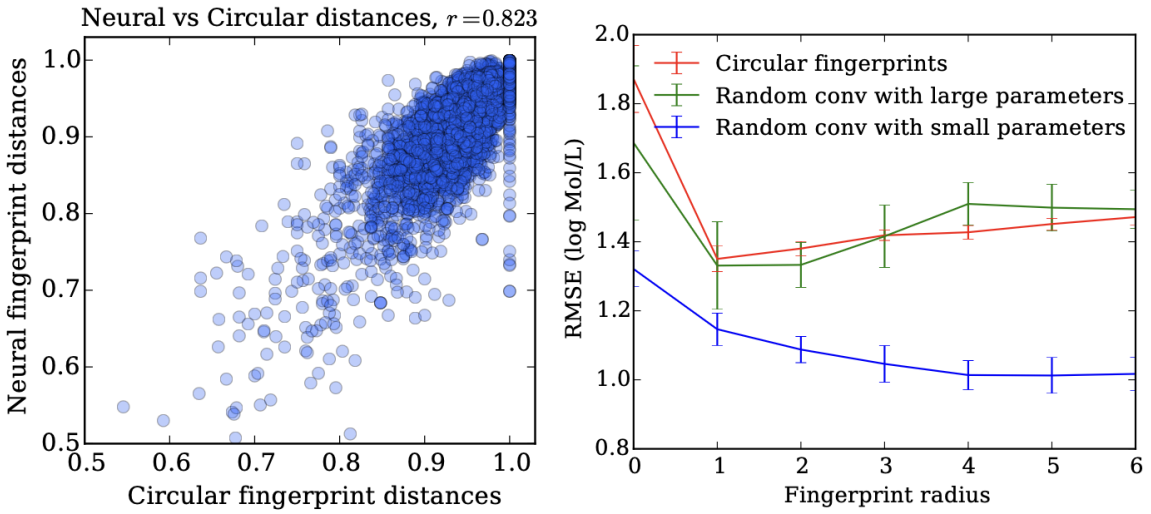

Figure 3: *Left:* Comparison of pairwise distances between molecules, measured using circular fingerprints and neural graph fingerprints with large random weights. *Right*: Predictive performance of circular fingerprints (red), neural graph fingerprints with fixed large random weights (green) and neural graph fingerprints with fixed small random weights (blue). The performance of neural graph fingerprints with large random weights closely matches the performance of circular fingerprints.

Optional

1. Examine the behaviour of similarity in downstream tasks.
2. Verify that **Algorithm 2** is correctly implemented in Pytorch Geometric. Are there any discrepancies? If yes, which ones and what are their impact?
3. Instead of looking at the binary fingerprint, we could also look at the count fingerprints.