Homework 4 : Invariant and Equivariant Neural Networks

Deep Learning and Groups, ECE

1 Character inner products and Burnside's lemma

We derived all permutation equivariant layers from a vector space $\mathbb{R}^n$ to itself in two seemingly unrelated ways:

1. Finding the S_n-orbits in X^2 where $X = \{1, 2, \dots, n\}$ and using them to construct equivariant linear maps.
2. Finding the irreducible subrepresentations of the permutation representation acting on $\mathbb{R}^n$ and then using Schur's Lemma to find all S_n-homomorphisms between the irreducible spaces of $\mathbb{R}^n$.

Problem. Use the analysis in Tutorial 12 to prove that these two methodologies both find the number of G-equivariant layers from $\mathbb{R}^n$ to $\mathbb{R}^n$ for any finite subgroup G and natural n.

Hints: 1. Use the restriction of the permutation representation to a subgroup $G \leq S_n$ to define the action of G on vectors. 2. Characterize both approaches using Burnside's Lemma.

2 Netflix Rating Matrices and Group Equivariance

Consider Netflix rating matrices holding ratings of m movies by n people. Answer the following questions:

1. What is the natural symmetry group in this case?
2. What are the linear equivariant layers? Provide the general form, including both the bias and linear part.
3. Draw the parameter sharing scheme for the linear equivariant layer.
4. What are the linear invariant layers? Provide their general form.
5. Generalize to 3-tensors with another dimension. What is the parameter-sharing scheme in this case? Draw it.

3 Implementing set networks

Consider set data with S_n symmetry, where each element of the set has multiple feature dimensions.

We will implement various types of invariant networks or equivariant layers in PyTorch. For each of the following network architectures, provide a PyTorch implementation and a test function to check equivariance/invariance. Ensure that your implementations can handle multiple feature dimensions for each set element.

For each method, consider first whether it is invariant or equivariant to devise the test function. For $(a)-(d)$, the test function does not learn optimal network parameters, it simply checks invariance or equivariance. In e, you need to come up with a training procedure so that the network optimizes its parameters to approximate the variance.

(a) **Canonization-based network:** Apply a canonization function and then your favorite model.

(b) **Symmetrization network:** Apply symmetrization to your favorite model by averaging over all permutations.

(c) **Sampled symmetrization:** Apply symmetrization to your favorite model but only with a subset of the group elements.

(d) **Linear equivariant layers:** Implement a network using linear equivariant layers concatenated with pointwise nonlinearities.

(e) **Data augmentation:** Implement a standard architecture that is trained with data augmentation.

Implementation Guidelines

1. For each network type, create a PyTorch `nn.Module` class that implements the described architecture.

2. Ensure that your implementations can handle input tensors of shape (n, d), where n is the number of set elements and d is the number of features for each element.

3. Implement a test function for each network that checks for invariance or equivariance as appropriate. The test function should:

   - Generate random input data
   - Apply random permutations to the input
   - Compare the output of the original and permuted inputs
   - Return a boolean indicating whether the network is invariant/equivariant

4 Benchmarking Permutation Invariant Networks on 3D Shape Classification

In this exercise, you will compare different approaches to creating permutation-invariant neural networks for set data. You will evaluate these networks on a distribution classification task and analyze their performance in terms of accuracy and computational efficiency.

Learning task. The learning task is a classification task on the ModelNet40 dataset, which contains 3D point cloud data of 40 object categories with 12,311 CAD-generated meshes (9,843 for training and 2,468 for testing).

Dataset: Download from here:
`https://www.dropbox.com/scl/fi/qcddtiathbfcp6v3wimoi/modelnet40_normal_resampled.zip?rlkey=3t9jz6lr954wxmfj4xp6u9m2a&dl=0`

The point clouds are encoded as text files and are stored according to their class. Each text file contains a list of points where the first three numbers in each row represent the 3D coordinates and the next three numbers represent the normal direction. Unless stated otherwise, use the first 256 points from each point cloud (in other words, each point cloud should be represented as a 256×3 matrix).
Simple code for point cloud loading should be something like:

```
import numpy as np

# Load point cloud (first 256 points)
data = np.loadtxt('airplane/airplane_0001.txt', delimiter=',')[:256]
coords = data[:, :3]  # x, y, z coordinates
```

Please visualize the data for several classes using a scatter plot and make sure it makes sense.

Architectures. Use the following types of permutation invariant networks:

(a) Canonization-based network

(b) Symmetrization network (would not work for 256 points, please select a lower number of points for which you could run the experiments below in a reasonable time, this might be even less than 10 points)

(c) Sampled symmetrization network (Sample random 5% of possible permutations, you can increase the number of points you used for full symmetrization to all points or to a larger subset of points.)

(d) Linear equivariant layers with pointwise nonlinearities

(e) Data augmentation with a standard (non-equivariant) architecture

For the canonization-based and symmetrization networks, implement using both standard MLPs. (Bonus: implement with transformer and positional encoding)

Experiment Setup

- Use CrossEntropyLoss for training.
- stop according to a predefined validation set and report the result on the test set.
- Train each network with varying numbers of training examples. Specifically, repeat training with a training set consisting of 5, 10, and 50 samples per class. If no considerable difference, modify these numbers accordingly.
- Set a maximum runtime of 0.5 hours for each experiment. If an experiment exceeds this time, mark it as "Out of Time" (OOT) and report the final text accuracy you got.
- bonus: What happens if we use the normals as inputs as well?

You may use free GPUs available online, such as those provided by Google Colab: `https://colab.research.google.com/`.

Analysis and Reporting

1. Create a single plot showing accuracy vs. number of training examples for each architecture.
2. Prepare a table summarizing the runtime of each architecture.
3. Analyze and discuss the trade-offs between the different approaches in terms of:
    - Accuracy
    - Computational efficiency
    - Scalability with set size and dimensionality
    - Ease of implementation
4. Discuss any challenges you encountered during implementation and experimentation.
5. Based on your results, recommend which approach(es) might be most suitable for different scenarios (e.g., small vs. large sets, limited vs. abundant computational resources).
6. Provide insights into why linear equivariant layers might be particularly well-suited for this task compared to other architectures.

7. (Bonus) There exists another group invariance in this problem. First, consider the pairwise distance matrix. What group invariant is it? Try to devise rotation-invariant node features based on this invariant. (Suggestion: First, 'centralize' the point cloud such that the mean of the points is at the origin, and that the point with the maximal norm is on the unit sphere.)