Exercise Sheet 3

Due date: Tuesday, December 13th, 23:59

- The content of your master branch at the time of the deadline is your submission.
- You should use the following Python packages in this exercise:
  - PyTorch
  - PyTorch Scatter
  - NetworkX
  - Numpy (Scipy might be useful too)
  - Scikit-Learn (might be useful)
  - argparse

Exercise 1 (Custom Dataset) — 6 points

Implement a custom dataset class for graph learning. Your class should inherit from the `torch.utils.data.Dataset` class and take a list of NetworkX graphs as input in the constructor. You have to overwrite the `__getitem__` and `__len__` methods accordingly. In particular, `__getitem__(i)` should return the sparse representation of the i-th graph in the dataset. The representation should consist of the following matrices (stored as PyTorch Tensors):

- The *directed* edge list $\text{idx}_E \in \{0, \dots, |V| - 1\}^{2 \times 2|E|}$
- The node feature matrix $X_V \in \mathbb{R}^{|V| \times d}$
- The edge feature matrix $X_E \in \mathbb{R}^{2|E| \times d'}$
- The graph label $y \in \mathbb{R}^c$

Exercise 2 (Custom Collation) — 6 points

Implement a collation function for your sparse graph representation as a Python method. The single input of the method is a list of sparse representations of the graphs in the batch. The output should be the sparse representation of the disjoint union graph, which merges all graphs into one. Recall that this graph carries an additional attribute `batch_idx`, which stores the index of the original graph for each vertex in the merged graph. Once implemented, you can pass your method as the `collate_fn` parameter to the data loader. The loader will then use the function to merge training samples generated by your dataset from Task 1.

Exercise 3 (GNN Layer) — 7 points

Implement a GNN layer based on scatter operations as a PyTorch Module. Your implementation should support SUM, MEAN and MAX aggregation. The type of aggregation should be passed to the layer in the constructor. You can choose $\mathbf{M}$ and $\mathbf{U}$ to be functions that first concatenate their respective inputs along the feature dimension and then apply an MLP:

$$Y^{(\ell)} = \mathbf{M}\big(\big[H^{(\ell-1)}\big[\mathrm{idx}_E[0]\big], X_E\big]\big)$$

$$H^{(\ell)} = \mathbf{U}\big(\big[H^{(\ell-1)}, Z^{(\ell)}\big]\big)$$

The width and depth of $\mathbf{M}$ and $\mathbf{U}$ should also be passed to the layer in the constructor.

Exercise 4 (Sparse Sum Pooling) — 3 points

Implement a PyTorch Module that performs sum pooling based on scatter operations.

Exercise 5 (Virtual Node) — 3 points

Implement a PyTorch Module that computes a Virtual Node as discussed in the lecture. Your implementation should also be based on scatter operations.

Reminder: A virtual node can be placed after each GNN Layer (except for the last one) to allow for global information exchange. It computes the following update for the embedding of each node v:

$$h^{\ell}(G) = \mathbf{V}^{\ell}\Big(\sum_{v \in V} h^{\ell}(v)\Big)$$
$$\tilde{h}^{\ell}(v) = h^{\ell}(v) + h^{\ell}(G)$$

Here, $\mathbf{V}^{\ell}$ is a trainable MLP.

Exercise 6 (Evaluation) — 10 points

Evaluate your implementation on the ZINC dataset. The ZINC dataset is a regression task. The graph labels are real numbers. Therefore, the final dense layer of your model should be linear and have output dimension 1.

Vary the hyperparameters of the network (depth, hidden dimension, type of aggregation, with or without virtual nodes,...) to empirically determine a good configuration. In your readme, provide a comparison of the main configurations you evaluated.

ZINC is already split in train, validation and test data. Use the train split for training and the validation split for model selection. Finally, use the test split to evaluate the models test performance. You do not have to perform cross validation in this exercise.

For each network configuration, report the *mean absolute error* (MAE) on the 3 splits. Your best model should have an MAE of at most 0.2 on the test split (-5 points if it does not).

Hint

- For training, you can use any standard regression loss. We recommend using the $\ell 1$-Loss (absolute error).
- Do not forget to use the edge labels of this dataset. They are stored as integers under "edge_label" as edge attributes in the NetworkX graphs. Convert these into one-hot vectors before training.

Exercise 7 (Code Quality, Comments and Presentation) 15 points

Clean your code and add useful comments. Your repository must contain a `README.md` file which provides the following information:

- A brief description of the structure of your repository
- How to run every executable script. For each script, all command line options must be fully specified
- The MAE achieved on the three splits of ZINC for each tested architecture.

Prepare a short presentation (∼5 min) for your group meeting (held in the week after the submission deadline). It should briefly provide the following information:

- What you implemented and how the work was split
- The results you obtained
- A brief interpretation and discussion of the results