Basile Terver
basile.terver@polytechnique.edu

Lab session # 7
ALTEGRAD 2023

12/05/23

Tasks 1, 2 and 3

In this lab we are working on multisets. The padding with zeros is to ensure to ensure the LSTM and Deepset models can process the multisets.
We want our Deepset model to be invariant (or equivariant) to permutation of the encoding of the elements of the set. Indeed, a set (in this case a multiset) does not have ordering. We want to hard-encode this fact in the architecture of the network. We want invariance, that is $\forall \pi \in S_n, f(\pi X) = f(X)$, where f represents the whole model architecture.
We one-hot encode our multisets with the embedding layer of both models. Thus we learn the identity with our MLP.

Question 1

The main motivation for LSTM [2] is precisely to process data in a sequential way, therefore LSTMs are highly not permutation-invariant by nature. LSTMs are built to process sequences of data without treating each point in the sequence independently, but rather, retaining useful context information from previous data in the sequence to help processing new data points. LSTMs are made to process data under the form of sequences such as text, speech and time-series.

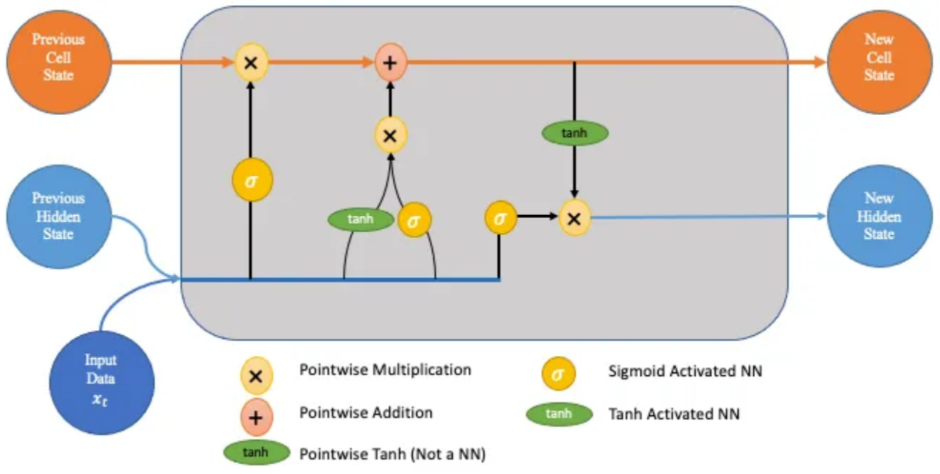

Figure 1: Diagram of how an LSTM cell works at some point t in time from [2]. Three notable elements are 1)The current long-term memory of the network, the *cell state*, 2)The output at the previous point in time, the *previous hidden state*, 3)The input data at the time t.

The pytorch documentation of the LSTM module clearly exhibits the sequential nature of LSTMs and of the data they are supposed to process. Therefore, LSTMs are not recommended at all for sets.

Task 5

One thing to remark here is that a permutation of the whole training set at each epoch of training. This allows to randomly select a new division into batches of the training set on which to do training for a given epoch.

Task 7

The performance of Deepsets depends on the randomly generated training and test sets, on the initial weights of the model and on training. When we re-create the dataset and re-train the model, in some cases, we achieve perfect accuracy for all cardinalities, like in Figure 2, or have an accuracy equal to one until a certain

cardinality (like 50) and dropping to zero afterwards (I would like to have plotted this but I have overwritten the weights of this training).
Meanwhile, the accuracy curve of LSTM always has this shape, with an accuracy peak at some low cardinality (cardinal 10 in 2) and zero performance elsewhere.
This confirms what we said in Question 1 about the fact that LSTM is not adapted at all to processing sets (but rather sequential data) while Deepsets is much more relevant because it is permutation-invariant.

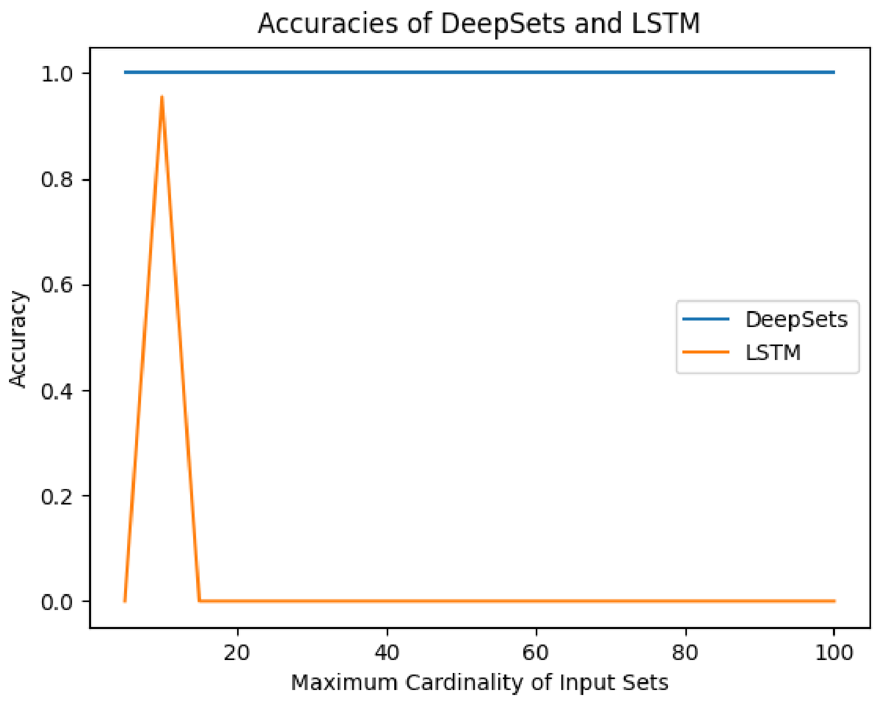

Figure 2: Accuracies of the LSTM and Deepsets model on the test dataset with respect to maximum cardinality of the input sets.

Question 2

In lab 6, the GNNs we were using for graph prediction tasks needed to aggregate the information of all nodes, they needed to aggregate it spatially using a readout function. The readout function was permutation invariant, it was either the sum or the mean function.
In Lab 6, the GNN model consisted of two message passing layers (having either mean or sum aggregation function) followed by either a sum or a mean readout function and then, by one or two fully-connected layer.

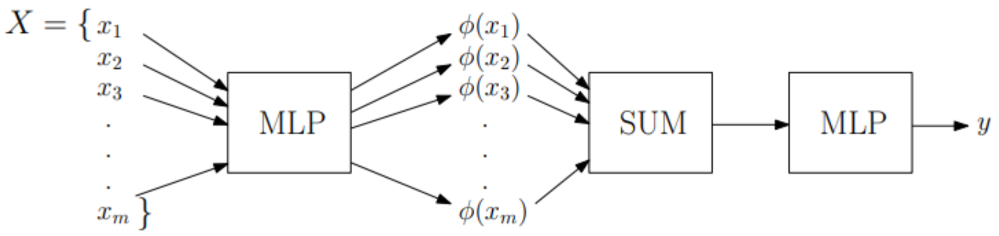

Figure 3: The Deepsets model, taken from the handout of the Lab.

Thus, the last layers, that is the readout layer followed by an MLP, is common to Lab 6 and Lab 7. The difference between the two models lies in the previous layers, which are message passing layers in the GNN case
Let us take the example of a graph without edges being processed by the GNN model. Then its adjacency matrix $A = 0$. Thus, taking notations of Lab 6 Section 3 (X is the input), we see that each message passing layer $Z = ReLU(XW_1 + AXW_0) = ReLU(XW_1)$, is equivalent to a fully-connected layer. Therefore, in this case, the whole GNN model is equivalent to the Deepsets model.

Question 3

Homophilic and heterophilic stochastic block model

In a stochastic block model with 2 blocks ($r = 2$), here are two edge probability matrices from which homophilic and heterophilic graphs can respectively be sampled.

$$P_{homo} = \begin{pmatrix} 0.9 & 0.05 \\ 0.05 & 0.9 \end{pmatrix} \tag{1}$$

$$P_{hetero} = \begin{pmatrix} 0.1 & 0.7 \\ 0.7 & 0.1 \end{pmatrix} \tag{2}$$

Example: Expected number of edges between nodes in different blocks of a heterophilic graph with 20 nodes and 4 blocks

Each of the 4 blocks B_1, B_2, B_3, B_4 has 5 nodes. The existence of an edge between each pair of nodes is considered as a random variable independent of the others.

Consider the first block B_1 and a node $v \in B_1$. It has 15 nodes that are in another block, that is not in B_1. For each node w of these 15 nodes, the probability of having an edge $\{v, w\}$ is 0.05. Thus, v has an expected number of edges with nodes in different blocks equal to $0.05 \times 15 = 0.75$. Let us sum over B_1 to get an expected number of nodes between B_1 and other blocks equal to $0.75 * 5 = 3.75$.

Now, we consider B_2. We can only consider its potential edges with B_3 and B_4. Each $v \in B_2$ has an expected number of edges with nodes in different blocks equal to $0.05 \times 10 = 0.5$. Let us sum over B_2 to get an expected number of nodes between B_2 and other blocks (except B_1) equal to $0.5 * 5 = 2.5$.

Now, we consider B_3. We can only consider its potential edges with B_4. Each $v \in B_2$ has an expected number of edges with nodes in different blocks equal to $0.05 \times 5 = 0.25$. Let us sum over B_2 to get an expected number of nodes between B_3 and B_4 equal to $0.25 * 5 = 1.25$.

Summing over B_1, B_2, B_3, we get that the **expected number of edges between nodes of different blocks equals** $3.75 + 2.5 + 1.25 = \mathbf{7.5}$.

Task 8

One should note, in the notations of the handout of this Lab, and the notation MP for a message passing layer from Lab 6, we have $MLP_1(\bar{A}X) = MLP(MP(\bar{A}, X)) = MLP(ReLU(\bar{A}XW_0))$, where $W_0 \in \mathbb{R}^{d \times h_1}$, with d being the input's node feature dimension.

Question 4

As explained in [1], the binary cross-entropy loss is only relevant for classification tasks, but reconstructing a weighted graph is rather a regression task.
For a weighted graph G with an adjacency matrix A and a reconstructed adjacency matrix $\hat{A}$, the MSE loss can be defined as follows:

$$MSE(A, \hat{A}) = \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} (A_{ij} - \hat{A}_{ij})^2 \tag{3}$$

Using MSE as the reconstruction loss is suitable for weighted graphs because it accounts for the continuous nature of edge weights and penalizes larger deviations between predicted and actual weights. This helps the model to better capture the nuances of weighted relationships in the graph.

Task 11

We note that d in the Task 11 of the handout means the latent (low) dimension of the space in which we learn a representation of the training graphs via learning $\mu, \sigma^2 \in \mathbb{R}^d$. One should note that we learn a normal multivariate distribution with independent components $N(\mu, diag(\sigma^2))$.
Let us describe how the graphs generated from the variational autoencoder are plotted. From `utils.py` and `main.py`, we see that:

- First, the values of the matrix generated by the decoder, denoted A^{gen}, is rounded to be an acceptable (non-weighted) adjacency matrix: the values of A^{gen} that are above 0.5 are set to 1 and the values below 0.5 are put to 0. This yields an acceptable adjacency matrix, but with potential zero degree nodes. We remove those nodes and obtain an acceptable adjacency matrix $\tilde{A}^{gen}$.
- We compute the best partition of the nodes into clusters using the Louvain algorithm, in the `find_communities_and_plot()` function.
- We reorder the adjacency matrix so that the nodes belonging to the same cluster have contiguous indices.
- We plot the adjacency matrices with black corresponding to 0 and white corresponding to 1.

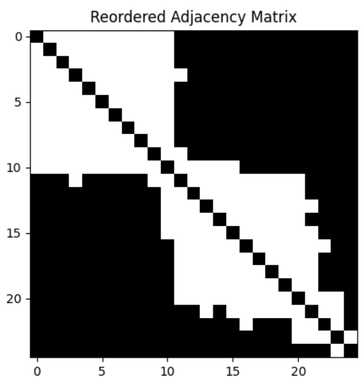

Figure 4: Drawing of the first generated adjacency matrix $\tilde{A}_1^{gen}$.

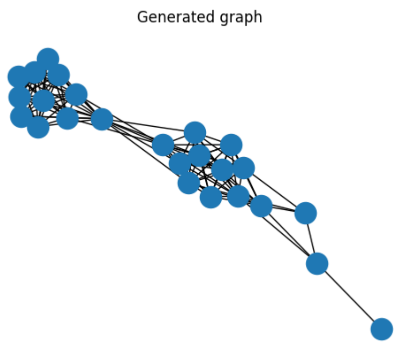

Figure 5: Drawing of the associated first generated graph.

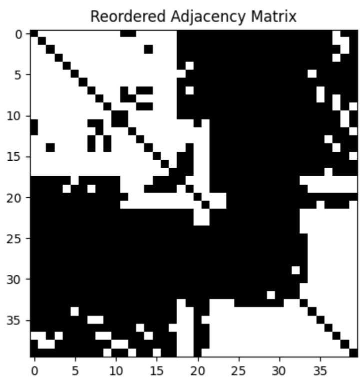

Figure 6: Drawing of the second generated adjacency matrix $\tilde{A}_2^{gen}$.

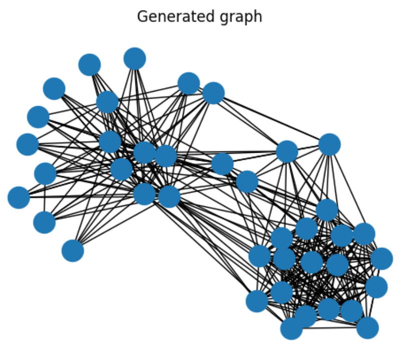

Figure 7: Drawing of the associated second generated graph.

References

[1] Richmond Alake. Loss functions in machine learning explained. `https://www.datacamp.com/tutorial/loss-function-in-machine-learning`, 2023. Accessed on December 1st 2023.

[2] Rian Dolphin. Lstm networks — a detailed explanation. `https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9`, 2020. Accessed on December 1st 2023.

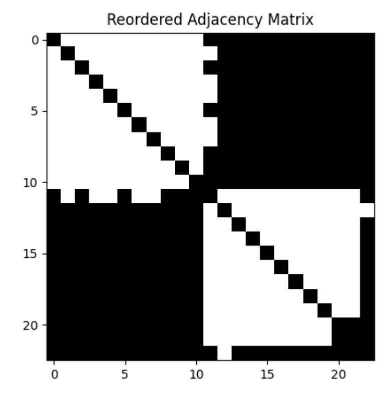

Figure 8: Drawing of the third generated adjacency matrix $\tilde{A}_3^{gen}$.

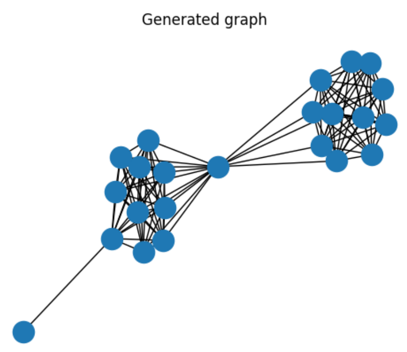

Figure 9: Drawing of the associated third generated graph.

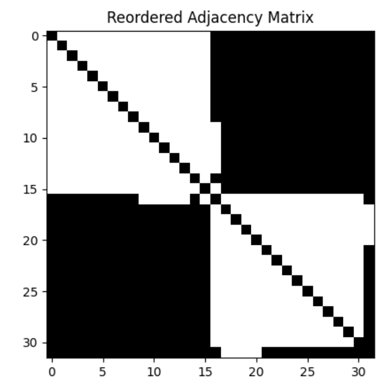

Figure 10: Drawing of the fourth generated adjacency matrix $\tilde{A}_4^{gen}$.

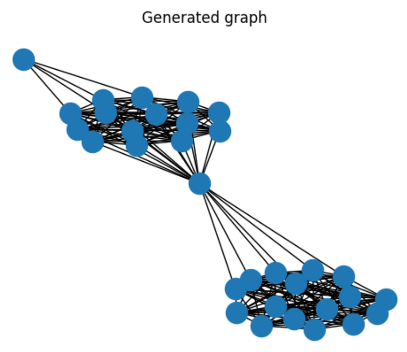

Figure 11: Drawing of the associated fourth generated graph.

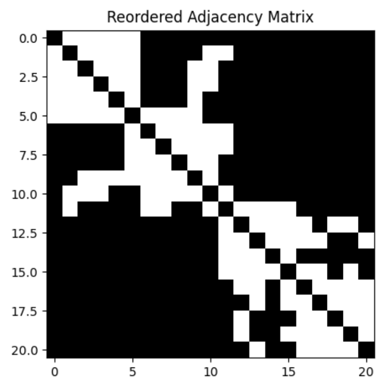

Figure 12: Drawing of the fifth generated adjacency matrix $\tilde{A}_5^{gen}$.

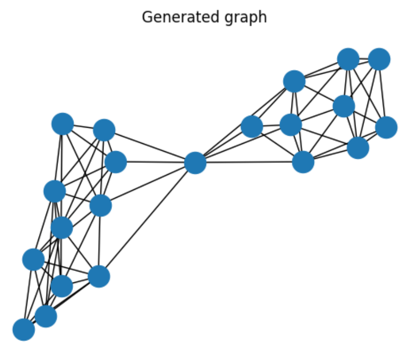

Figure 13: Drawing of the associated fifth generated graph.