

# EESM5900V

## Assignment 2

Please submit a written report.

### 1 Attention and Transformer [40 points]

- Consider we have  $n$  pairs of keys and values as well as  $m$  queries. Each key, value, or query is a vector of dimension  $d$ . What is the time complexity to perform the attention operation? Please use the notation  $O(\cdot)$  in terms of  $n, m, d$  for your result (assume matrix multiplication is implemented in a simple way).
- In a transformer, there is self-attention in its encoder and cross-attention in its decoder. Explain the difference between self-attention and cross-attention. Can we replace self-attention in the encoder with cross-attention?

### 2 Graph Neural Networks [60 points]

Graph Neural Networks (GNNs) are a class of neural network architectures used for deep learning on graph-structured data. Broadly, GNNs aim to generate high-quality embeddings of nodes by iteratively aggregating feature information from local graph neighborhoods using neural networks; embeddings can then be used for recommendations, classification, link prediction or other downstream tasks. Two important types of GNNs are GCNs (graph convolutional networks) and GraphSAGE (graph sampling and aggregation).

Let  $G = (V, E)$  denote a graph with node feature vectors  $X_u$  for  $u \in V$ . To generate the embedding for a node  $u$ , we use the neighborhood of the node as the computation graph. At every layer  $l$ , for each pair of nodes  $u \in V$  and its neighbor  $v \in V$ , we compute a message function via neural networks, and apply a convolutional operation that aggregates the messages from the node's local graph neighborhood (Figure 1), and updates the node's representation at next layer. By repeating this process through  $K$  GNN layers, we capture feature and structural information from the local  $K$ -hop neighborhood. For each of the message computation, aggregation and update functions, the learnable parameters are shared across all nodes in the same layer.

We initialize the feature vector of each node  $X_u$  based on the data we have available. If we already have outside information about the nodes, we can embed that as a feature vector. Otherwise we can use constant feature (vector of 1) or the degree of the node as the feature vector.

These are the key steps in each layer of a GNN:

- **Message computation:** We use a neural network to learn a message function between nodes. For each pair of nodes  $u$  and its neighbor  $v$ , the neural network message function can be expressed as  $M(h_u^k, h_v^k, e_{u,v})$ . In GCN and GraphSAGE, this can simply be  $\sigma(Wh_v + b)$ , where  $W$  and  $b$  are the weights and bias of a neural network linear layer. Here  $h_u^k$  refers to the hidden representation of node  $u$  at layer  $k$ , and  $e_{u,v}$  denotes available information about the edge  $(u, v)$ , like the edge weight or other features. For GCN and GraphSAGE, the neighbors of  $u$  are simply defined as nodes that are connected to  $u$ . However, many other variants of GNNs have different definitions of neighborhood.
- **Aggregation:** At each layer, we apply a function to aggregate information from all of the neighbors of each node. The aggregation function is usually permutation invariant, to reflect the fact that nodes' neighbors have no canonical ordering. In a GCN, the aggregation is done by a weighted sum, where the weight for aggregating from  $v$  to  $u$  corresponds to the  $(u, v)$  entry of the normalized adjacency matrix  $D^{-1/2}AD^{-1/2}$ .
- **Update:** We update the representation of a node based on the aggregated representation of the neighborhood. For example, in GCNs, a multi-layer perceptron (MLP) is used; GraphSAGE combines a skip layer with the MLP.
- **Pooling:** The representation of an entire graph can be obtained by adding a pooling layer at the end. The simplest pooling methods are just taking the mean, max or sum of all of the individual node representations<sup>1</sup>. This is usually done for the purposes of graph classification<sup>2</sup>.

We can formulate the Message computation, Aggregation and Update steps for a GCN as a layerwise propagation rule given by:

$$h^{k+1} = \sigma(D^{-1/2}AD^{-1/2}h^k W^k), \quad (1)$$

where  $h^k$  represents the matrix of activations in the  $k$ -th layer,  $D^{-1/2}AD^{-1/2}$  is the normalized adjacency of graph  $G$ ,  $W^k$  is a layer-specific learnable matrix, and  $\sigma$  is a non-linearity function. Dropout and other forms of regularization can also be used.

We provide the pseudo-code for GraphSAGE embedding generation below.

---

<sup>1</sup>More complex pooling such as DiffPool and Graph CNN (Defferrard et al.) can be done at intermediate layers.

<sup>2</sup>However, note that some architectures such as graph networks (Battaglia et al.) use pooling mechanism for node-level tasks as well.

---

**Algorithm 1:** Pseudo-code for forward propagation in GraphSAGE

---

**Input** : Graph  $G(V, E)$ ; input features  $\{x_v, \forall v \in V\}$ ; depth  $K$ ; non-linearity  $\sigma$ ; weight matrices  $\{W^k, \forall k \in [1, K]\}$ ; neighborhood function  $\mathcal{N} : v \rightarrow 2^V$ ; aggregator functions  $\{\text{AGGREGATE}_k, \forall k \in [1, K]\}$

**Output:** Vector representations  $z_v$  for all  $v \in V$

$h_v^0 \leftarrow x_v, \forall v \in V$  ;

**for**  $k = 1 \dots K$  **do**

**for**  $v \in V$  **do**

$h_{\mathcal{N}(v)}^k \leftarrow \text{AGGREGATE}_k(\{h_u^{k-1}, \forall u \in \mathcal{N}(v)\})$  // aggregation

$h_v^k \leftarrow \sigma(W^k \cdot \text{CONCAT}(h_v^{k-1}, h_{\mathcal{N}(v)}^k))$  // MLP with skip connection

$h_v^k \leftarrow h_v^k / \|h_v^k\|_2, \forall v \in V$  // update step

$z_v \leftarrow h_v^K, \forall v \in V$

---

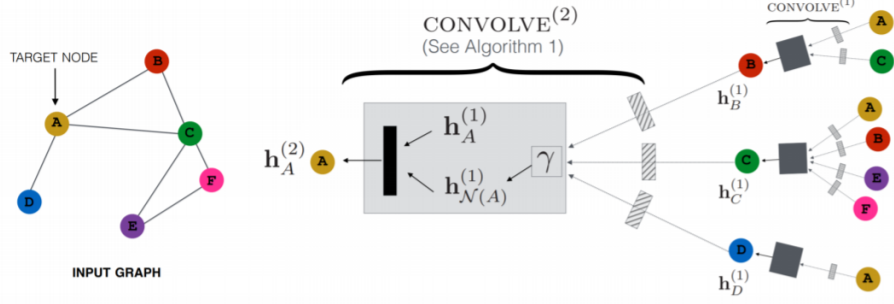


Figure 1: GNN architecture.

In this question, we investigate the effect of the number of message passing layers on the expressive power of Graph Convolutional Networks. In neural networks, expressiveness refers to the set of functions (usually the loss function for classification or regression tasks) a neural network is able to compute, which depends on the structural properties of a neural network architecture.

## 2.1 Effect of Depth on Expressiveness [20 points]

1. Consider the following 2 graphs, where all nodes have 1-dimensional feature. We use a simplified version of GNN, with no nonlinearity and linear layers, and sum aggregation. We run the GNN to compute node embeddings for the 2 red nodes respectively. Note that the 2 red nodes have different 4-hop neighborhood structure. How many layers of message passing are needed so that these 2 nodes can be distinguished (i.e., have different GNN embeddings)?



reached by BFS has embedding 1, and nodes not reached by BFS has embedding 0. Write the update rule at every step of BFS execution.

2. Describe a message function and an aggregation function for the GNN such that it learns the task perfectly.

## What to submit

Question 2.1:

- Reasoning of why a proposed number of message passing step is enough to distinguish the neighborhoods.
- An example that should be classified to True.
- Reasoning of why GNNs with fewer than 5 layers do not have enough expressive power.

Question 2.2:

- Correct transition matrix, in terms of  $D$  and  $A$ .
- Correct transition matrix for the case with skip layer.

Question 2.3:

- Update rule: the function that determines whether a node is reachable from source at step  $t$ , given its previous reachability and its neighbors' previous reachability.
- aggregation method (anything containing sum+clamp; max; min) all acceptable.