Chahine NEJMA
chahine.nejma@student-cs.fr

Question 1

1. **Output of a single head.**
For head $k \in \{1, \dots, K\}$, we first compute unnormalized attention scores

$$e_{ij}^{(t+1,k)} = \text{LeakyReLU}\Big(\big(\mathbf{a}^{(k)}\big)^\top \big[\mathbf{W}^{(k)}\mathbf{z}_i^{(t)} \,\|\, \mathbf{W}^{(k)}\mathbf{z}_j^{(t)}\big]\Big),$$

and normalize them over the neighbors of i:

$$\alpha_{ij}^{(t+1,k)} = \frac{\exp\big(e_{ij}^{(t+1,k)}\big)}{\sum\limits_{p\in\mathcal{N}(i)} \exp\big(e_{ip}^{(t+1,k)}\big)}.$$

The output of head k for node i is then

$$\mathbf{z}_i^{(t+1,k)} = \sigma\left(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}^{(t+1,k)}\, \mathbf{W}^{(k)}\mathbf{z}_j^{(t)}\right),$$

where $\sigma(\cdot)$ is a nonlinearity (e.g. ELU or ReLU).

2. **Concatenation over K heads.**
The final representation of node i is obtained by concatenating the outputs of all heads:

$$\mathbf{z}_i^{(t+1)} = \big\|_{k=1}^{K} \mathbf{z}_i^{(t+1,k)}.$$

Since each head produces a vector in $\mathbb{R}^{F'_{\text{out}}}$, the total dimensionality of $\mathbf{z}_i^{(t+1)}$ is

$$\dim\big(\mathbf{z}_i^{(t+1)}\big) = K\, F'_{\text{out}}.$$

3. **Total number of learnable parameters.**
For a single head k:

   - The weight matrix $\mathbf{W}^{(k)} \in \mathbb{R}^{F'_{\text{out}}\times F_{\text{in}}}$ contributes $F'_{\text{out}}F_{\text{in}}$ parameters.
   - The attention vector $\mathbf{a}^{(k)} \in \mathbb{R}^{2F'_{\text{out}}}$ contributes $2F'_{\text{out}}$ parameters.

   Hence, one head has

$$F'_{\text{out}}F_{\text{in}} + 2F'_{\text{out}} = F'_{\text{out}}(F_{\text{in}} + 2)$$

   parameters. With K independent heads, the total number of parameters (ignoring biases for simplicity) is

$$K\, F'_{\text{out}}(F_{\text{in}} + 2).$$

Question 2

We assume that all nodes share the same feature vector:

$$\mathbf{x}_i = \mathbf{c} \in \mathbb{R}^d \quad \text{for all } v_i \in V.$$

1. **Unnormalized score e_{ij} and coefficient α_{ij}.**
In a single-head GAT layer, we have

$$e_{ij} = \text{LeakyReLU}\Big(\mathbf{a}^\top \big[\mathbf{W}\mathbf{x}_i \,\|\, \mathbf{W}\mathbf{x}_j\big]\Big), \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k\in\mathcal{N}(i)} \exp(e_{ik})}.$$

Under the assumption $\mathbf{x}_i = \mathbf{c}$ for every i, we have

$$\mathbf{W}\mathbf{x}_i = \mathbf{W}\mathbf{c}, \qquad \mathbf{W}\mathbf{x}_j = \mathbf{W}\mathbf{c},$$

which implies

$$e_{ij} = \text{LeakyReLU}\Big(\mathbf{a}^\top \left[\mathbf{W}\mathbf{c} \,\|\, \mathbf{W}\mathbf{c}\right]\Big) =: \tilde{e},$$

where $\tilde{e}$ is a constant (independent of i and j). Therefore, for any neighbor $j \in \mathcal{N}(i)$:

$$\alpha_{ij} = \frac{\exp(\tilde{e})}{\sum_{k\in\mathcal{N}(i)} \exp(\tilde{e})} = \frac{\exp(\tilde{e})}{|\mathcal{N}(i)| \exp(\tilde{e})} = \frac{1}{|\mathcal{N}(i)|}.$$

2. **Effect on the expressiveness of the GAT layer.**

   All neighbors of a node receive the same attention weight:

   $$\alpha_{ij} = \frac{1}{|\mathcal{N}(i)|}.$$

   The update rule becomes

   $$\mathbf{z}_i^{(t+1)} = \sigma\left(\frac{1}{|\mathcal{N}(i)|} \sum_{j\in\mathcal{N}(i)} \mathbf{W}\mathbf{z}_j^{(t)}\right),$$

   i.e. a simple average of transformed neighbor features. The mechanism no longer distinguishes between neighbors, so it no longer behaves as a meaningful *attention* mechanism.

   In this limit, the layer effectively reduces to a standard neighborhood averaging GNN, similar in spirit to a mean-aggregator GraphSAGE layer or a simplified GCN layer.

3. **Why can it still beat random guessing on the Karate graph?**

   Even though the node features carry no useful information and attention becomes uniform, the model still operates on a non-trivial graph structure: the connectivity pattern in the Karate club network is highly correlated with the community labels.

Question 3

To obtain a *conditional* VGAE, we introduce an additional conditioning variable $\mathbf{c}$ (for instance a graph-level label, side information, or some global attributes). Instead of modeling

$$q_\phi(\mathbf{Z} \mid A, X) \quad \text{and} \quad p_\theta(A \mid \mathbf{Z}),$$

we make both the encoder and decoder explicitly depend on $\mathbf{c}$:

$$q_\phi(\mathbf{Z} \mid A, X, \mathbf{c}), \qquad p_\theta(A \mid \mathbf{Z}, \mathbf{c}),$$

and possibly define a conditional prior $p(\mathbf{Z} \mid \mathbf{c})$.
From an architectural viewpoint, this can be implemented in several ways:

- **Encoder:** concatenate $\mathbf{c}$ to node features or graph embeddings before the encoder GNN, or inject it via conditioning mechanisms (e.g. FiLM layers). Symbolically,

  $$X' = [X \,\|\, \mathbf{c}], \quad q_\phi(\mathbf{Z} \mid A, X, \mathbf{c}) = q_\phi(\mathbf{Z} \mid A, X').$$

- **Decoder:** condition the adjacency reconstruction on both $\mathbf{Z}$ and $\mathbf{c}$, for example by concatenating $\mathbf{c}$ to graph-level latent codes before the decoder MLP, or using $\mathbf{c}$ to modulate decoder layers:

  $$A \sim p_\theta(A \mid \mathbf{Z}, \mathbf{c}).$$

- **Prior:** instead of a standard isotropic prior, use

  $$p(\mathbf{Z} \mid \mathbf{c}) = \mathcal{N}\big(\mu(\mathbf{c}), \Sigma(\mathbf{c})\big),$$

  where $\mu(\mathbf{c})$ and $\Sigma(\mathbf{c})$ are learned functions of the condition.

Question 4

In a VAE/VGAE, the encoder outputs the parameters (μ, σ) of a Gaussian distribution and we want to sample

$$\mathbf{z} \sim \mathcal{N}(\mu, \sigma^2 I)$$

while still being able to backpropagate through μ and σ.

Reparameterization trick

We rewrite the sampling step as

$$\mathbf{z} = \mu + \sigma \odot \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I),$$

so that $\boldsymbol{\epsilon}$ is independent noise and $\mathbf{z}$ is a differentiable function of (μ, σ) for a fixed draw of $\boldsymbol{\epsilon}$. Standard backpropagation can then be used to obtain gradients with respect to μ and σ.

Without reparameterization

If we instead sampled

$$\mathbf{z} \sim \mathcal{N}(\mu, \sigma^2 I)$$

directly via a black-box sampler, $\mathbf{z}$ would not be differentiable with respect to μ and σ. No useful gradients could flow from the loss to the encoder, and one would need high-variance estimators. The reparameterization trick avoids this by making the sampling step itself differentiable, allowing stable end-to-end training by backpropagation.