Wardakhan Kevin
kevin.wardakhan@icloud.com

Lab session # 7
ALTEGRAD 2023

12/06/25

1 Question 1

1.1

For a given attention head $k \in \{1, \dots, K\}$, the output of node v_i at layer $t+1$ is given by

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

where the attention coefficients of head k are

$$\alpha_{ij}^{(t+1,k)} = \frac{\exp\Big(\text{LeakyReLU}\left(\mathbf{a}^{(k)\top} [\mathbf{W}^{(k)} \mathbf{z}_i^{(t)} \,\|\, \mathbf{W}^{(k)} \mathbf{z}_j^{(t)}]\right)\Big)}{\sum_{\ell \in \mathcal{N}_i} \exp\Big(\text{LeakyReLU}\left(\mathbf{a}^{(k)\top} [\mathbf{W}^{(k)} \mathbf{z}_i^{(t)} \,\|\, \mathbf{W}^{(k)} \mathbf{z}_\ell^{(t)}]\right)\Big)}$$

Here $\mathbf{W}^{(k)} \in \mathbb{R}^{F'_{\text{out}} \times F_{\text{in}}}$ is the weight matrix of head k, $\mathbf{a}^{(k)} \in \mathbb{R}^{2F'_{\text{out}}}$ is its attention vector, and $\mathcal{N}_i$ is the set of neighbors of node v_i.

1.2

Concatenating all the output vectors $\mathbf{z}_i^{(t+1,k)} \in \mathbb{R}^{F'_{\text{out}}}$ for $k = 1, \dots, K$ gives

$$\mathbf{z}_i^{(t+1)} = \left[\mathbf{z}_i^{(t+1,1)} \,\|\, \dots \,\|\, \mathbf{z}_i^{(t+1,K)}\right] \in \mathbb{R}^{K F'_{\text{out}}}$$

1.3

The learnable parameters of this multi-head GAT layer are, for each head k, the weight matrix $\mathbf{W}^{(k)}$ and the attention vector $\mathbf{a}^{(k)}$. We have

$$\mathbf{W}^{(k)} \in \mathbb{R}^{F'_{\text{out}} \times F_{\text{in}}} \quad \Rightarrow \quad F'_{\text{out}} F_{\text{in}} \text{ parameters,}$$

and

$$\mathbf{a}^{(k)} \in \mathbb{R}^{2F'_{\text{out}}} \quad \Rightarrow \quad 2F'_{\text{out}} \text{ parameters.}$$

Thus, one head contains

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

trainable parameters.
Since there are K heads, the total number of learnable parameters in the multi-head layer is

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

2 Question 2

2.1

All nodes have the same feature vector $x_i = c \in \mathbb{R}^d$. For any pair of nodes i, j we then have

$$Wx_i = Wx_j = Wc$$

so the concatenated vector used in the attention mechanism is

$$[Wx_i \,\|\, Wx_j] = [Wc \,\|\, Wc]$$

The unnormalized attention score is therefore

$$e_{ij} = \text{LeakyReLU}\left(a^\top [Wx_i \,\|\, Wx_j]\right) = \text{LeakyReLU}\left(a^\top [Wc \,\|\, Wc]\right) =: \gamma$$

which is a constant that does not depend on i or j.
For a fixed node i, all neighbors $j \in \mathcal{N}_i$ share the same score $e_{ij} = \gamma$. The normalized attention coefficient is

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} = \frac{\exp(\gamma)}{|\mathcal{N}_i| \exp(\gamma)} = \frac{1}{|\mathcal{N}_i|}$$

for every neighbor $j \in \mathcal{N}_i$.

2.2

2.2 From Question 2.1, when all node features are identical the attention coefficients become uniform:

$$\alpha_{ij} = \frac{1}{|\mathcal{N}_i|}, \qquad j \in \mathcal{N}_i.$$

Plugging this into the GAT update rule gives

$$\mathbf{z}_i^{(t+1)} = \sum_{j \in \mathcal{N}_i} \alpha_{ij}\, \mathbf{W}\mathbf{z}_j^{(t)} = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \mathbf{W}\mathbf{z}_j^{(t)}.$$

Thus, each neighbor contributes with the same fixed weight $1/|\mathcal{N}_i|$, independently of its features: the layer can no longer assign different importance to different neighbors, so it no longer behaves as a true attention mechanism.
This update rule is a standard neighborhood-averaging GNN propagation rule.

2.3

In the degenerate case all node embeddings are identical, so the final classifier outputs the same probability $p \in (0, 1)$ for every node. With binary labels $y_i \in \{0, 1\}$ and binary cross-entropy, the loss is

$$L(p) = -\frac{1}{n} \sum_{i=1}^{n} \big(y_i \log p + (1 - y_i) \log(1 - p) \big).$$

Let $q = \frac{1}{n} \sum_{i=1}^{n} y_i$ be the empirical fraction of class 1. Then

$$L(p) = -\big(q \log p + (1 - q) \log(1 - p) \big),$$

and

$$\frac{\partial L}{\partial p} = -\Big(\frac{q}{p} - \frac{1 - q}{1 - p} \Big) = 0 \quad \Rightarrow \quad p = q,$$

which is the unique minimiser.
Thus the trained model outputs $p = q$ for every node and predicts class 1 iff $q > 1/2$. It therefore always predicts the majority class, achieving accuracy $\max(q, 1 - q)$, which is strictly larger than 0.5 on the Karate network where the classes are imbalanced. Hence the model can still perform better than random guessing despite the loss of feature information.

3 Question 3

To introduce conditions into a Variational Graph Autoencoder, we add a condition vector $\mathbf{c}$ and make the model conditional:

$$q_\phi(\mathbf{z} \mid G, \mathbf{c}), \qquad p_\theta(G \mid \mathbf{z}, \mathbf{c})$$

In practice, we can choose one of the following:
1. *Conditional encoder:* feed $\mathbf{c}$ to the GNN encoder together with $(\mathbf{X}, \mathbf{A})$, for example by concatenating it to a graph-level embedding:

$$\boldsymbol{\mu}, \boldsymbol{\sigma} = \mathrm{GNN}_\phi(\mathbf{X}, \mathbf{A}, \mathbf{c}), \quad \mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \hat{\mathbf{A}} = \mathrm{MLP}_\theta(\mathbf{z})$$

2. *Conditional decoder:* keep the encoder as usual and concatenate $\mathbf{c}$ to the latent code:

$$\boldsymbol{\mu}, \boldsymbol{\sigma} = \mathrm{GNN}_\phi(\mathbf{X}, \mathbf{A}), \quad \mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \hat{\mathbf{A}} = \mathrm{MLP}_\theta([\mathbf{z} \,\|\, \mathbf{c}])$$

In both cases the model is trained with a conditional ELBO,

$$\mathbb{E}_{q_\phi(\mathbf{z} \mid G, \mathbf{c})}[\log p_\theta(G \mid \mathbf{z}, \mathbf{c})] - \mathrm{KL}\big(q_\phi(\mathbf{z} \mid G, \mathbf{c}) \,\|\, p(\mathbf{z})\big)$$

3.1 Question 4

In the ELBO we have an expectation w.r.t. $q_\phi(z \mid x)$:

$$\mathbb{E}_{z \sim q_\phi(z \mid x)}[\log p_\theta(x \mid z)]$$

If we sample *directly* as $z \sim \mathcal{N}(\mu_\phi(x), \sigma_\phi^2(x)I)$, the sampling step is non–differentiable, so $\partial z/\partial\phi$ is treated as zero by backprop and the encoder parameters ϕ cannot be updated using standard gradients.
With the reparameterization trick we write

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I)$$

so the randomness is isolated in ϵ, independent of ϕ, and z is now a differentiable function of $\mu_\phi(x)$ and $\sigma_\phi(x)$. This gives

$$\nabla_\phi \text{ELBO}(\phi, \theta) = \mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\left[\nabla_\phi \log p_\theta(x \mid z) + \nabla_\phi \log q_\phi(z \mid x) - \nabla_\phi \log p(z)\right]$$

with $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon$, which can be estimated by Monte Carlo and optimized with standard backpropagation.