Homework 4

Question 1

Graph Attention Network (GAT): In a GAT, each node in the graph has its own features. These features are updated by aggregating information from neighbouring nodes, weighted by attention scores. The attention mechanism allows each node to attend to different neighbours with different weights, based on the similarity of their features. Fully Connected Graph: In a fully connected graph, every node is connected to every other node. This means that in a GAT with a fully connected graph, each node can potentially attend to all other nodes in the graph. Standard Transformer Architecture: A standard transformer architecture consists of encoder and decoder layers. Each layer in the transformer consists of multi-head self-attention mechanisms followed by feed-forward neural networks. In the self-attention mechanism, each word in a sequence attends to all other words in the sequence, weighted by their similarities. Now, let's see how a fully connected GAT can be equivalent to a standard transformer architecture:

- Node-to-Node Attention in GAT: In a fully connected GAT, each node attends to all other nodes in the graph, similar to how each word attends to all other words in a sequence in the self-attention mechanism of a transformer.
- Aggregation of Information: In both GAT and transformer, after computing attention scores, the information from neighbouring nodes/words is aggregated using a weighted sum. In GAT, this is achieved by computing attention coefficients and then taking a weighted sum of neighbour node features. In transformer, this is done through the self-attention mechanism where attention scores are used to compute weighted sums of all word embeddings.
- Feed-forward Neural Networks: After the aggregation of information, both GAT and transformer architectures use feed-forward neural networks to process the aggregated information and produce the final output for each node/word.

Therefore, when the graph in a GAT is fully connected, the mechanism of node-to-node attention and information aggregation resembles the self-attention mechanism in a standard transformer architecture. Thus, a fully connected GAT can be considered equivalent to a standard transformer architecture.

Question 2

a. Gathering node-embedding vectors:

Given the node-embedding vectors $\{h_n\}$, we gather them into an $N \times D$ matrix H such that row n is given by h_n^T.

b. Expressing neighborhood-aggregated vectors in matrix form:

The neighborhood-aggregated vector z_n for each node n is defined as the sum of the embeddings of its neighbors:

$$z_n = \sum_{m \in N(n)} h_m$$

Here, h_m represents the embedding of node m.

We can write this operation in matrix form by using the adjacency matrix A. Let A be the adjacency matrix where $A_{ij} = 1$ if there is an edge between nodes i and j, and $A_{ij} = 0$ otherwise. Then, the neighborhood aggregation operation can be expressed as:

$$Z = AH$$

Here, Z is the $N \times D$ matrix in which row n is given by z_n^T, and A is the adjacency matrix.

c. Writing the argument to the nonlinear activation function in matrix form:

The argument to the nonlinear activation function in equation (13.16) is:

$$AH \cdot W_{\text{neigh}} + H \cdot W_{\text{self}} + \frac{1}{D}\mathbf{1}_D^T \cdot b$$

Here, W_{neigh} and W_{self} are weight matrices, b is the bias vector, and $\mathbf{1}_D$ is the D-dimensional column vector in which all elements are 1.

We can write this in matrix form as:

$$AH \cdot W_{\text{neigh}} + H \cdot W_{\text{self}} + \frac{1}{D}\mathbf{1}_D^T \cdot b = (AH)W_{\text{neigh}} + HW_{\text{self}} + \frac{1}{D}\mathbf{1}_D^T \cdot b$$

$$= ZW_{\text{neigh}} + HW_{\text{self}} + \frac{1}{D}\mathbf{1}_D^T \cdot b$$

So, the argument to the nonlinear activation function in matrix form is $AH \cdot W_{\text{neigh}} + H \cdot W_{\text{self}} + \frac{1}{D}\mathbf{1}_D^T \cdot b$, where $Z = AH$.

Question 3

The two properties

- Equivariance Property (13.19): The equivariance property states that for any permutation matrix P, if we apply the permutation to the input node features and then apply the graph convolution operation, it's equivalent to applying the graph convolution first and then applying the same permutation to the output features.
- Permutation Property (13.4): The permutation property states that for any permutation matrix P, if we permute the node features, the output of the graph convolutional layer remains invariant up to permutation.

Now, let's consider a complete deep graph convolutional network (GCN) defined by equation (13.18). This equation represents the operation of a graph convolutional layer followed by a non-linear activation function, and it is applied recursively over multiple layers. We want to show that this network is equivariant. To do this, let's consider the input node features represented by matrix X(0) and the output features after l layers represented by X(l).

Now, according to the equivariance property (13.19), if we permute the input features X(0) with a permutation matrix P, and then apply the graph convolution operation defined by equation (13.18), it should be equivalent to first applying the graph convolution to the original input features and then permuting the output features X(l) with the same permutation matrix P.

Since the graph convolution operation defined by equation (13.18) is permutation invariant (as per permutation property (13.4)), applying it to the permuted input features will produce the same output features regardless of the permutation. Therefore, the complete deep graph convolutional network defined by equation (13.18) is equivariant, as its output remains invariant up to permutation under permutation of the input features.

Question 4

As the given matrix A is symmetric, its eigen vectors are orthogonal. This implies we just have to transform the eigen values matrix into a singluar value matrix. The singular values must be non-zero and must be in decreasing order. Let's transform the given equation using this objective:

$$
\begin{aligned}
A &= \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{pmatrix} \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{pmatrix} \\
&= \begin{pmatrix} u_{11} & u_{12} & -u_{13} \\ u_{21} & u_{22} & -u_{23} \\ u_{31} & u_{32} & -u_{33} \end{pmatrix} \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{pmatrix}
\end{aligned}
$$

$$= \begin{pmatrix} u_{11} & -u_{13} & u_{12} \\ u_{21} & -u_{23} & u_{22} \\ u_{31} & -u_{33} & u_{32} \end{pmatrix} \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{pmatrix}$$

This is the SVD for the given matrix A

Question 5

Since H is a symmetric matrix, its eigen decomposition would look like $H = U\Sigma U^T$. Substituting this in RHS of the equation would result in,

$$A^T H A = A^T U \Sigma U^T A = (U^T A)^T \Sigma (U^T A)$$

Assume $B = U^T A$, then

$$B^T B = (U^T A)^T (U^T A) = A^T U U^T A$$

Because U is an orthogonal matrix $U^T U = UU^T = I_n$, we also know that $A^T A = I_k$, then

$$B^T B = A^T (UU^T) A = A^T A = I$$

This means, B is a matrix with orthonormal columns. The trace can be written as,

$$\text{trace}(A^T H A) = \text{trace}(B^T \Sigma B) = \text{trace}(\Sigma B B^T) = \sum_{i=1}^{n} \lambda_i ||b_i||^2$$

where λ_i is the ith eigenvalue of H and b_i is the ith row of B. b_i can also be written as the ith column of B^T, i.e., $b_i = (u_i^T A)^T = A^T u_i$. Then the above trace becomes:

$$\text{trace}(A^T H A) = \sum_{i=1}^{n} \lambda_i ||A^T u_i||^2$$

We know that $A^T A = I_k$ and $[a_1, a_2, \ldots, a_k]$ are orthonormal vectors. Another way to look at this is A is the first k columns orthogonal matrix of size $n \times n$.That is, consider $A^c = [a_{k+1}, a_{k+2}, \ldots, a_k]$ to be another set of orthonormal vectors such that, $[A, A^c]$ is an orthogonal matrix. Then,

$$\begin{aligned} ||A^T u_i||^2 &\leq ||A^T u_i||^2 + ||(A^c)^T u_i||^2 \\ &\leq ||[A, A^c]^T u_i||^2 \\ &\leq u_i^T [A, A_c][A, A_c]^T u_i \\ &\leq u_i^T u_i \quad \text{(since, } [A, A_c] \text{ is orthogonal)} \\ &\leq 1 \end{aligned}$$

Given that we already know $||[A, A^c]^T u_i|| = 1$, the above expression becomes equal when $||A^T u_i|| = 1$. This means, $(A^c)^T u_i = 0$. Therefore, u_i should be in the orthogonal complement space of the linear space formed by $[a_{k+1}, a_{k+2}, \ldots, a_k]$ vectors.

In other words, u_i should be in the column space of A. Which means, u_i could be written as a linear combination of columns of A. Therefore, u_i could be written as

$$u_i = Aq_i$$

where q_i is a unit vector $\in \mathbb{R}$ (4). However, we did not yet establish for which i this is true.
Summation over all rows i will be,

$$\begin{aligned}
\sum_{i=1}^{n} ||A^T u_i||^2 &= \sum_{i=1}^{n} (u_i)^T A A^T u_i \\
&= \text{trace}((U^T A)(A^T U)) \\
&= \text{trace}((A^T U)(U^T A)) \\
&= \text{trace}(A^T U U^T A) \\
&= \text{trace}(A^T A) \\
&= \text{trace}(I_k) \\
&= k \quad \text{(since } I_k \text{ is a } k \times k \text{ identity matrix)}
\end{aligned}$$

Now,

$$\begin{aligned}
\text{trace}(A^T H A) &= \sum_{i=1}^{n} \lambda_i ||A^T u_i||^2 \\
&\leq \sum_{i=1}^{k} \lambda_i ||A^T u_i||^2 + \sum_{i=k+1}^{n} \lambda_i ||A^T u_i||^2 \\
&\leq \sum_{i=1}^{k} \lambda_i ||A^T u_i||^2 + \lambda_{k+1} \sum_{i=k+1}^{n} ||A^T u_i||^2 \\
leq \sum_{i=1}^{k} \lambda_i ||A^T u_i||^2 &+ \lambda_{k+1} \left(\sum_{i=1}^{n} ||A^T u_i||^2 - \sum_{i=1}^{k} ||A^T u_i||^2 \right) \\
&\leq \sum_{i=1}^{k} \lambda_i ||A^T u_i||^2 + \lambda_{k+1} (k - \sum_{i=1}^{k} ||A^T u_i||^2) \\
&\leq \sum_{i=1}^{k} (\lambda_i - \lambda_{k+1}) ||A^T u_i||^2 + k\lambda_{k+1} \\
&\leq \sum_{i=1}^{k} (\lambda_i - \lambda_{k+1}) + k\lambda_{k+1} \leq \sum_{i=1}^{k} \lambda_i
\end{aligned}$$

Therefore we can say $||A^T u_i|| = 1$ for $1 \leq i \leq k$ and $||A^T u_i|| = 0$ for $i \geq k$. Finally, from the upper bound derived above, we can claim that $\max(\text{trace}(A^T H A)) = \sum_{i=1}^{k} \lambda_i$. Hence, part 1 is proved.

We also have $u_i = Aq_i$ for $1 \leq i \leq k$. This can be also written in compact form as $U = AQ^T$ where $U = [u_1, u_2, \ldots, u_k]$ and $Q^T = [q_1, q_2, \ldots, q_k]$. Note that transpose is used for notation convenience.

Now, if Q^T is an orthogonal matrix then,

$$U = AQ^T$$

$$UQ = AQ^T Q$$

$$UQ = A$$

Therefore, the optimal A^* is given by $A^* = [u_1, u_2, \ldots, u_k]Q$ where Q is an orthogonal matrix. Hence, part 2 is also proved.

Question 6 : PCA vs Autoencoder

(b) The results were as follows:

Number of principal components (p)	Reconstruction Error
32	134.91234
64	87.07439
128	46.16377
256	$7.37e^{-13}$

(d) In this variant of the Autoencoder, we use shared weights between encoder and decoder i.e, decoder weights is the transpose of the encoder weights. The results were as follows:

hidden dim (d)	Reconstruction Error
32	129.85867
64	86.33544
128	46.45157
256	2.64873

(e) In summary, these are the reconstruction erros when we set p=d and compare both PCA and AE:

p,d	PCA error	AE error
32	134.91234	129.85867
64	87.07439	86.33544
128	46.16377	46.45157
256	$7.37e^{-13}$	2.64873

From the above results, we can see that when p is low(32, 64), AE has marginally lower reconstruction error than PCA. When p is high, PCA error is marginally low. So, if we ignore the minor differences in the errors, overall in conclusion, when the weights are shared between encoder and decoder, AE is equivalent to PCA.

(f)

p = d	$\|G - W\|_F$
32	8.07431
64	11.42209
128	15.88385
256	22.60100

We can deduce that G and W are not same. This is expected because PCA aims to achieve minimal reconstruction error through singular value decomposition which is not unique for a given matrix, whereas AE aims to achieve minimal reconstruction error through gradient descent. Therefore, G and W may not be the same. However, we know that solutions of PCA are related by an orthogonal transformation. And one of them must be W. Let us assume W = GR_{orth} where R_{orth} is some orthogonal transformation. First let's verify that both are orthonormal matrices. For that we verify $\|GTG - WTW\| \approx 0$

p = d	$\|G^TG - W^tW\|_F$
32	0.00722
64	0.00994
128	0.01714
256	1.23904

We can see, all are close to zero. Which means, both G and W are indeed orthonormal matrices. Now, let's verify the transformation W = GR_{orth}. If this were true, then

$$\begin{aligned} G^TW &= G^TGR_{\text{orth}} \\ &= R_{\text{orth}} \end{aligned}$$

Now as R_{orth} is an orthogonal matrix, $R_{\text{orth}}^TR_{\text{orth}} = R_{\text{orth}}R_{\text{orth}}^T = I$ must be true. Now, let's verify that:

$p = d$	$\|R_{\text{orth}}R_{\text{orth}}^T - I\|_F$	$\|R_{\text{orth}}^T R_{\text{orth}} - I\|_F$
32	0.15125	0.15125
64	0.15861	0.15861
128	1.07329	1.07329
256	1.23904	1.23904

This proves that R_{orth} is indeed an orthogonal matrix, which means our original assumption, i.e., $W = GR_{\text{orth}}$, must also be true. Therefore, the transformation that takes from G to W is: $W = GR_{\text{orth}}$, where $R_{\text{orth}} = G^T W$.

(g)

d	error with shared weights	error without shared weights
32	129.85867	130.51717
64	86.33544	86.94984
128	46.45157	46.90643
256	2.64873	6.09348

From the above results, we can observe that both then networks perform almost the same. AE with shared weights may be just a special case of AE without shared weights. Perhaps the gradient descent pushed the decoder weights to be close to the transpose of encoder weights in order to achieve minimal reconstruction error.

(h) After setting d=64, learning rate =0.001, max epochs = 300 and tuning rest of the hyper parameters, following were the results:

Network	Batch Size	Reconstruction Error
$[256, 128, 64, 128, 256]w.ReLU$	32	77.73651
$[256, 128, 64, 128, 256]w.ReLU$	64	74.12938
$[256, 128, 64, 128, 256]w.ReLU$	128	72.45694
$[256, 128, 64, 128, 256]w.tanh$	32	77.42769
$[256, 128, 64, 128, 256]w.tanh$	64	72.22675
$[256, 128, 64, 128, 256]w.tanh$	128	69.89325

We can see, AE with more layers and non-linear activation gives lower reconstruction error than both PCA and AE with single layer. This implies that with non-linearity, the autoencoder can learn to make a more accurate reconstruction.