Social Network Node Analysis and Community Mining

Yuquan Zhou*
21307100050@m.fudan.edu.cn

Zhenchu Xiao*
22300180074@m.fudan.edu.cn

Bo Li*
21307110183@m.fudan.edu.cn

In our project, the workload is evenly distributed among the team.

1 Dataset Intro

1.1 Cora

The CORA dataset is a well-known benchmark in machine learning and graph-based research, particularly for tasks involving graph neural networks (GNNs). It consists of a citation network where nodes represent academic papers, and edges represent citation relationships between those papers.Table 1 and Figure 1 are the detailed information.

Cora Dataset Statistics	
Nodes	2708
Edges	5429
Number of features	1433
Nodes Categories	7

Categories: Case_Based, Theory, Rule_Learning, Neural_Networks, Probabilistic_Methods, Genetic_Algorithms, Reinforcement_Learning

Table 1: Cora database

1.2 Facebook

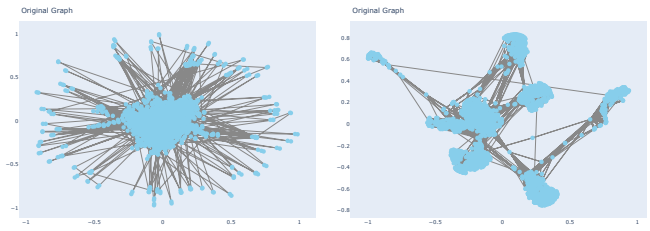
The Facebook dataset is a rich graph-based dataset that contains social interaction information, which is widely used for social network analysis, graph neural network research, and recommendation systems. It provides abundant node and edge data, allowing researchers to explore social network structures, relationships, and user behavior patterns.Table 2 and Figure 1 are the detailed information.

Facebook Dataset Statistics	
Nodes	4039
Edges	88234
Fraction of closed triangles	0.2647
Diameter(longest shortest path)	8
Average clustering coefficient	0.6055

Table 2: Facebook database

2 Community Mining

Social networks are often described using complex networks, which feature a topology of node interconnections that combine both organization and randomness. Community detection, also known as graph partitioning, helps uncover hidden relationships between nodes in a network. It can be applied in machine learning to detect groups with similar attributes and to extract communities for various purposes. In this section, we will primarily discuss the well-known Louvain algorithm, Infomap algorithm, Label Propagation, and the corresponding asynchronous Label Propagation algorithm.



(a) CORA Dataset **(b) FACEBOOK Dataset**

Figure 1: CORA and Facebook Datasets

2.1 Louvain

Algorithm 1: Louvain Algorithm

Input: Graph $G = (V, E)$
Output: Communities C
Initially, assign each node to its own community
repeat
  for *each node* i **do**
    for *each neighbor* j *of node* i **do**
      Calculate the modularity gain ΔQ
      if $\Delta Q > 0$ **then**
        Assign node i to community of j
until *No community assignments change*
Condense each community into a supernode
Update the edge weights: internal edges become self-loop weights, and inter-community edges sum to the weight between supernodes
Calculate the modularity Q of the new graph
if *Modularity* Q *does not change* **then**
  Terminate the algorithm
else
  Repeat the process from step 2

The quality of community division is defined by the modularity equation in the **Louvain** [4] algorithm:

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j), \tag{1}$$

where:

- A_{ij} represents the weight of the edge between node i and node j,

- $k_i = \sum_j A_{ij}$ is the sum of the weights of edges connected to node i,
- c_i is the community to which node i belongs,
- $\delta(c_i, c_j)$ is an indicator function, which equals 1 if $c_i = c_j$, and 0 otherwise.

Using this modularity equation, we can easily calculate the change in modularity when a node i is moved into a community C. The formula is:

$$\Delta Q = \left[\frac{\sum_{\text{in}} + k_{i,\text{in}}}{2m} - \left(\frac{\sum_{\text{tot}} + k_i}{2m} \right)^2 \right] - \left[\frac{\sum_{\text{in}}}{2m} - \left(\frac{\sum_{\text{tot}}}{2m} \right)^2 - \left(\frac{k_i}{2m} \right)^2 \right]. \tag{2}$$

Then, refer to Algorithm 1 for the detailed description of the Louvain algorithm.

2.2 Infomap

Infomap starts with a random walk beginning from a certain node j. It then probabilistically transitions to the next node i with probability $p(i|j)$, and from node i, the walk continues to the next node according to the transition probabilities. This process is repeated, generating a long sequence. The essence of Infomap is to use the minimization of the encoding of the random walk sequence and its duality to community detection. By minimizing the encoding, Infomap is able to find a better community partition.

Infomap introduces a two-layer encoding scheme for the graph. The first layer encodes the communities, where each community has a unique encoding. The second layer encodes the nodes within the same community. Note that, since the community encodings distinguish different communities, nodes from different communities can share the same node encoding. Inspired by information encoding, a good community partition leads to shorter code lengths in the structure described above. Starting from the concept of Shannon entropy, the author proposes a lower bound on the coding length for the module partition M, represented by the following mapping equation:

$$L(M) = q_{\curvearrowright} H(\mathscr{Q}) + \sum_{i=1}^{m} p^i_{\circlearrowright} H(\mathscr{P}^i). \tag{3}$$

Where $H(\mathscr{Q})$ represents the frequency-weighted average code length of the index codebook, and $H(\mathscr{P}^i)$ is the frequency-weighted average code length of the codebook for the i-th module. $q_{i\curvearrowright}$ denotes the probability that a random walker exits module i, and the frequency of the index codebook is given by $q_{\curvearrowright} = \sum_{i=1}^{m} q_{i\curvearrowright}$, which is the probability that the walker switches modules at each step. p_α is the probability of visiting node α, and the frequency of the codebook for the i-th module is given by $p^i_{\circlearrowright} = \sum_{\alpha \in i} p_\alpha + q_{i\curvearrowright}$. Therefore, the entropy of the index codebook is:

$$H(\mathscr{Q}) = -\sum_{i=1}^{m} \frac{q_{i\curvearrowright}}{\sum_{j=1}^{m} q_{j\curvearrowright}} \log \left(\frac{q_{i\curvearrowright}}{\sum_{j=1}^{m} q_{j\curvearrowright}} \right) \quad . \tag{4}$$

And the entropy of the module codebook i is calculated as:

$$H(\mathscr{P}^i) = -\frac{q_{i\curvearrowright}}{q_{i\curvearrowright} + \sum_{\beta \in i} p_\beta} \log \left(\frac{q_{i\curvearrowright}}{q_{i\curvearrowright} + \sum_{\beta \in i} p_\beta} \right) - \sum_{\alpha \in i} \frac{p_\alpha}{q_{i\curvearrowright} + \sum_{\beta \in i} p_\beta} \log \left(\frac{p_\alpha}{q_{i\curvearrowright} + \sum_{\beta \in i} p_\beta} \right) \tag{5}$$

Algorithm 2: IofoMap Algorithm

Input: Graph $G = (V, E)$, random jump probability p, hyperparameters
Output: Community assignments for each node
Initialize community for each node: $C(v) = v$ for all $v \in V$
while *community assignments change* **do**
  for *each node* $v \in V$ **do**
    Sample a sequence of nodes using random walk starting from v
    for *each node* w *in the sequence* **do**
      Calculate the current community assignment $C(w)$
      Compute $L(M)$ for the current assignment
      if *The change in* $L(M)$ *is the greatest* **then**
        Assign node v to the community of w

Based on the above principles, the fundamental steps of the Infomap algorithm are outlined in Algorithm 2.

2.3 Label Propagation

The **Label Propagation** [5] algorithm works as follows: Assume that for node x, its neighbors are $x_1, x_2, \ldots, x_k$. Each node has a corresponding label, which represents the community to which the node belongs. The label update process in the Label Propagation algorithm can be divided into two types: **Synchronous Update** and **Asynchronous Update**.

For the synchronous update, the algorithm proceeds as follows:

(1) For each node x, initialize its community label as $C_x^{(0)} = x$.
(2) Set the iteration count t.
(3) Define a traversal order for the nodes in set X.
(4) For each node $x \in X$, update its community label as:
$$C_x^{(t)} = f\left(C_{x_1}^{(t-1)}, C_{x_2}^{(t-1)}, \ldots, C_{x_k}^{(t-1)}\right)$$
where the function f assigns the most frequent label among the neighbors' community labels.
(5) If the algorithm has not converged, increment t and repeat from step 4.

In the synchronous update method, all nodes update their labels in parallel using the formula above. The potential issue with this method is that it may cause label oscillations, especially in bipartite or nearly bipartite networks.

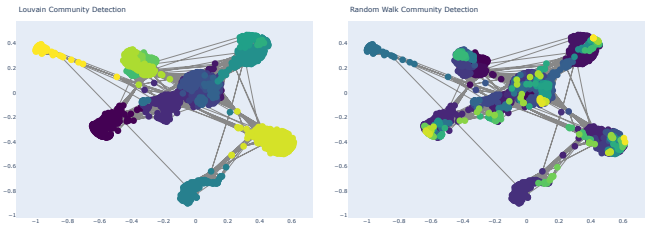
Asynchronous Update: In the asynchronous update method, the update formula remains the same:

$$C_x^{(t)} = f\left(C_{x_{i_1}}^{(t)}, \ldots, C_{x_{i_m}}^{(t)}, C_{x_{i_{m+1}}}^{(t-1)}, \ldots, C_{x_{i_k}}^{(t-1)}\right),$$

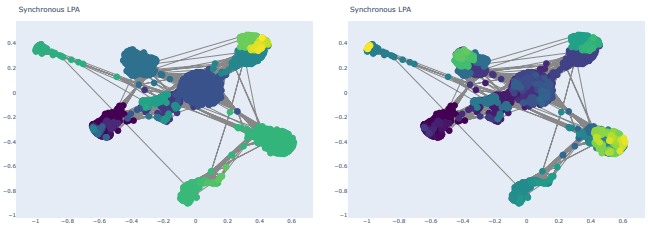
but the labels of neighbors $x_{i_1}, \ldots, x_{i_m}$ are updated at iteration t, while those of neighbors $x_{i_{m+1}}, \ldots, x_{i_k}$ are kept from iteration $t-1$ if they have not been updated yet.

2.4 Results

We applied the four algorithms mentioned above to perform community detection on the ego-Facebook dataset, yielding the following results.



(a) Louvain Algorithm (b) Infomap Algorithm



(c) Synchronous LPA (d) Asynchronous LPA

Figure 2: Community detection

	Louvain	Infomap	Sync LPA	Async LPA
Num. of Comm	17	75	44	82
modularity	0.835	0.812	0.737	0.812

Table 3: Comparison of community detection results among Louvain, Infomap, Sync LPA, and Async LPA.

The visualization, created using the Plotly package in Python, shows that Async LPA identifies more communities, while Louvain performs better in terms of modularity. Since Louvain is designed to optimize modularity, it is expected to outperform on this metric. However, Async LPA is also an excellent method that may uncover more refined community structures.

3 Network Analysis

In this section, we will use various methods to explore the network of the Ego-Facebook dataset from different aspects(node centrality, graph metrics, etc.), and present the results of conducted simulation experiments using various network models.

3.1 Node Centrality

Node centrality is a fundamental component of network analysis, which provides insight into the significance of individual nodes based on their structural positions in the graph. Common centrality metrics include degree centrality[10], eigenvector centrality[6], Katz centrality[13], between centrality[20], closeness centrality[20], and Pagerank centrality[7]. Specific definitions of these metrics are listed in table 4. We used all these metrics for the computation of the centrality of the node and plotted the visualization result.

Centrality Metric	Formula/Description
Degree Centrality	$C_D(v) = \deg(v)$
Eigenvector Centrality	$C_E(v) = \frac{1}{\lambda} \sum_{u \in N(v)} A_{uv} C_E(u)$
Katz Centrality	$C_K(v) = \alpha \sum_{u \in N(v)} A_{uv} C_K(u) + \beta$
Betweenness Centrality	$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$
Closeness Centrality	$C_C(v) = \frac{1}{\sum_{u \neq v} d(v,u)}$
PageRank Centrality	$PR(v) = (1-d) + d \sum_{u \in N(v)} \frac{PR(u)}{\deg(u)}$

Table 4: Centrality Metrics and Their Formulas

Figure 3 show the results of different centrality measures for nodes in a subgraph related to a community randomly selected from the output of the Louvain algorithm. The nodes are divided into 4 groups by the percentile of their centrality value(50%, 75%, 90%). It can be seen all 6 methods give reasonable and similar result.

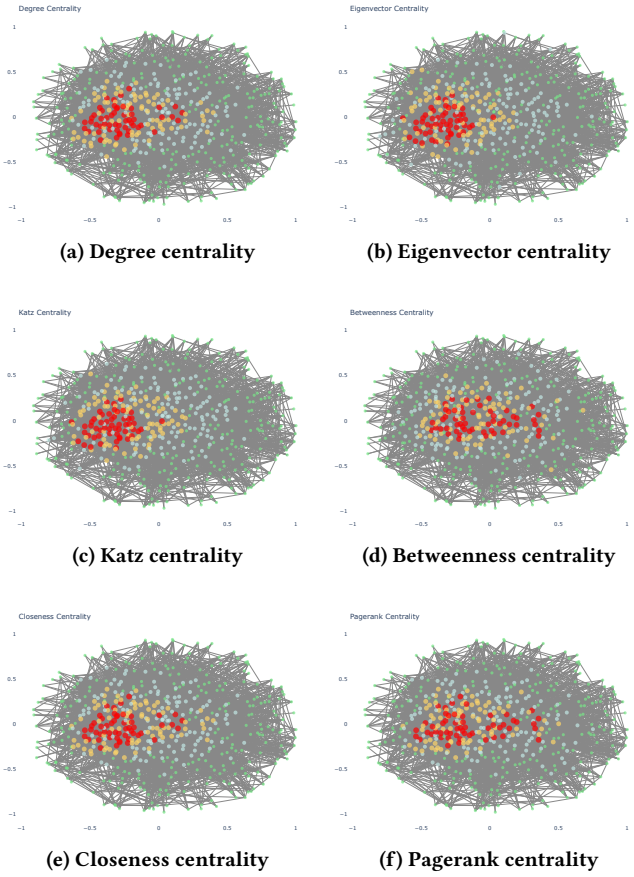
3.2 Network Metrics

Common metrics for a network include the number of nodes and edges, clustering coefficient, average distance, and diameter. For comparison, we generated random graphs with similar number of nodes and edges using models including Erdos-Renyi model of stochastic graph(ER), Watts-Strogatz model of small world, and Barabási–Albert model of preferential attachment network(ba). Metrics of the original dataset(Ego-Facebook) and networks generated with models are computed and listed in table 5. We also calculated the degree distribution of the networks. Results are shown in fig. 4

	Facebook	ER	BA	WS
Nodes	4039	4039	4039	4039
Edges	88234	88163	88374	88858
Clustering	0.606	0.011	0.038	0.712
Avg. Dist.	3.692	2.606	2.512	4.148
Diameter	8	4	4	7

Table 5: Network Statistics Comparison

It can be seen from the results that none of the generated networks perfectly reflect all attributions of real-world network. The clustering coefficient and diameter of ER and BA model is too low,



(a) Degree centrality (b) Eigenvector centrality

(c) Katz centrality (d) Betweenness centrality

(e) Closeness centrality (f) Pagerank centrality

Figure 3: Centrality measures: Degree, Eigenvector, Katz, Betweenness, Closeness, and Pagerank.

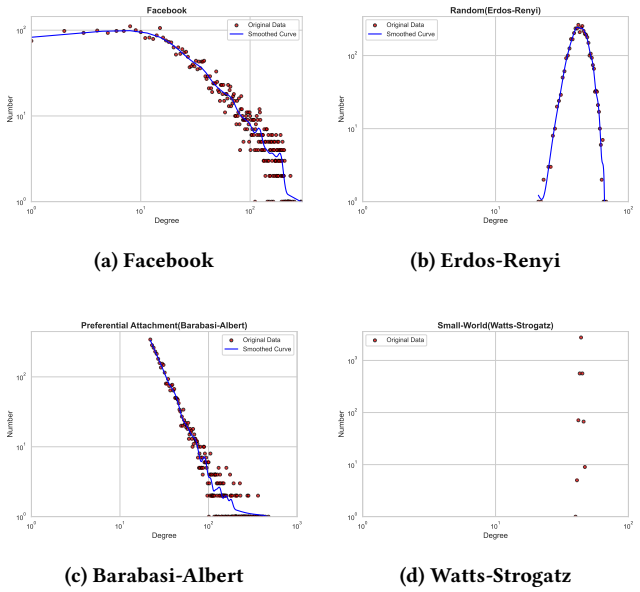
while although WS model have similar metric values with Ego-Facebook, the degree distribution of it is nearly uniform rather than power-law distribution showcased in the dataset network.

3.3 Network Evolution

We chose the WS which has the most similar metric values with real dataset to simulate the evolution of a network. The scale of the network is of 500 nodes, and by setting $c = 2, 4, 6, 16$ respectively, we can see how the network evolves as the number of edges increase. Results are shown in fig. 5.

4 Link prediction

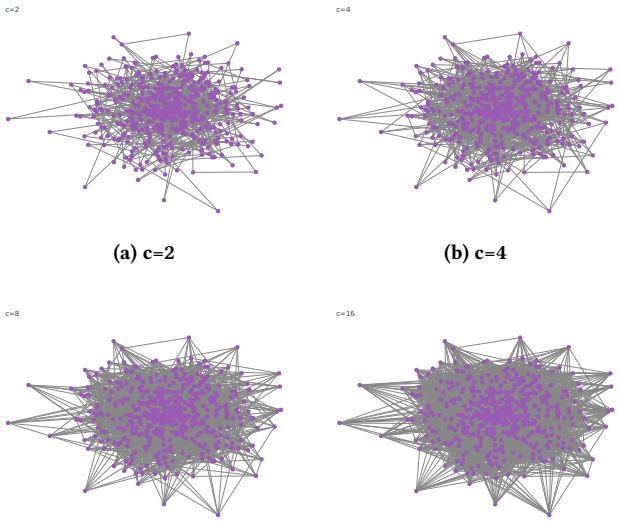
Link prediction is a fundamental task in network science, which aims at identifying missing links in static networks or predicting the likelihood of future links in dynamic networks. Traditional link prediction methods can be broadly classified into three categories: similarity-based approaches, probabilistic models, and dimensionality reduction techniques. Similarity based methods compute scores based on structural or node properties, leveraging metrics including common neighbors, preferential attachment, and Adamic/Adar indices, etc. [17][3][1]. Probabilistic models, such as the Stochastic Block Models(SBM) and Hierarchical Structure Model (HSM),



(a) Facebook (b) Erdos-Renyi

(c) Barabasi-Albert (d) Watts-Strogatz

Figure 4: Degree distribution of Ego-Facebook and other models



(a) c=2 (b) c=4

(c) c=8 (d) c=16

Figure 5: Network evolution of WS model with $c = 2, 4, 8, 16$.

use statistical frameworks to estimate link probabilities[12][11]. Dimensionality reduction methods, including matrix factorization and network embeddings, represent nodes in a low-dimensional latent space to capture structural and attribute-based relationships[9][19]. In recent years, deep learning-based methods have occupied the mainstream. Among them GCN[15], VGAE[16], NBFNe[24], Walk-pooling[18] and NESS [21] all achieved SOTA performance at the time.

4.1 similarity-based approaches

Feature	Formula
Common Neighbors	$\lvert\mathcal{N}(x) \cap \mathcal{N}(y)\rvert$
Jaccard Coefficient	$\frac{\lvert\mathcal{N}(x)\cap\mathcal{N}(y)\rvert}{\lvert\mathcal{N}(x)\cup\mathcal{N}(y)\rvert}$
Resource Allocation	$\sum_{z\in\mathcal{N}(x)\cap\mathcal{N}(y)} \frac{1}{\lvert\mathcal{N}(z)\rvert}$
Preferential Attachment	$\lvert\mathcal{N}(x)\rvert \cdot \lvert\mathcal{N}(y)\rvert$
Dice Coefficient	$\frac{2\lvert\mathcal{N}(x)\cap\mathcal{N}(y)\rvert}{\lvert\mathcal{N}(x)\rvert+\lvert\mathcal{N}(y)\rvert}$
Cosine Similarity	$\frac{2\lvert\mathcal{N}(x)\cap\mathcal{N}(y)\rvert}{\sqrt{\lvert\mathcal{N}(x)\rvert}\cdot\sqrt{\lvert\mathcal{N}(y)\rvert}}$
Adamic-Adar	$\sum_{z\in\mathcal{N}(x)\cap\mathcal{N}(y)} \frac{1}{\log\lvert\mathcal{N}(z)\rvert}$
Pearson Correlation	$\frac{\sum_k (A_{i,k}-\bar{A}_i)(A_{j,k}-\bar{A}_j)}{\sqrt{\sum_k (A_{i,k}-\bar{A}_i)^2}\sqrt{\sum_k (A_{j,k}-\bar{A}_j)^2}}$

Table 6: Graph-based similarity features and their formulas.

Any method for calculating the similarity between nodes can be applied to the link prediction task. In this project, we primarily used the nine similarity calculation methods listed in table 6, and combined them with classical classification models, such as Decision Trees and KNN, for link prediction.

Ego-Facebook dataset		
Model	**AUC**	**AP**
Decision Tree	0.834998	0.873852
KNN	0.893061	0.894446
Naive Bayes	0.898779	0.926500
Logistic Regression	0.910104	0.934840
SVM	0.899591	0.928521
MLP	0.916813	0.938999
Node2vec+Logistic	0.990051	0.985951
CORA dataset		
Model	**AUC**	**AP**
Decision Tree	0.929871	0.903929
KNN	0.907618	0.876139
Naive Bayes	0.679915	0.789205
Logistic Regression	0.851754	0.884712
SVM	0.709225	0.814020
MLP	0.938992	0.924284
Node2vec+Logistic	0.861507	0.893164

Table 7: Performance comparison on Facebook and CORA datasets.

Results We performed link prediction on the Ego-Facebook and CORA datasets using similarity-based link prediction methods. Additionally, we implemented a combination of node2vec and logistic regression. The results are presented using AUC (Area Under the ROC Curve) and AP (Average Precision) metrics, as shown in table 7. It can be observed that the results obtained using MLP are consistently high across both datasets. Node2vec + Logistic Regression performs exceptionally well on the Facebook dataset, but its performance on the CORA dataset is less impressive.

4.2 Graph Convolutional Network

In deep learning-based methods, we primarily use GCN and VGAE models for link prediction. Graph Convolutional Network (GCN) is a deep learning model based on graph convolution operations. GCN propagates information in a hierarchical aggregation manner by integrating graph structure information and node features. The key idea is to perform convolution operations on each node and its neighboring nodes to update node representations.

Specifically, GCN utilizes a normalized graph Laplacian operator to implement message passing and updates node representations using the following formula[15]:

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}H^{(l)}W^{(l)}\right), \tag{6}$$

where $\tilde{A} = A+I_N$ is the adjacency matrix of the undirected graph G with added self-connections. I_N is the identity matrix, $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$, and $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function, such as the $ReLU(\cdot) = \max(0, x)$; $H^{(l)} \in \mathbb{R}^{N\times D}$ is the matrix of activations in the l-th layer, and $H^{(0)} = X$.

Figure 6: Schematic depiction of multi-layer Graph Convolutional Network (GCN) for semi-supervised learning with C input channels and F feature maps in the output layer. The graph structure (edges shown as black lines) is shared over layers, labels are denoted by Y_i.

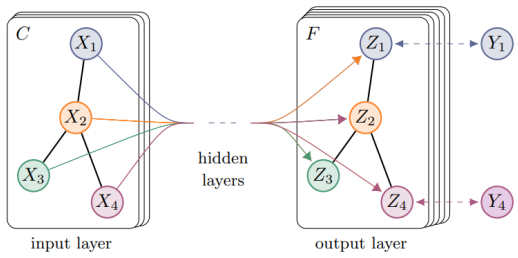


Figure 6: Graph Convolutional Network

4.3 Variational Graph Auto-Encoder

Variational Graph Auto-Encoder (VGAE) is an unsupervised learning model for graph-structure data that excels in tasks like link prediction and graph completion. VGAE employs a two-part framework: a GCN-based encoder that learns a probabilistic distribution over latent representations for each node, and a decoder that reconstructs the graph structure by predicting the adjacent matrix. We adopt this model for its competitive performance, theoretical

simplicity and confound impact on following studies, for example, S-VGAE[8], VGNAE[2], and NESS[21].

4.3.1 Preliminary: Variational Auto-Encoders. Variational Auto-Encoders (VAE)[14] are a type of generative models that extends traditional auto-encoders by introducing a probabilistic framework. The VAE comprises two components: the encoder, which maps input data to the parameters of the latent distribution(mean and variance), and the decoder, which reconstructs the input from samples drawn from this latent distribution. The probabilistic nature of the encoder helps extend the latent space and stabilizes the decoder.

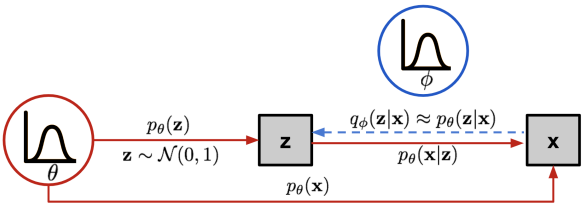


Figure 7: Variational Auto-Encoder

As shown in fig. 7[23], the encoder is a estimated posterior distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ parameterized by ϕ, and the decoder is the likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. The sampling process is the red arrows: first sample $\boldsymbol{z}$ from the latent distribution, and then use the decoder to map $\boldsymbol{z}$ to $\boldsymbol{x}$. The training target is an MLE problem, i.e. to maximize the probability of generating the real data samples:

$$\boldsymbol{\theta}^* = \mathbf{argmax}_{\boldsymbol{\theta}} \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) = \mathbf{argmax}_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}). \quad (7)$$

However, when we formulate the problem as a posterior-likelihood form, we also want to minimize the difference between the estimated posterior and the real one. It can be proved by simple calculation that:

$$\begin{aligned} &\log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})), \end{aligned} \quad (8)$$

where the LHS is exactly what we want to maximize, and the RHS, which is refered to as the evidence lower bound(ELBO) can be the loss function. It contains two parts: the reconstruction loss(first term) and the difference between estimated posterior and designated latent distribution.

4.3.2 Model. In VGAE, the encoder layer is specified as a two-layer GCN, which outputs mean μ and std σ of posterior latent distribution. It is worth noticing that the weight is shared for the first layer. To specify:

$$\begin{aligned} \mu &= \mathrm{GCN}_1(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0)\mathbf{W}_1, \\ \sigma &= \mathrm{GCN}_2(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\,\mathrm{ReLU}(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0)\mathbf{W}_2, \end{aligned} \quad (9)$$

where$\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2$. The latent is then sampled from the normal distribution $q(\mathbf{z}_i \mid \mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i \mid \boldsymbol{\mu}_i, \mathrm{diag}\boldsymbol{\sigma}_i^2))$. The decoder layer is:

$$\hat{\mathbf{A}} = \sigma(\mathcal{Z}\mathcal{Z}^\top), \quad (10)$$

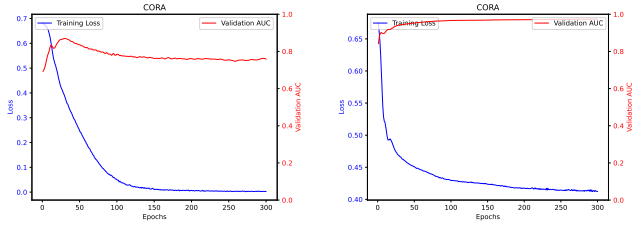
i.e. the connection probability of node i and node$j$ is predicted by $\mathbf{A}_{ij} = \sigma(\mathbf{z}_i\mathbf{z}_j^\top)$

4.4 Experiments

Experiments are conducted on both Ego-Facebook and CORA. The training is carried out in transductive setting. All nodes are included in training, validation and testing dataset, while edges are split by proportion of 70%, 10%, and20% respectively as positive samples. Negative samples are generated for validation set and testing set from node pairs that are not connected with an edge. Training edges, positive and negative samples for validation and testing sets are NOT overlapped by any means. Also, only training edges serve as message edges which transfers messages during training, and validation and testing samples are excluded from computation of training loss. These setting should ensure there no leakage of information among training, validation and testing dataset. Moreover, different weight is placed on positive and negative samples for balance of classes.

The feature vector of nodes differs between two datasets. As CORA provides information of articles, we use these information as features directly. For the feature-free Ego-Facebook, we adopted Node2Vec to extract a 128-dim feature vector from the graph structure. Note that the feature is extracted from training edges only.

4.4.1 GCN. We trained the model for 300 epochs with a learning rate of 0.005 using the Adam optimizer on both the Ego-Facebook and Cora datasets. On the CORA dataset, we achieved a Best Test ROC of 86.5354% and a Best Test AP of 86.4887%. On the Ego-Facebook dataset, we achieved a Best Test ROC of 97.5682% and a Best Test AP of 97.2526%. The training history can be seen in Figure 8.We can observe a slight overfitting on the Cora dataset.



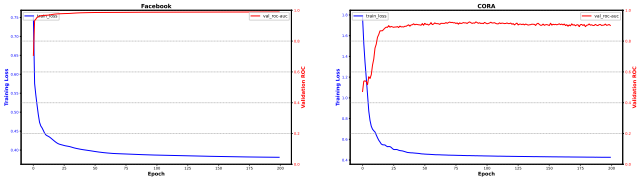
(a) Cora **(b) Facebook**

Figure 8: Training history of GCN on Ego-Facebook and CORA.

4.4.2 VGAE. We trained the model for 200 epochs using Adam optimizer and learning rate is set to 0.01. We reached test AP of 92.2% and test AUROC of 91.0% on CORA, and 98.8% and 99.0% for Ego-Facebook respectively. In addition, we also calculated the approximate test accuracy, taking into account of the ratio of edges and non-edges. The result is 55.6% and 77.8% for CORA and Ego-Facebook respectively. Training history is visulized in fig. 9

4.5 Variations of VGAE

Many variations of VGAE have arisen in recent years and have achieved new SOTA results in the link prediction task. Ahn and



(a) Facebook (b) Cora

Figure 9: Training history of VGAE on Ego-Facebook and CORA.

Kim introduced a variational graph normalized auto-encoder model (VGNAE)**??**. The main contribution is that they add a normalizing step after the transformation of features before aggregating, i.e.:

$$\mathrm{GNCN}(\mathbf{X}, \mathbf{A}, s) = s\tilde{\mathbf{D}}^{-1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-1/2}g(\mathbf{XW}), \tag{11}$$

where $g([h_1, h_2, \ldots, h_n]) = \left[\frac{h_1}{\|h_1\|}, \frac{h_2}{\|h_2\|}, \ldots, \frac{h_n}{\|h_n\|}\right]$ and s is a constant. Also, when calculating the potential z, they puts a weight on normalized neighbor features according to the degree of nodes:

$$z_i = \frac{1}{d_i + 1}n_i + \sum_{j \in N(i)} \frac{1}{\sqrt{d_i + 1}\sqrt{d_j + 1}}n_j, \tag{12}$$

where n_i is the normalized feature of the i^{th} node.

There are also works on the training scheme. For example, Ucar[21] introduces a paradigm of training on several subgraphs jointly. For training, they first divide the graph into several distjoint subgraphs, and then calculate the node embeddings independently from these subgraphs. The reconstruction loss is computed between reconstructed subgraph and original subgraph, and are aggregated together to derive the total loss. They also introduce a optional contrastive loss between embeddings derived from different subgraphs. The loss function is summarized as:

$$\mathcal{L}_t = \mathcal{L}_r + \alpha\mathcal{L}_c, \tag{13}$$

where

$$\begin{aligned}\mathcal{L}_r &= \frac{1}{K}\sum_{k=1}^{K} l_k \\ &= \frac{1}{K}\sum_{k=1}^{K} -\frac{1}{2n_{kp}}\sum_{i=1}^{n_{kp}}\left[\log \hat{e}^{+}_{ki} + \log(1 - \hat{e}^{-}_{ki})\right] \\ \mathcal{L}_c &= \frac{1}{J}\sum_{\{\boldsymbol{h}_a, \boldsymbol{h}_b\} \in S} p(\boldsymbol{h}_a, \boldsymbol{h}_b),\end{aligned}$$

where $p(\cdot, \cdot)$ is the contrastive loss between latent of two subgraphs. (14)

This enables each node to treat its local neighborhood in different subgraphs as multiple views for link prediction, enabling the model to leverage information from different perspectives.

5 Node Classification

Node Classification is a type of graph machine learning task aimed at predicting the category labels of nodes in a graph. In this section, we will focus on a recent and efficient modeling tool—Graph Neural Networks (GNNs)—and implement the node classification task using two classic methods within this framework: Graph Convolutional Networks (GCN) and Graph Attention Networks (GAT). We will also compare their strengths and weaknesses.

5.1 GAT

The Graph Attention Network (GAT) introduces an attention mechanism, further enhancing the flexibility of graph convolution. By assigning different weights to different neighbors, GAT is able to give higher importance to the information from significant neighbors, thereby improving the model's performance.

GAT primarily implements node updates through the following formula[22]:

$$\vec{h}'_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}\mathbf{W}\vec{h}_j\right). \tag{15}$$

Here, $\vec{h}$ is a set of node features the input to our layer. $\vec{h}'$ is the output. $\alpha_{ij} = \mathrm{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i}\exp(e_{ik})}$ and e_{ij} denotes $LeakyReLU\left(\vec{a}^{\top}\left[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j\right]\right)$;$\vec{a}$ denotes the role of a single-layer feedforward neural network; and $\mathbf{W}$ denotes the trainable weight matrix, same meaning as $W^{(l)}$ in GCN.

Figure 10 :Left:The attention mechanism $a(\mathbf{W}\tilde{h}_i, \mathbf{W}\tilde{h}_j)$ employed by our model, parametrized by a weight vector $\tilde{\mathbf{a}} \in \mathbb{R}^{2F'}$, applying a LeakyReLU activation. **??** :Right:An illustration of multi-head attention (with $K = 3$ heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\tilde{h}'_1$.

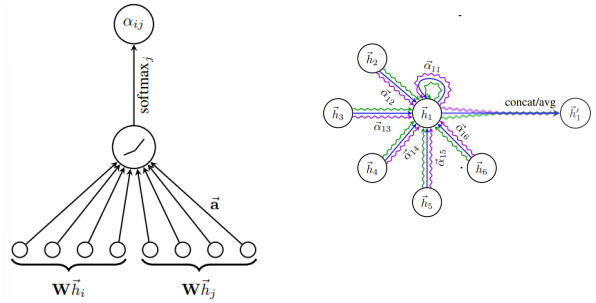


Figure 10: Graph Attention Network

5.2 Result

We selected a portion of nodes from each category in the Cora and Facebook datasets to serve as the training and test sets. Table 8 and table 9 show the log loss and accuracy on the test set after every 10/100 epochs when using the GCN method for node classification.

From the results, we can see that the performance of GCN is slightly better than GAT. However, in our experiments, it was observed that GCN reached its peak accuracy at 438 iterations, while GAT reached its peak at 51 iterations.This indicates that GCN has better predictive performance, while GAT achieves faster iterative convergence.

	LOSS		ACCURACY	
EPOCH	GCN	GAT	GCN	GAT
100	0.06	0.81	0.79	0.81
200	0.04	0.77	0.78	0.80
300	0.02	0.71	0.79	0.80
400	0.02	0.69	0.79	0.80
500	0.01	0.61	0.80	0.80

GCN performs best in 438 epoch,acc=0.81
GAT performs best in 51 epoch,acc=0.82

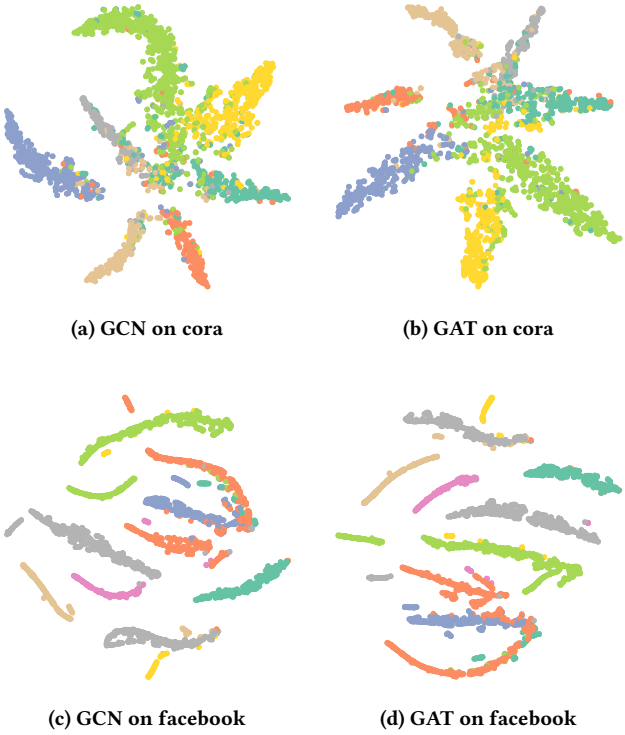
Table 8: Based on Cora database

	LOSS		ACCURACY	
EPOCH	GCN	GAT	GCN	GAT
10	0.57	1.67	0.92	0.85
20	0.25	1.28	0.95	0.92
30	0.15	1.04	0.95	0.94
40	0.13	0.93	0.96	0.94
50	0.13	0.83	0.96	0.94

GCN performs best in 26 epoch,acc=0.97
GAT performs best in 48 epoch,acc=0.96

Table 9: Based on Facebook database

Figure 11 illustrates the hidden layer activations of a two-layer GCN and GAT on the training set after t-SNE visualization, with colors representing document categories.



(a) GCN on cora **(b) GAT on cora**

(c) GCN on facebook **(d) GAT on facebook**

Figure 11: t-SNE visualization

References

[1] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social networks* 25, 3 (2003), 211–230.

[2] Seong Jin Ahn and MyoungHo Kim. 2021. Variational graph normalized autoencoders. In *Proceedings of the 30th ACM international conference on information & knowledge management*. 2827–2831.

[3] Albert-Laszlo Barabâsi, Hawoong Jeong, Zoltan Néda, Erzsebet Ravasz, Andras Schubert, and Tamas Vicsek. 2002. Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications* 311, 3-4 (2002), 590–614.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[6] Phillip Bonacich. 1972. Factoring and weighting approaches to status scores and clique identification. *Journal of mathematical sociology* 2, 1 (1972), 113–120.

[7] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems* 30, 1-7 (1998), 107–117.

[8] Tim R Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M Tomczak. 2018. Hyperspherical variational auto-encoders. *arXiv preprint arXiv:1804.00891* (2018).

[9] Daniel M Dunlavy, Tamara G Kolda, and Evrim Acar. 2011. Temporal link prediction using matrix and tensor factorizations. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 5, 2 (2011), 1–27.

[10] Linton C Freeman et al. 2002. Centrality in social networks: Conceptual clarification. *Social network: critical concepts in sociology. Londres: Routledge* 1 (2002), 238–263.

[11] Roger Guimerà and Marta Sales-Pardo. 2009. Missing and spurious interactions and the reconstruction of complex networks. *Proceedings of the National Academy of Sciences* 106, 52 (2009), 22073–22078.

[12] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. 1983. Stochastic blockmodels: First steps. *Social networks* 5, 2 (1983), 109–137.

[13] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.

[14] Diederik P Kingma. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[15] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[16] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[17] Mark EJ Newman. 2001. Clustering and preferential attachment in growing networks. *Physical review E* 64, 2 (2001), 025102.

[18] Liming Pan, Cheng Shi, and Ivan Dokmanić. 2021. Neural link prediction with walk pooling. *arXiv preprint arXiv:2110.04375* (2021).

[19] Antonio Pecli, Bruno Giovanini, Carla C Pacheco, Carlos Moreira, Fernando Ferreira, Frederico Tosta, Júlio Tesolin, Marcio Vinicius Dias, Maria Claudia Cavalcanti, Ronaldo Goldschmidt, et al. 2015. Dimensionality reduction for supervised learning in link prediction problems. In *International Conference on Enterprise Information Systems*, Vol. 2. SCITEPRESS, 295–302.

[20] Claude Elwood Shannon. 1949. Communication in the presence of noise. *Proceedings of the IRE* 37, 1 (1949), 10–21.

[21] Talip Ucar. 2023. NESS: Node Embeddings from Static SubGraphs. *arXiv preprint arXiv:2303.08958* (2023).

[22] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[23] Lilian Weng. 2018. From Autoencoder to Beta-VAE. *lilianweng.github.io* (2018). https://lilianweng.github.io/posts/2018-08-12-vae/

[24] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. 2021. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems* 34 (2021), 29476–29490.