# An In-depth Study of Machine Learning on Graphs

Lesi Chen, Junzhe Jiang, Miaopeng Yu

**Abstract**

We finish some classic tasks of machine learning on graphs. First, we compare two representative methods(modularity-based and coding-based) to detect communities in networks. Second, we analyze and visualize the attributes of different networks(real networks and synthetic networks), and we also observe the evolution of networks. Third, we reproduce some famous models to classify the nodes in networks, including graph neural networks. Furthermore, we applied techniques for both shallow and deep graph convolution networks, outperforming the original methods. Fourth, we use graph auto-encoders for link prediction to achieve the performance of state-of-the-art, and introduce two important techniques to tackle the KL-vanishing problem in variational graph auto-encoders.

**Keywords**

Network analysis, Community detection, Node classification, Link prediction

## Contents



## 1. Community detection

Social, technological and information systems can often be described in terms of complex networks that have a topology of interconnected nodes combining organization and randomness. To study the network, a promising approach consists in decomposing the networks into sub-units or communities, which are sets of highly interconnected nodes.

We use Infomap[1] and Louvain algorithm[2] to detect the communities below.

### 1.1 Infomap

Consider random walk, starting from a certain point j, in accordance with the probability p(i|j) jump to the next point, and then jump to the next point in the same way, repeat the process. Therefore, we can get a really long sequence. Infomap detect communities in a network by minimizing the coding length of the sequence.

#### 1.1.1 Two-layer coding

Infomap[1] uses a two-layer coding rule to encode the above sequence. The two-layer coding uses a two-layer structure to divide different nodes into groups, and two types of information need to be distinguished during coding. The first is the name of the group, and the name encoding of different groups is different; the second is the node (and the exit group flag) within each group, and the name of the different node has different encoding. Note that in the coding rule the codes of nodes in different groups can be reused.

Infomap's two-layer coding method connects community discovery with information coding. A good group division can lead to shorter codes. Therefore, if the group division that makes the random walk sequence length the shortest is found, then a good group division is found.

According to Shannon's information entropy theory, the average coding length of random walk is:

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

Here $H(Q)$ is the frequency-weighted average length of codewords in the index codebook and $H(P^i)$ is frequency weighted average length of codewords in module code book i. Further, the entropy terms are weighted by the rate at which the codebooks are used. With $q_{\curvearrowright}$ for the probability to exit module i, the index codebook is used at a rate $q_{\curvearrowright} = \sum_{i=1}^{m} q_{i\curvearrowright}$, the probability that the random walker switches modules on any given step. With $p_\alpha$ for the probability to visit node $\alpha$, module codebook i is used at a rate $p^i_{\circlearrowright} = \sum_{\alpha \in i} p_\alpha + q_{i\curvearrowright}$, the fraction of time the random walk spends in module i plus the probability that it exits the module and the exit message is used. Now it is straightforward to express the entropies in $q_{\curvearrowright}$ and $p_\alpha$. For the index codebook, the entropy is

$$H(Q) = -\sum_{i=1}^{m} \frac{q_{i\curvearrowright}}{\sum_{j=1}^{m} q_{j\curvearrowright}} \log\left(\frac{q_{i\curvearrowright}}{\sum_{j=1}^{m} q_{j\curvearrowright}}\right) \tag{2}$$

and for module codebook i the entropy is

$$\begin{aligned} H\left(P^i\right) = &- \frac{q_{i\curvearrowright}}{q_{i\curvearrowright} + \sum_{\beta \in i} p_\beta} \log\left(\frac{q_{i\curvearrowright}}{q_{i\curvearrowright} + \sum_{\beta \in i} p_\beta}\right) \\ &- \sum_{\alpha \in i} \frac{p_\alpha}{q_{i\curvearrowright} + \sum_{\beta \in i} p_\beta} \log\left(\frac{p_\alpha}{q_{i\curvearrowright} + \sum_{\beta \in i} p_\beta}\right) \end{aligned} \tag{3}$$

#### 1.1.2 Algorithm

Infomap takes a similar approach to PageRank to obtain the access probability of all nodes.

With the access probability of all nodes, Infomap detects the communities in the network by:

(1) Initialize each node as an independent community.

(2) Randomly sample a sequence of nodes in the graph, and try to assign each node to the community where the neighbor node is located in order, and assign the community where the average bit drops $L(M)$ the most to the node.

(3) Repeat Step2 until $L(M)$ can no longer be optimized.

### 1.2 Louvain

Louvain algorithm is another representative algorithm for community detection.

#### 1.2.1 Modularity

The modularity of a partition is a scalar value between $-\frac{1}{2}$ and 1 that measures the density of links inside communities as compared to links between communities. In the case of weighted networks, it is defined as

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m}\right] \delta\left(c_i, c_j\right) \tag{4}$$

where $A_{ij}$ represents the weight of the edge between i and j, $k_i = \Sigma_j A_{ij}$ is the sum of the weights of the edges attached to vertex i, $c_i$ is the community to which vertex i is assigned, the $\delta$ function $\delta(u,v)$ is 1 if u = v and 0 otherwise and $m = \frac{1}{2}\sum_{ij} A_{ij}$

#### 1.2.2 Algorithm

Our algorithm is divided into two phases that are repeated iteratively. Assume that we start with a weighted network of N nodes.

**Part I**:

(1)Assign a different community to each node of the network

(2) For each node i we consider the neighbours j of i and we evaluate the gain of modularity that would take place by removing i from its community and by placing it in the community of j. The node i is then placed in the community for which this gain is maximum, but only if this gain is positive.

(3) Apply (2) repeatedly and sequentially for all nodes until no further improvement can be achieved.

**Part II**:

Build a new network whose nodes are now the communities found during the first phase. To do so, the weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities. Once this second phase is completed, it is then possible to reapply the first phase of the algorithm to the resulting weighted network and to iterate.

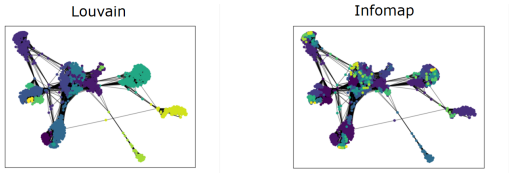


**Figure 1.** Community Detection

| | Louvain | Infomap |
|---|---|---|
| number of communities | 16 | 78 |
| modularity | 0.83 | 0.81 |

**Table 1.** Community detection

The algorithm is efficient because the gain in modularity $\Delta Q$ obtained by moving an isolated node i into a community C can easily be computed by

$$\Delta Q = \left[\frac{\Sigma_{\text{in}} + 2k_{i,\text{ in}}}{2m} - \left(\frac{\Sigma_{\text{tot}} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{\text{in}}}{2m} - \left(\frac{\Sigma_{\text{tot}}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right] \tag{5}$$

where $\Sigma_{in}$ is the sum of the weights of the links inside C, $\Sigma_{tot}$ is the sum of the weights of the links incident to nodes in C, $k_i$ is the sum of the weights of the links incident to node i, $k_{i,in}$ is the sum of the weights of the links from i to nodes in C and m is the sum of the weights of all the links in the network.

A similar expression is used in order to evaluate the change of modularity when i is removed from its community. So we can easily evaluate the change of modularity by removing i from its community and then by moving it into a neighbouring community.

### 1.3 Comparison of community detection methods

The results of Infomap and Louvain algorithm when detecting communities in Facebook-combined are shown in Figure 1 and Table 1.

We can see in the Table that the modularity of communities detected by Infomap and Louvain are both relatively high. And Infomap is capable to divide more communities, indicating that it can find a more refined community structure.

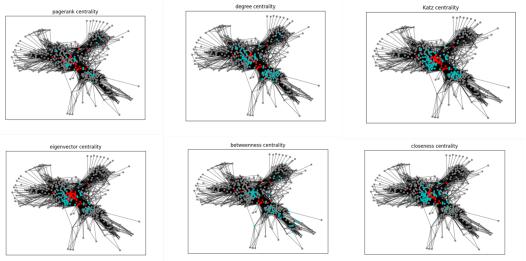

**Figure 2.** Centrality measurement

| | distance | diameter | clustering |
|---|---|---|---|
| Facebook | 3.69 | 8 | 0.606 |
| BA | 4.17 | 9 | 0.015 |
| Small world | 3.03 | 5 | 0.579 |
| Random | 4.42 | 9 | 0.003 |
| Complete | 1 | 1 | 1 |

**Table 2.** Network attribute metrics

## 2. Network analysis

### 2.1 Centrality measurement

We select a community in facebook-combined detect by Louvain algorithm to measure the centrality of the nodes in that community. In Figure 2, we mark the nodes with high centrality red, cyan for medium and gray for low. It can be observed that the results given by the six algorithm are similar, as is expected.

### 2.2 Network attribute metrics

We use ego-facebook and the preferential attachment graph(BA model), small world graph, random graph and complete graph for analysis in Table 2, where the distance denotes the average distance between nodes and clustering denoted the clustering coefficient.

Also, we plotted the degree distribution of different networks in Figure 3. It can be seen that the real network has a smaller average distance and diameter, and a higher clustering coefficient, and the degree distribution conforms to the power-law distribution. Preferential attachment graphs and random graphs do not fit the clustering coefficients well, but degree distribution in preferential attachment graphs conforms to the power-law distribution. The small-world model can better fit the average distance, diameter and clustering coefficient, but the degree distribution does not conform to the real

| time | distance | diameter | clustering |
|---|---|---|---|
| 0 | 1.95 | 2 | 0.58 |
| 1 | 1.92 | 2 | 0.29 |
| 2 | 1.88 | 2 | 0.30 |
| 3 | 1.86 | 2 | 0.32 |
| 4 | 1.83 | 2 | 0.35 |

**Table 3.** ego-facebook evolution

network at all.

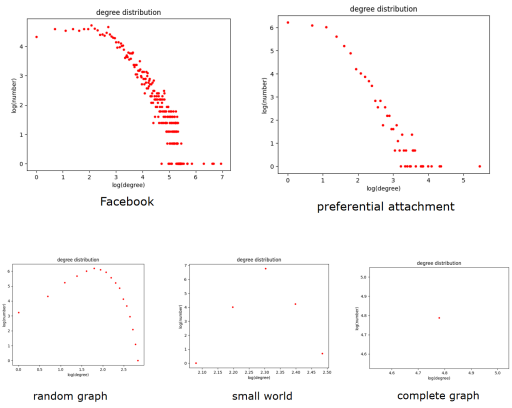


**Figure 3.** degree distribution

### 2.3 Network evolution

First, we use ego-facebook and our link prediction model to view the evolution process. The results are shown in Table 3.

We can observe that the clustering coefficient decreases due to the connecting edges between non-central points. In addition, as more and more edges are linked, the network approaches to the complete graph, leading to increase of the clustering coefficient.

We also use different generation probabilities to see the evolution of the random graph. The results are shown in Table 4 ,we set the number of nodes as 500, and we focus on the largest connected subgraph.

As is shown in Figure 4, when the average degree is about 1, the average distance and diameter reach the maximum, indicating phase transition point is reached, which is also what can be proved by theoretical analysis.

## 3. Node classification

| time | nodes | distance | diameter | clustering | degree |
|---|---|---|---|---|---|
| 0 | 2 | 1.00 | 1 | 0.0 | 0.04 |
| 1 | 9 | 2.83 | 6 | 0.0 | 0.56 |
| 2 | 39 | 6.40 | 16 | 0.0 | 0.96 |
| 3 | 497 | 4.08 | 8 | 0.01 | 4.92 |
| 4 | 500 | 1.90 | 3 | 0.1 | 51.09 |

**Table 4.** random graph evolution

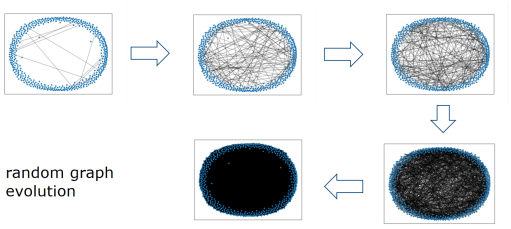


**Figure 4.** random graph evolution

### 3.1 Label Propagation

Label Propagation(LP)[3] is a simple learning-free algorithm for node classification. We first try LP on Cora, and found out that such a simple iterative algorithm can have an unexpected expressive power on Cora, achieving an accuracy of 71.30%. The reason is that in a citation network, papers with the same class tends to cite one another, making up academic circles.

As a simple algorithm, the drawbacks of LP are obvious. First, The outcome will be unstable because a different update sequence may lead to a different classification. Second, and more importantly, without any learnable parameters, LP is less powerful than most machine learning methods.

However, we still regard LP as an important baseline for it discovers the similarity of neighbors, which is a basic idea of all the following methods we use.

### 3.2 Node2Vec

Node2Vec[4] is a node embedding algorithm that computes a vector representation of a node based on random walks in the graph. After learning the node embedding by Node2Vec, we use some classic machine learning method(eg. SVM, random forest, logistic regression) to get the prediction of the label of each node. As a semi-supervised multi-class classification task, we evaluate the cross-entropy error over all labeled examples.

We get the best accuracy using SVM (75.20%) for learning the mapping from the hidden embedding given by Node2Vec to the node labels. As a learning method, Node2Vec outperforms LP much. However, the performance of Node2Vec is still not satisfying. We believed it is mainly because the two following reasons. First, the embedding learned by Node2Vec is not expressive enough as proved in [5, 6], the embedding has a rotation invariance in the hidden space, which is not required for the model to learn. Second, since the whole procedure is not learned end-to-end, the parameters of Node2Vec and SVM may not complement each other well.

### 3.3 Graph Neural Networks

We utilize Graph Neural Networks (GNNs) for better performance. First, we investigate some representative baselines of GNNs, such as GCN[7], GraphSAGE[6] and GAT[8].

Deeply thankful to pytorch geometric, a great framework of GNNs, with the help of which, we finished all the experiments of GNNs.

#### 3.3.1 Graph Convolutional Networks

Graph Convolution Networks (GCN)[7] is a scalable approach for semi-supervised learning on graph-structure data. The convolutional architecture in GCN is motivated by a localized first-order approximation of spectral graph convolution. And the model scales only linearly in the number of graph edges and learns hidden layer representation.

#### 3.3.2 GraphSAGE

GraphSAGE[6] is an inductive algorithm for computing node embeddings. GraphSAGE uses node feature information to generate node embeddings on unseen nodes or graphs. Instead of training individual embeddings for each node, the algorithm learns a function that generates embeddings by sampling and aggregating features from a node′s local neighborhood.

#### 3.3.3 Graph Attention Networks

Graph attention networks (GATs)[8] leverages self-attentional layers to address the shortcomings of prior methods based on graph convolutions or their approximations. By stacking layers in which nodes are able to attend over their neighborhoods' features, GAT enables (implicitly) specifying different weights to different nodes in a neighborhood, without requiring any kind of costly matrix operation or depending on knowing the graph structure upfront.

| Method | Accuracy (%) |
|---|---|
| LP | 71.30 |
| Node2Vec | 75.20 |
| GCN | 81.10 |
| GraphSAGE | 79.60 |
| GAT | 81.19 |

**Table 5.** Node classification on Cora

### 3.4 Comparison of classic GNNs

In the comparison of GNNs, we set train the models for 200 epochs on Cora dataset, using SGD with the learning rate of 0.01 and weight decay of 5e-4, as a general setup. The results are shown in Table 5.

As transductive models, GCN (81.80%) and GAT (81.19%) get much better results than the inductive model GraphSAGE (79.60%), since the node classification task in this project is also a transductive learning task. With attention mechanisms, GAT outperforms GCN little with more computation complexity, more hyperparameters and larger convergence rate. Plus, GCN has a rigorous theoretical derivation which is not provided by GAT.

For comprehensive consideration, we choose GCN as our basic model and also implement it inherited from the MessagePassing class in pytorch geometric.

## 4. Improvement for node classification

### 4.1 Better optimizers

As mentioned above, we use SGD for training GNNs. In this section, we try to use some more advanced optimizers for improvement, including Kronecker-factored approximate curvature(K-FAC)[9] and Sharpness-aware minimization(SAM)[10] .

In our experiment, K-FAC bring less improvement, while SAM is indeed helpful. It is worth noting that we, for the first time, try to use SAM, to train GNNs. We successfully verified that SAM can not only play

an important role in training convolution neural networks(CNNs) in computer vision, but SAM also works in training graph neural networks.

#### 4.1.1 Kronecker-factored approximate curvature

Kronecker-factored approximate curvature (K-FAC) is a second-order optimization method based on natural gradient descent, and it effectively approximates the Fisher information matrix. And the approximation has the form of what is known as a Khatri-Rao product in multivariate statistics.

#### 4.1.2 Sharpness-Aware Minimization

Optimizing only training loss value as is commonly done, can easily lead to suboptimal model quality. And the procedure Sharpness-Aware Minimization(SAM)[10], motivated by prior work connecting geometry of the loss landscape and generalization, an effective procedure simultaneously minimizing loss value and loss sharpness, and this formulation results in a min-max optimization problem on which gradient descent can be performed efficiently:

For any $\rho > 0$, with high probability over training set $\mathscr{S}$ generated from distribution $\mathscr{D}$,

$$L_{\mathscr{D}}(w) \leq \max_{\|\varepsilon\|_2 \leq \rho} L_{\mathscr{S}}(w+\varepsilon) + h(\|w\|_2^2/\rho^2) \tag{6}$$

where $h : \mathbb{R}_+ \to \mathbb{R}_+$ is a strictly increasing function (under some technical conditions on $L_{\mathscr{D}}(w)$). And inspired by the terms from the bound, the author of SAM proposed to select parameter values by solving the following SAM problem:

$$\min_w L_{\mathscr{S}}^{SAM}(w) + \lambda \|w\|_2^2 \tag{7}$$

where $L_{\mathscr{S}}^{SAM}(w) = \max_{\|\varepsilon\|_p \leq \rho} L_{\mathscr{S}}(w+\varepsilon)$, $\rho \geq 0$ is a hyperparameter. In our experiment, we set $\rho$ as 0.5.

Note that SAM is based on other optimizers(eg. SGD, Adam) with approximate to minimize this problem. In our experiment, we use SGD as our base optimizer, with the same learning rate and weight decay we set in training vanilla GCN. Our results are in Table 6.

| Method | Optimizer | Accuracy(%) |
|---|---|---|
| GCN | SGD | 81.1 |
| GAT | SGD | 81.9 |
| GCN | K-FAC | 81.7 |
| GCN | SAM | 82.1 |
| GAT | SAM | 82.3 |

**Table 6.** Node classification with different optimizers

### 4.2 Correct and smooth

Correct and Smooth (C&S)[11] is a graph representation learning method based on iteration. C&S combines LP[3] into simple models(eg. MLP), with a understanding of the essence of GNNs, which can be summarized as correct and smooth.

C&S first uses "error correlation" to correct errors by spread residual errors in training data to test data. And then it uses "prediction correlation" to smooths the predictions on the test data.The residuals $E$ corresponding to training nodes are zero only when the base predictor makes a perfect prediction:

$$\hat{E} = \arg\min_{W \in \mathbb{R}^{n \times c}} trace\left(W^\top (I-S)W\right) + \mu \|W - E\|_F^2 \tag{8}$$

and the solution can be obtained via iteration:

$$E^{(t+1)} = (1-\alpha)E + \alpha S E^{(t)} \tag{9}$$

where $\alpha = 1/(1+\mu)$ and $E^{(0)} = E$, which converges rapidly to $\hat{E}$.

After the correct iteration mentioned above, C&S uses a smooth iteration, encouraging smoothness over the distribution over labels by another label propagation. C&S iterate $G^{(t+1)} = (1-\alpha)G + \alpha S G^{(t)}$ with $G^{(0)} = G$ until converge to give the final prediction $\hat{Y}$, where $G$ is out best guess of the model, and $G_{L_t} = Y_{l_t}$, $G_{L_v,U} = Z^{(r)}_{L_v,U}$. $Y$ is the label matrix, and Z is the base prediction.

Note that C&S can be regarded as a post-processing method to improve the prediction given by any base predictor. We first reproduced the experiment in [11], which used MLP as the base predictor. And we found that C&S gives more than 20% improvement to MLP. In addition, if we use GCN as the base predictor, GCN+C&S can outperform with a improvement of approximate 1%, with 5 iterations for correcting and 5 iterations for smoothing. Our results are shown in Table 7.

### 4.3 Deep GCN

According to the over-smoothing effect, deepen the GNN endlessly may not be a wise choice. In the meanwhile, six degrees of separation also points out that we

| Method | C&S | SAM | Accuracy(%) |
|---|---|---|---|
| MLP | | | 51.8 |
| MLP | √ | | 72.9 |
| GCN | | | 81.80 |
| GCN | √ | | 82.1 |
| GCN | √ | √ | 83.0 |

**Table 7.** Node classification with shallow GCNs

| Method | DropEdge | JK | Accuracy(%) |
|---|---|---|---|
| GCN2 | | | 81.1 |
| GCN2 | √ | | 82.4 |
| GCN4 | | | 62.5 |
| GCN4 | √ | | 69.5 |
| GCN4 | √ | √ | 82.9 |
| GCN8 | | | 56.4 |
| GCN8 | √ | | 65.2 |
| GCN8 | √ | √ | 82.8 |

**Table 8.** Node classification with deep GCNs

do not need GNN with so many layers.However, the great success of deep learning heavily depends on the application of deeper neural networks. Therefore, proposing new methods to handle the paradox may be the key to improve GNNs. Recent works such as DropEdge[12], Jump Knowledge Network (JKNet) [13] make efforts to solve this problem.

DropEdge[12] drops some edges randomly while training. And it is proved that DropEdge can reduce the issue of over-fitting and over-smoothing by regarding GNN as a dynamic system as in [14], significantly improving the performance of deep GCN.

JKNet[13] uses a jumping knowledge operation, which is similar to the dense connection in DenseNet[15] In the last layer of JKNet, each node will combine the output in each layer. And there are 3 methods proposed by JKNet to merge these nodes: concatenation, max-pooling, and LSTM-attention. In our experiment, we simply use the concatenation for GCN recommended by [13].

In our experiment, we found that the use of Jump Knowledge and DropEdge can supplement each other. To avoid over-fitting effect of deep GCNs ,w e add the Dropout layer between GCN layers. Interestingly, using Jump Knowledge and DropEdge can make the Dropout rate up to 0.9, which is impossible for vanilla GCN.

Table 8 shows the results of deep GCNs. We did our experiments with GCN of 2,4,8 layers (GCN2,GCN4, GCN8). From the table, we can see the over-smoothing effect of vanilla GCN, and the improvement brought by JumpKnowledge and DropEdge.Adding all the techniques, deep GCNs (GCN4, GCN8) overcome over-smoothing and outperform shallow GCN(GCN2).

### 4.4 Go deeper with GCNII

Graph convolutional network via initial residual and identity mapping (GCNII) [7]is an extension of the vanilla GCN model with two simple yet effective techniques: Initial residual and Identity mapping. And these two techniques effectively relieve the problem of over-smoothing and improve the performance of GCNII consistently as we increase its network depth. With GCNII, we can go far deeper. The network in GCNII has a depth of 64, which is 8 times of GCN8 we implemented above.

With deeper network structure, the classification accuracy of GCNII can reach 85.5%. In addition, with SAM as the optimizer instead of SGD, GCNII can reach an accuracy of 86.5%, verifying the effectiveness of SAM again.In our experiment, we found that despite the upper bound of GCNII, the accuracy of GCNII fluctuates between 83% and 86%. Therefore, future work of GCNII may focus on enhancing the stability such a deep GNN as GCNII.

## 5. Link prediction

Link prediction can be regarded as a binary classification problem, which given arbitrary two nodes, and judge if there is an edge between these two nodes.

### 5.1 Variational Graph Auto-Encoder

Variational graph auto-encoder(VGAE)[16] is a framework for unsupervised learning on graph structure based on the Variational auto-encoder (VAE), which makes use of latent variables and is capable of learning interpretable latent representations for graphs.

The architecture of VGAE consists of an inference model and a generative model. In our experiment, we

use a simple inference model parameterized by a two-layer GCN and a generative prediction given by inner product between latent variables. When training, VGAE optimize the Variational lower bound:

$$\mathscr{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}[\log p(\mathbf{A}|\mathbf{Z})] - \mathrm{KL}[q(\mathbf{Z}|\mathbf{X},\mathbf{A}) \parallel p(\mathbf{Z})], \quad (10)$$

where $\mathrm{KL}[q(\cdot) \parallel p(\cdot)]$ is the Kullback-Leibler divergence between $q(\cdot)$ and $p(\cdot)$.

### 5.2 Graph Auto-Encoder

Graph auto-encoder (GAE) is a non-probabilistic variant of the VGAE model. The loss function of GAE is defined by the reconstruction error of the graph data:

$$\mathscr{L} = \mathbb{E}_{q(\mathbf{Z}|(\mathbf{X},\mathbf{A}))}[\log p(\hat{\mathbf{A}}|\mathbf{Z})] \quad (11)$$

which is equal to the loss function of VGAE removing the KL term.

## 6. Improvement for link prediction

GAEs typically focus on preserving the topological structure or minimizing the reconstruction errors of graph data, but they have mostly ignored the data distribution of the latent codes from the graphs, which often results in inferior embedding in real-world graph data.

VGAEs attach more importance to the distribution of latent codes, however, in our experiments, we can not find the advantage of VGAEs against GAEs. We believe the problem may lie in the KL-vanishing problem, or posterior collapse[17].

To tackle this problem, Sphere-VGAE(S-VGAE) [18, 19] and Adversarial Regulated VGAE(ARVGAE)[20] are tried. We verified that both of them can significantly improve the performance of vanilla VGAE.

### 6.1 Sphere-VGAE (S-VGAE)

Since the posterior distributions are defined as a normal distribution, former VGAE has the issue of KL-vanishing problem. Sphere-VGAE (S-VGAE) [18, 19] address this issue by altering the normal distribution by von Mises-Fisher (vMF) distribution .

And since in S-VGAE, KL divergence only depends on fixed hyperparameter $\kappa$, and it increases monotonically with $\kappa$, as does concentration measured by cosine similarity.Therefore, by controlling the hyperparameter $\kappa$, S-VGAE makes the KL divergence has a positive lower bound. In our implement, we simply add a scalar 1 on the $\kappa$ encoded by the GCN encoder.

### 6.2 Adversarial Regulated VGAE

Adversarial Regulated VGAE(ARVGE)[20] is an adversarial graph embedding framework for graph data that not only make use of the distribution of nodes as in VGAE, but also introduce Generative Adversarial Networks (GANs) into GNNs.The workflow of ARVGE consists of two modules: the graph auto-encoder which is the same as VGAE and the adversarial network which serves as the adversarial regularization.

The author of ARVGE gave explanation of the success of ARVGE from the perspective of regularization. However, we regard the adversarial network as also a solution of the KL-vanishing problem. Even the KL divergence collapse to zero, the discriminator will still force the latent codes to match a prior distribution by discriminating whether the current latent code $\mathbf{z}_i \in \mathbf{Z}$ comes from the encoder or from the prior distribution.

The equation for training the encoder model with Discriminator $\mathscr{D}(\mathbf{Z})$ can be written as follows:

$$\min_{\mathscr{G}} \max_{\mathscr{D}} \mathbb{E}_{\mathbf{z}\sim p_z}[\log \mathscr{D}(\mathbf{Z})] + \mathbb{E}_{\mathbf{x}\sim p(x)}[\log(1 - \mathscr{D}(\mathscr{G}(\mathbf{X},\mathbf{A})))] \quad (12)$$

where $\mathscr{G}(\mathbf{X},\mathbf{A})$ and $\mathscr{D}(\mathbf{Z})$ indicate the generator and discriminator.

### 6.3 Results

The results of these methods are shown in Table 9. We use Adam optimizer to train our link prediction models for 600 epochs. The reason we do not use SAM is because the negative sampling in link prediction makes the deduction in SAM invalid.In VGAE and GAE we use a learning rate of 0.01, while we found the learning rate of 0.02 is better when training S-VGAE. As for ARVGE, we train the encoder with the learning rate of 0.05 and the discriminator of 0.01.

Note that by introducing Sphere-VGAE and ARVGE, our models for link prediction achieve the state-of-the-art level.

| Method | AUC(%) | AP(%) |
|---|---|---|
| GAE | 91.14 | 91.28 |
| VGAE | 90.56 | 91.56 |
| ARVGE | 92.50 | 92.90 |
| S-VGAE | 92.88 | 93.1 |

**Table 9.** Link prediction on Cora

## 7. Conclusion

**1.**Use Louvain algorithm for high modularity and Infomap for finest community structure.

**2.**Clustering coefficient increases when real networks evolve.

**3.**The phase transition point is reached when average degree of random graph reached 1.

**4.**Use SAM for better performance in node classification.

**5.**Deeper GCNs is better than shallow GCNs after tackling the over-smoothing.

**6.**Tackle the KL-vanishing problem for performance improvement in VGAE.

## References


[1] Martin Rosvall, Daniel Axelsson, and Carl T Bergstrom. The map equation. *The European Physical Journal Special Topics*, 178(1):13–23, 2009.

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

[3] Xiaojin Zhu and Zoubin Ghahramani . Learning from labeled and unlabeled data with label propagation. 2002.

[4] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[5] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems*, 27:2177–2185, 2014.

[6] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[7] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020.

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

[9] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417. PMLR, 2015.

[10] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *arXiv preprint arXiv:2010.01412*, 2020.

[11] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R Benson. Combining label propagation and simple models out-performs graph neural networks. *arXiv preprint arXiv:2010.13993*, 2020.

[12] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. *arXiv preprint arXiv:1907.10903*, 2019.

[13] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pages 5453–5462. PMLR, 2018.

[14] Kenta Oono and Taiji Suzuki. On asymptotic behaviors of graph cnns from dynamical systems perspective. 2019.

[15] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected con-

volutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

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

[17] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

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

[19] Jiacheng Xu and Greg Durrett. Spherical latent spaces for stable variational autoencoders. *arXiv preprint arXiv:1808.10805*, 2018.

[20] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407*, 2018.