GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders

January 21, 2022

Outline

1 Introduction
- Architecture

2 Sample simulation

Variational Graph Auto-Encoders

Encoder: Two-layer GCN:

$$GCN(X, A) = \tilde{A} Relu(A\tilde{X}W_0)W_1$$

$\boldsymbol{\mu} = \mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A})$ is the matrix of mean vectors $\boldsymbol{\mu}_i$; similarly $\log \boldsymbol{\sigma} = \mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$

$\mathrm{GCN}_{\boldsymbol{\mu}}(\mathbf{X}, \mathbf{A})$ and $\mathrm{GCN}_{\boldsymbol{\sigma}}(\mathbf{X}, \mathbf{A})$ share first-layer parameters $\mathbf{W}_0$

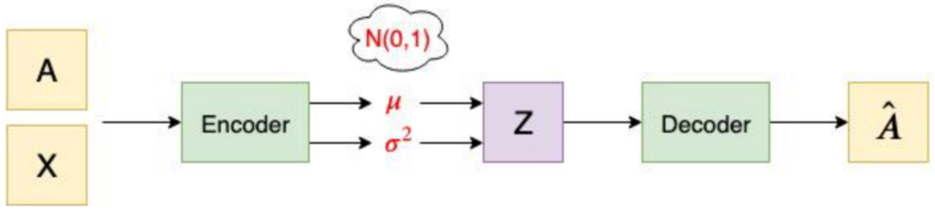
Inference model:

$$q(\mathbf{Z} \,|\, \mathbf{X}, \mathbf{A}) = \prod_{i=1}^{N} q(\mathbf{z}_i \,|\, \mathbf{X}, \mathbf{A}), \quad \text{with} \quad q(\mathbf{z}_i \,|\, \mathbf{X}, \mathbf{A}) = \mathcal{N}(\mathbf{z}_i \,|\, \boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i^2))$$

```
tensor([[ 1.3924e+00,  2.3742e-03, -2.0968e-03,  5.6259e-04,  2.5119e-02],
        [ 1.4365e+00,  1.8989e-02, -2.2564e-02,  2.3635e-04,  3.4869e-02],
        [ 1.3924e+00,  2.3742e-03, -2.0968e-03,  5.6259e-04,  2.5119e-02],
        [ 1.5349e+00,  1.7675e-02,  1.1449e-04,  1.0096e-02,  3.4326e-02],
        [ 1.4815e+00,  3.5499e-03, -2.1504e-02,  1.0316e-02,  3.1015e-02],
        [-7.1078e-01, -6.8555e-03, -4.6863e-02, -2.7428e-02,  3.4037e-02],
        [-7.0468e-01, -7.9992e-04, -9.8651e-03,  1.2867e-02, -7.4852e-03],
        [-6.9762e-01,  9.0425e-03, -2.3550e-02, -9.4596e-04,  5.3763e-03],
        [-6.8976e-01,  2.0575e-02,  1.2874e-03,  2.2452e-02, -3.0748e-02],
        [-7.8341e-01,  1.9478e-02, -1.7059e-02, -1.3157e-03,  9.0686e-03]],
       grad_fn=<AddBackward0>),
```

Here, μ and Z are $N \times F$ matrices, in which F is the latent space dimension.

Variational Graph Auto-Encoders



Generative model:

$$p\left(\mathbf{A} \mid \mathbf{Z}\right) = \prod_{i=1}^{N} \prod_{j=1}^{N} p\left(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j\right) \;, \quad \text{with} \quad p\left(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j\right) = \sigma(\mathbf{z}_i^\top \mathbf{z}_j$$

where A_{ij} are the elements of $\mathbf{A}$ and $\sigma(\cdot)$ is the logistic sigmoid function.

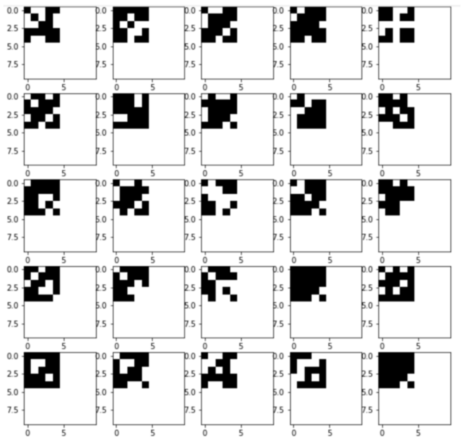
We optimize the variational lower bound $\mathcal{L}$ w.r.t. the variational parameters $\mathbf{W}_i$:

$$\mathcal{L} = \mathbb{E}_{q(\mathbf{Z}|\mathbf{X},\mathbf{A})}\left[\log p\left(\mathbf{A} \mid \mathbf{Z}\right)\right] - \mathrm{KL}\left[q(\mathbf{Z} \mid \mathbf{X}, \mathbf{A}) \,||\, p(\mathbf{Z})\right] ,$$

in which

$$\text{Gaussian prior } p(\mathbf{Z}) = \prod_i p(\mathbf{z_i}) = \prod_i \mathcal{N}(\mathbf{z}_i \mid 0, \mathbf{I})$$

Variational Graph Auto-Encoders



Input: $N \times N$ adjacency matrix A and N × D feature matrix X, where N is the number of nodes, D is the dimension of features.

For the upper left submatrix of A, let

$$A(i,j) \sim Bernoulli(p = 0.75);\ A(j,i) = A(i,j), \qquad for\ j \leq \frac{N}{2}, i < j.$$

$$A(\mathrm{i}, j) = 0, \text{elsewhere}$$

Let X be the one-hot matrix with respect to the node index.

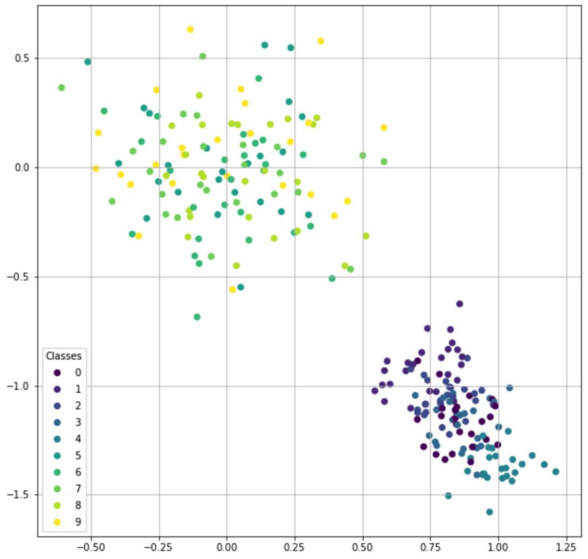
Hyperparameters

- Input: 30 graphs with 10 nodes
- Optimizers: Adams optimizer with learning rate 0.01
- Epoches: 10000 epoches
- Latent space: 2 or 10 out-channels

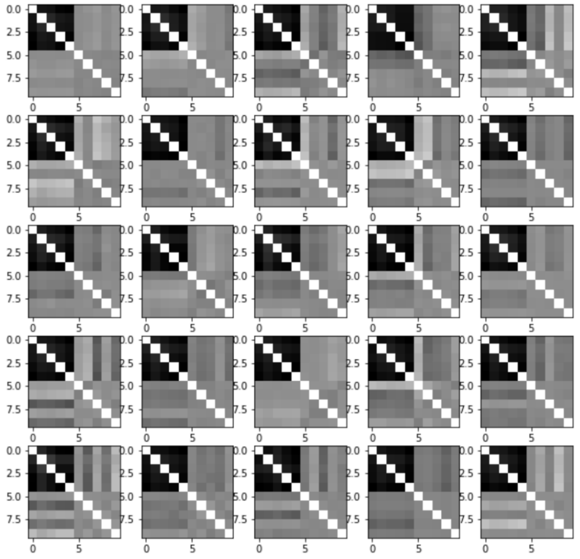
Sampling process

- Flatten the n $1 \times \times F$ μ_i vectors $\mu_i^{'}$
- Fit a multivariate Gaussian distributions $N_{\mu'}$ from $\mu_i^{'}$.
- Sample μ'_{sample} from the Gaussian distribution.
- Restore μ'_{sample} to n $1 \times \times F$ matrix μ_{sample}.
- Let $z = \mu_{sample}$.
- Decode the latent variable z into a sampling adjacency matrix using the inner product decoder:
  $p(\mathbf{A} \mid \mathbf{Z}) = \prod_{i=1}^{N} \prod_{j=1}^{N} p\left(A_{ij} \mid \mathbf{z}_i, \mathbf{z}_j\right),$ with
  $p\left(A_{ij} = 1 \mid \mathbf{z}_i, \mathbf{z}_j\right) = \sigma\left(\mathbf{z}_i^{\top} \mathbf{z}_j\right)$

Model with 2 out-channels

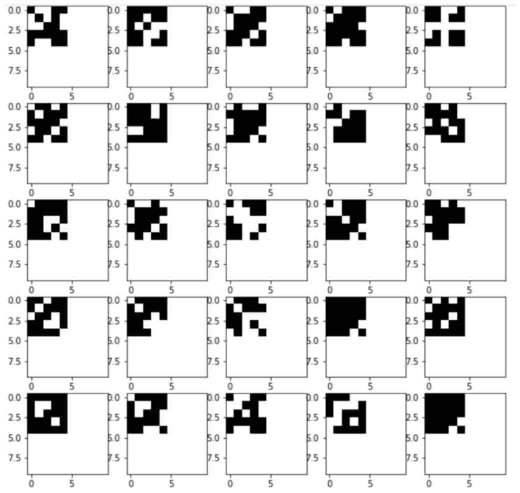


The μ for each nodes for 25 μ sampled from the Gaussian distribution. The blue and purple points are the first five nodes (connected nodes). The yellow and green points are the last five nodes (empty nodes).

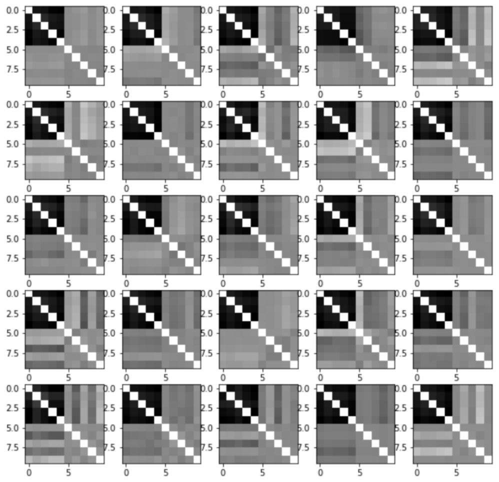


25 generated adjacency matrices.

Model with 2 out-channels

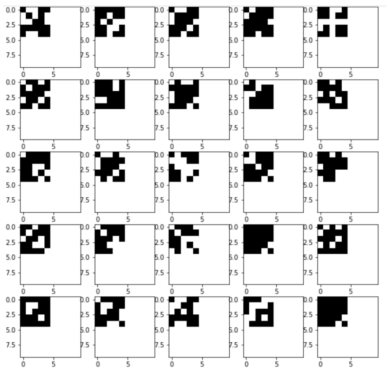


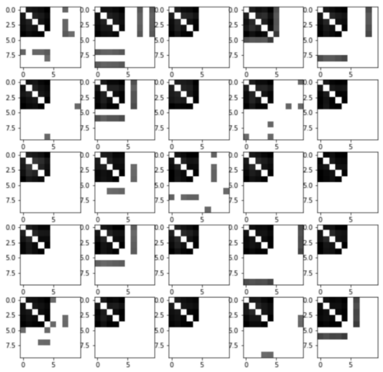
original

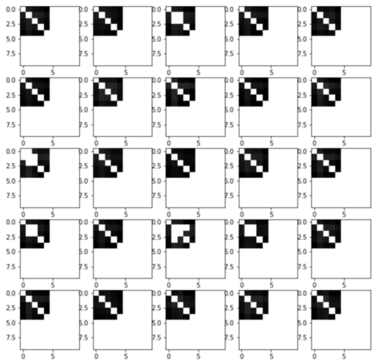


generated

Model with 2 out-channels


original

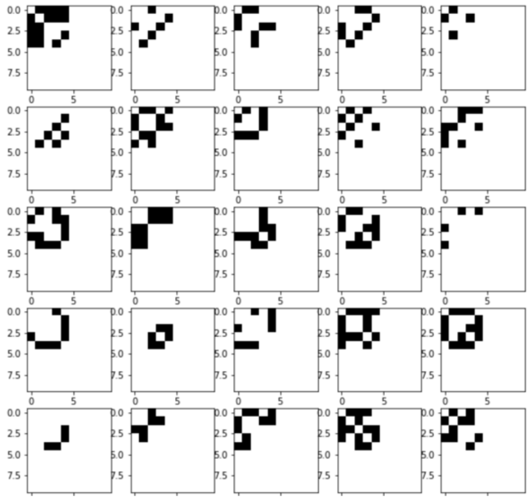

Generated with filter value=0.3

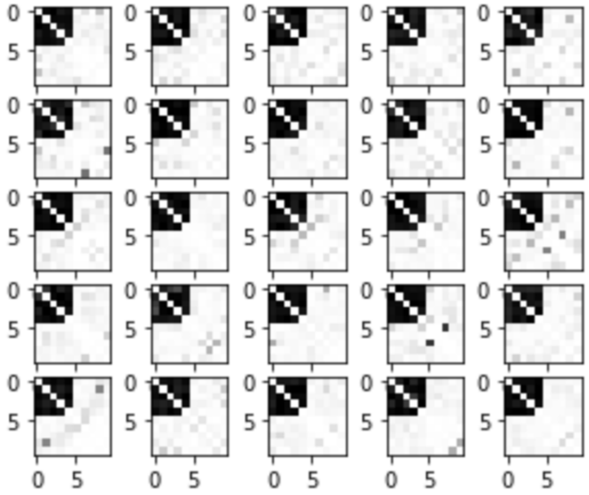

Generated with filter value=0.5

Problem: Inner product decoder: $p\left(A_{ij}=1 \mid \mathbf{z}_i, \mathbf{z}_j\right)=\sigma\left(\mathbf{z}_i^{\top} \mathbf{z}_j\right)$. It is hard for 5 nodes or more to be almost orthogonal in 2-dimension (2 out-channels) latent space. Hence, we consider to increase the out-channels to 10.

Model with 10 out-channels: Modification

- Increase the training epoch to 1,000,000 for larger parameter size.
- Since we only have 30 μ_i from the input data with 100 dimensions with high covariance. It is bad to use the covariance of the input data as Gaussian distribution's variance. Just use the $diag(N)$ instead. (Any better methods to get the distribution in high-dimension?)
- No filtering for the generated data.
- Any better methods to visualize 10-dim μ variables?


Input adjacency matrices


Generated adjacency matrices

centrality and t-test

In each graph, we first calculate the node degree $D_i = \sum_j A_{ij}$ for all nodes and then normalize them by the sum of node degrees to obtain the fraction of node degree centrality.

For each cell type, we compare the network centrality distribution between input samples and generated samples using a two-sample t-test of the degree centrality fraction.

Mean centrality in the input data: [0.20999998, 0.18333332, 0.18666664, 0.20333335, 0.2033333 , 0. , 0. , 0. , 0. , 0.].

Use $D_i = \sum_j p_{ij}$ to compute centrality. Mean centrality in the generated data: [0.39040652, 0.39277376, 0.38441248, 0.38861404, 0.38953033, 0.05740054, 0.0553917 , 0.06158545, 0.06013786, 0.05734267].

centrality and t-test

- Mean centrality in the input data: [0.20999998, 0.18333332, 0.18666664, 0.20333335, 0.2033333 , 0. , 0. , 0. , 0. , 0.].
- Use $D_i = \sum_j p_{ij}$ to compute centrality. Mean centrality in the generated data: [0.39040652, 0.39277376, 0.38441248, 0.38861404, 0.38953033, 0.05740054, 0.0553917 , 0.06158545, 0.06013786, 0.05734267].
- t-test Result: (statistic=array([1.05450925e+01, 1.39907999e+01, 1.16459299e+01, 1.05558862e+01, 1.11389892e+01, 1.24932758e+16, 1.30239788e+16, 1.21649121e+16, 1.18756043e+16, 1.29796585e+16]), pvalue=array([4.18485928e-15, 3.05209365e-20, 8.12784721e-17, 4.02340804e-15, 4.90142538e-16, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00]))