Anomaly Detection on Attributed Networks

Alex Gittens, Dylan Walker

August 2021

1 Abstract

Anomaly detection is the task of identifying a subset of elements in a set that differ from a majority of the rest of the set. In the case of attributed networks, this involves identifying the nodes in a network that have unusual behavior within the context of the entire network. What differentiates this task for network-structured data is the complex interaction between the underlying network structure and the individual node features. In this paper we introduce a novel deep learning framework for anomaly detection on attributed networks. Our outlined model is able to capture the non-linear relationships between both node features and network structure through a deep embedding process. Then our model is able to use these deep embeddings to learn a latent probability distribution that can be used to identify the nodes with data that deviates most from the expectations of the network.

2 Introduction

Attributed networks are a common representation of many real-world graphical structures such as social networks, publication citation networks, and physical road networks. Almost all real-world networks are composed of nodes with many features and details association with them. In most cases the network structure has an intricate co-dependence on these node features. For example, in a social network details such as age, hometown, or university are typically shared abundantly with the friends of that user. The goal of anomaly detection is to determine nodes or regions in a graph that behave abnormally with respect to the majority of the network.

In this paper we outline a deep learning method for anomaly detection that aims to model the intricate interaction between network topology and node features. Our model is directed at the task of detecting local community outliers within the context of the entire network. That is to say that our methods seek to identify the nodes that have the most abnormal features and connections given the behavior of their surrounding community within the network. This is done by first applying a Relational Graph Convolutional Network (R-GCN) to the neighborhood of each node to learn a deep embedding of each community. We then use these neighborhood embedding to learn a probability distribution that models the expected behavior of a nodes neighborhood given its attributes. We then apply this probability distribution to measure the nodes that least fit the overall behavior of the network, which we then determine to be anomalies. In summary, our main contributions are as follows:

- We introduce a deep learning method for anomaly detection that uses deep neural networks to learn the expected behavior of a nodes local community, taking into consideration the complex interaction between network structure and node attribution.
- We experiment with our model on real-world public datasets and compare to state-of-the-art models for anomaly detection to display the validity of our proposed model.

3 Problem Statement

Definition 1: An **attributed network** $\mathcal{G} = (V, E, X)$ is an undirected graph with vertex set V, edge set E, and node feature matrix X. It follows that the number of vertices n is given by $n = |V|$ and the number of edges m is given by $m = |E|$. Also, we denote the node feature dimension D to be the number of features for each node in $\mathcal{G}$. For the sake of this paper we will use the notation $\mathcal{G} = (A, X)$ where X is the node feature matrix and A is the adjacency matrix of $\mathcal{G}$.

Problem: Given an attributed network $\mathcal{G} = (V, E.X)$, our goal is to rank the vertices of $\mathcal{G}$ by how likely they are to be anomalous within the overall context of the network $\mathcal{G}$. More succinctly, the goal of our model is to learn a threshold value λ and a scoring function $f : v_i \to y_i \in \mathbb{R}$ for each vertex $v_i \in V$ such that we can classify each node as anomalous or normal. Let y_i denote the output anomaly label for node i under our model where $y_i = 1$ for the anomaly class and $y_i = 0$ for the normal class. Our goal is to learn f and λ such that:

$$y_i = \begin{cases} 1 & f(v_i) \geq \lambda \\ 0 & \text{otherwise} \end{cases}$$

4 Model Overview

Consider a large undirected attributed network $\mathcal{G}$. The goal of our model is to detect anomalous nodes in G. Our model accomplishes this by considering the local neighborhoods surrounding a given node. If a node i is an outlier within it's neighborhood given the context of the overarching network G, then i should be detected by the model as an anomaly. Unlike most other models, our model will learn the distribution of node features and connections in G through sampled training neighborhoods in G. The sampling process used is described in more detail in section 4.3.

Let $G = (A, X)$ be the neighborhood of node i in $\mathcal{G}$. Our model aims to learn the following probability of node i having its features X_i and connections A_i given the features and connections of its local neighborhood. Thus our model aims to learn the following probability distribution:

$$p(A_i, X_i | A, X) \tag{1}$$

Our model learns this process through a two-step deep learning process. Given a sampled neighborhood G, our model first applies Relational Graph Convolutional Network (R-GCN) to learn a deep embedding for each node in the neighborhood. After forming these node embeddings, our model passes them to a trainable neural network that learns the probability distribution in equation 1. Once the parameters of this probability distribution are learned, we denote the anomaly score of a node i to be the probability value of its sampled neighborhood under our model. Using artificial anomalies, we are then able to determine the probability threshold that best divides the anomalous samples from the normal samples. The result is an end-to-end model representing a scoring function that assigns an anomaly score to each node in a large attributed network $\mathcal{G}$.

4.1 Algorithm Outline

As mentioned before, our model trains on egonet neighborhoods sampled from a larger network. Let $G = (A, X)$ be such a sample centered around node i with a total of n nodes and D feature dimensions. Let $A \in \{0, 1\}^{n \times n}$ be the adjacency matrix of G in which $A_{u,v} = 1$ if there exists an edge (u, v) in the edge set of G and $A_{u,v} = 0$ otherwise. This can naturally be extended to weighted graphs but in our testing we only consider unweighted graphs. Let $X \in \{0, 1\}^{n \times D}$ be a node feature matrix in which the jth row of X represents the feature vector for node j in G.

In the general case both the adjacency matrix A and the feature matrix X will be discrete matrices that represent quantized data. For data that has been quantized from a continuous space, we find it reasonable to apply uniform dequantization [HCT21] to map our data $G = (A, X)$ into a continuous space $z = (z^A, z^X)$. Let $0 \leq \alpha \leq 1$ be the dequantization coefficient. We then apply dequantization as follows:

$$z_i^X = X_i + u_i, \quad u_i \sim U[0, \alpha)^D \tag{2}$$

When the features X are already continuous, there is no need to add noise to them in the second equation above. Also if the features X are discrete but represent data that is unquantized, for example a binary matrix representing a bag-of-words feature space, then dequantization should not be applied. If our input data fits either of these descriptions then we set $\alpha = 0$. In most cases the adjacency matrix should not have dequantization applied, only in the case where the adjacency matrix values represent quantized edge weights over a continuous domain. If the edges do represent quantized data then we can apply a similar uniform dequantization to A:

$$z_{ij}^A = A_{ij} + u_{ij}, \quad u_{ij} \sim U[0, \alpha) \tag{3}$$

Next, we will take our input matrices z^A and z^X that represent the neighborhood G and pass them through a Relational Graph Convolutional Network (R-GCN) [Sch+17]. Let L be the number of layers in the R-GCN. Also let $A^* = z^A + I$ and D be the diagonal degree matrix of A^*. We can define the R-GCN as follows where H^l denotes the node embeddings at the lth layer of the R-GCN:

$$H_0 = z^X \tag{4}$$

$$H^l = \sigma(D^{-\frac{1}{2}} A^* D^{-\frac{1}{2}} H^{l-1} W^l) \tag{5}$$

Where W^l is a learnable weight matrix for layer l and σ is a nonlinear activation function such as the sigmoid function. After a total of L layers we then perform a linear combination of the node embeddings at each layer with learnable weight coefficients to get the final embedding H. If we let k denote the embedding dimension then it follows that $H \in \mathbb{R}^{n \times k}$. Thus we can also use pooling to get an embedding of the entire graph $h \in \mathbb{R}^k$. The terms w_j in equation 7 represent the learnable weights that weigh each layer of the R-GCN to form the final embedding H.

$$H^1, H^2, \ldots, H^L = \text{R-GCN}(G) \tag{6}$$

$$H = \sum_{l=1}^{L} w_l H^l \tag{7}$$

$$h = \sum_{i=1}^{n} (H_i) \tag{8}$$

Given these nodewise and aggregated embeddings, we can then use trainable neural networks to learn the distributions of our data. Depending on the structure of our input data, we apply different underlying distributions to learn. In the case where the input data is continuous or has been dequantized, we model the data with a Gaussian distribution. For a neighborhood centered at node i, we define the distributions as follows [Shi+20]:

$$p(z_i^X | G) = \mathcal{N}(\mu_i^X, \Sigma_X), \text{ where } \mu_i^X = g_{\mu^X}(G) \text{ and } \Sigma_X = L_X L_X^T \text{ for } L_X = g_{L_X}(G) \tag{9}$$

$$p(z_{ij}^A | G) = \mathcal{N}(\mu_{ij}^A, \alpha_{ij}^A), \text{ where } \mu_{ij}^A = g_{\mu^A}(G) \text{ and } \alpha_{ij}^A = g_{\alpha^A}(G) \tag{10}$$

where the matrix L_X is a learned lower triangle matrix with non-negative entries. The functions g are learned and are outlined below where the functions m are trainable neural networks:

Node-MLPs:

$$g_{\mu^X} : \mathbb{R}^k \to \mathbb{R}^D = m_{\mu^X}(h) \tag{11}$$

$$g_{L_X} : \mathbb{R}^k \to \mathbb{R}^{\binom{D}{2}+D} = m_{L_X}(h) \tag{12}$$

Edge-MLP:

$$g_{\mu^A} : \mathbb{R}^{3k} \to \mathbb{R} = m_{\mu^A}(h, H_i, H_j) \tag{13}$$

$$g_{\alpha^A} : \mathbb{R}^{3k} \to \mathbb{R} = m_{\alpha^A}(h, H_i, H_j) \tag{14}$$

The other case to be handled is where both the adjacency matrix A and node features X represent discrete data, such as binary feature vectors and unweighted network edges. In this case the Gaussian distribution is far less fitting so we instead apply a Bernoulli distribution. This is defined as follows where $\mathcal{B}(\mu)$ represents a Bernoulli distribution with mean μ:

$$p(X_i|G) = \mathcal{B}(X_i|\mu_i^X), \text{ where } \mu_i^X = g_{\mu^X}(G) \tag{15}$$

$$p(A_{ij}|G) = \mathcal{B}(A_{ij}|\mu_{ij}^A), \text{ where } \mu_{ij}^A = g_{\mu^A}(G) \tag{16}$$

We can then define the learnable networks g in the same manner as our previous definition where the functions m are trainable neural networks:

Node-MLP:

$$g_{\mu^X} : \mathbb{R}^k \to \mathbb{R}^D = m_{\mu^X}(h) \tag{17}$$

Edge-MLP:

$$g_{\mu^A} : \mathbb{R}^{3k} \to \mathbb{R} = m_{\mu^A}(h, H_i, H_j) \tag{18}$$

Altogether, these embeddings and neural networks can be used to learn the probability of a node fitting its neighborhood given its connections and features, all within the context of the overarching network. It then follows that the goal of the training process is to learn the model parameters that maximize this probability over the training data. These probability values can then be used to score unseen test samples in order to identify anomalous nodes. Under this construction we define our loss function as follows:

$$\mathcal{L}^X = -log(\prod_{d=1}^{D} p(z_{id}^X)) \tag{19}$$

$$\mathcal{L}_j^A = -log(p(z_{ij}^A)) \tag{20}$$

$$\mathcal{L}(G) = \mathcal{L}^X + \sum_{j=1}^{N} \mathcal{L}_j^A \tag{21}$$

Where z_{id}^X represents the dth feature of the central node i and z_{ij}^A represents the adjacency value between the central node i and another node j.

4.2 Anomaly Detection

In order to determine anomalies, we use our loss function to measure the likelihood of node i fitting its neighborhood G. More succinctly, for a node i with neighborhood sample G we define the scoring function f of our model as follows:

$$f(i) = \mathcal{L}(G) = p(A_i, X_i | A, X) \tag{22}$$

Under this construction, we can simply use the scoring function f to rank the nodes by order of abnormality. Nodes with a lower score are less likely to fit their neighborhoods and are thus more likely to be anomalies.

4.3 Sampling Process

For the training of our model, we used a breadth-first-search approach to generate neighborhood samples with a consistent node ordering. The generation process is primarily controlled by two parameters, the maximum neighborhood size N and the search depth d. First we sample a random node u from the network G. We then perform BFS from u and add each node in the search to the neighborhood while recording the resulting ordering. BFS is run until either the neighborhood reaches the maximum size N or the search depth limit d is reached. Finally we take the sampled neighborhood H to be the subgraph of G containing all vertices in the neighborhood and all edges with both endpoints in the neighborhood. This process provides us with neighborhoods that have a consistent ordering. We then use the same BFS ordering that was found as the ordering to form the neighborhood adjacency matrix A_H and feature matrix X_H.

The AnomalyDAE and DeepAD models are deep autoencoder models that take as input the entire network $G = (A, X)$ and produce an anomaly score for each node in the graph based on the reconstruction error. Thus these models do not require the additional parameters for generating the sample neighborhoods.

5 Datasets / Results

For testing our model, we primarily relied on two attributed citation networks, Citeseer and Cora. In both networks nodes represent publications and edges represent references between a given pair of publications. The node attributes are binary vectors representing the presence of words in a predefined vocabulary. The data in both of these datasets is discrete and binary, and thus we apply no dequantization and use the Bernoulli distribution for the neural networks. Both datasets are summarized in the table below:

Table 1

Network	Vertices	Edges	Attributes
Citeseer	3312	4732	3703
Cora	2708	5429	1433

Summary of testing datasets

5.1 Baseline Methods

To measure the relative performance of our model we use three state-of-the-art models for anomaly detection as baseline comparision methods. DeepAD [ZML20] is a deep autoencoder model that learns a bottleneck reconstruction process and in turn uses nodal reconstruction errors to detect anomalies. AnomalyDAE [FZL20] is a similar dual autoencoder model that also uses reconstruction errors to measure abnormalities. SpecAE [Li+19] applies a spectral convolution and deconvolution process that projects the network into a tailored space for anomaly detection.

5.2 Anomaly Formation

In order to test these models we implement a common method of adding random cliques for generating detectable anomalies. Let G be an attributed undirected network such as Citeseer or Cora. We can then randomly select a subset of nodes $H \subseteq V(G)$ such that $|H| = n$. Then to generate the anomalies we add edges to G such that a clique is formed between all of the vertices in H. We then mark every vertex in H as an anomalous neighborhood. This process is repeated m times in order to generate a total of mn anomalous nodes. This method is used to generate an anomaly set upon which we tested our trained model to generate the results in section 5.3. In the appendix section 8.2 we experiment with other types of anomalies and discuss their validity and performance results in the context of our datasets.

Our baseline methods are autoencoders that train and run on the entire network and thus we follow a similar procedure for training our model. We first introduce the mn anomalous nodes and sample the neighborhoods of each node in the network, keeping record of samples are generated from anomalous nodes. We then train our model on these samples. Lastly we run each sample through the model and use the anomaly labels to measure the performance of the model at detecting the anomalies. Since our model learns the probability of a node fitting it's neighborhood, we aim to learn a probability threshold that divides the anomalous neighborhoods from the normal neighborhoods.

5.3 Test Results

Tables 2-3 were generated using training neighborhoods with a search depth of 3 hops and a maximum neighborhood size of 30 nodes. For the sake of time complexity in testing this model, we take only the first 100 features of each node. The anomaly set is composed of nine 12-node cliques for a total of 108 anomalous nodes.

Table 2

Model	AUC	F1 Score	Precision	Recall
Our model	0.994	0.809	0.679	1.000
AnomalyDAE	0.984	0.631	0.463	0.991
% Difference	1.02%	28.21%	46.65%	0.91%

Hyperstatistics on Citeseer Dataset versus AnomalyDAE

Table 3

Model	AUC	F1 Score	Precision	Recall
Our model	0.997	0.893	0.806	1.000
AnomalyDAE	0.993	0.895	0.817	0.991
% Difference	0.40%	-0.22%	-1.35%	0.91%

Hyperstatistics on Cora Dataset versus AnomalyDAE

At the moment we do not have an implementation of DeepAD or SpecAE. I reached out to the authors of both papers but did not receive a response. However DeepAD saw consistent performance of an AUC value on the Citeseer dataset of roughly 0.90, so our models performance of 0.994 suggests that we likely outperform DeepAD. SpecAE uses a different formulation for generating anomalies that is similar to the feature-swapping technique we analyze in section 8.3.

6 Improvements / Future Directions

Going forward, there are a few potential avenues for which our model can be improved. Our method currently relies on an underlying distribution that models the latent data. In our implementation we only cover the application of Gaussian and Bernoulli distributions. Depending on the format of a given network dataset, it may be more fitting to model the data with a different distribution. One suggestion would be the heavier-tailed t-distribution. When working with continuous data, our model aims to learn a covariance matrix by learning a lower-triangular decomposition matrix. There is an inherent trade-off between complexity and expressiveness in this process. In order to improve the expressiveness of the model, one could instead learn the entire covariance matrix instead of a decomposition. This would double the output parameters for the node attribue neural network and would also introduce additional complexity in the calculation of the covariance matrix determinant and inverse needed to evalaute the distribution. In just learning a lower triangular decomposition matrix we saw runtime limitations with our model. For this reason we suggest replacing the learned matrix L with a new lower-triangular L' that is low-rank. This would allow for more efficient training while still being able to model node feature relationships.

In order to get a more complete grasp of the capabilities of our model we must conduct more cross-validation with other state-of-the-art models. This involves implementing and testing DeepAD and SpecAE on consistent datasets with those that were used to generate the metrics for our model. While our numbers may suggest our model can outperform these models we must be able to show that our model can outperform them on more robust types of anomalies. Also, our testing abilities were hindered by runtime limitations, leading us to use only a partial node feature set. Alleviating this limitation would allow us to gather a more accurate measure of the models performance on the entire networks.

7 Conclusion

In this paper we propose a novel algorithm for anomaly detection on attributed networks. Our model captures the complex interaction between network structure and node attributes. In order to identify local anomalies our model learns the expected relationships between a node and its local neighborhood. We create a deep latent representation of a neighborhood via graph convolutional networks (GCN) and then apply neural networks to learn a probability distribution for a node being consistent with its neighborhood. By training these two systems simultaneously, we are able to apply the learned distribution to determine the likelihood of each node fitting the network. These probability values are then used to score each node and determine the nodes that are most anomalous within the network. Our results display the capabilities of our methods for identifying artificial local anomalies.

8 Appendix

8.1 Training Algorithm:

Parameters:

M - Batch size

α - dequantization coefficient

η - learning rate

β_1, β_2 - Adam Optimizer parameters

Note: Prod(.) refers to the product of elements across the dimensions of a tensor

for $m = 1, \ldots, M$:

sample $G = (A, X)$ around node i

$z_i^X = X_i + u, \quad u \sim U[0, \alpha)^D$

$\mu_i^X = g_{\mu^X}(G), \quad L_X = g_{L_X}(G)$

$\Sigma_X = L_X L_X^T$

$p_1 = \mathcal{N}(z_i^X | \mu_i^X, \Sigma_X)$

$\mathcal{L}^X = -log(Prod(p_1))$

$z_{ij}^A = A_{ij} + u, \quad u \sim U[0, \alpha)$

for $j = 1, \ldots, N$:

$\mu_{ij}^A = g_{\mu^A}(G), \quad \alpha_{ij}^A = g_{\alpha^A}(G)$

$p_2 = \mathcal{N}(z_{ij}^A | \mu_{ij}^A, \alpha_{ij}^A)$

$\mathcal{L}_j^A = -log(p_2)$

end

$\mathcal{L}_m = \mathcal{L}^X + \sum_{j=1}^N \mathcal{L}_j^A$

$\theta \longleftarrow ADAM(\frac{1}{M} \sum_m \mathcal{L}_m, \theta, \eta, \beta_1, \beta_2)$

end

8.2 Hyperparameter Sensitivity

Tables 5-8 below display how our model performs under varying settings for the neighborhood and clique generation methods. These tests differ from the tests in section 5.3 as they are not run on the entire network. For these test cases we sample 500 training neighborhoods from the graph and a total of 350 testing neighborhoods (100 anomalies, 250 normal) that are distinct from the training examples. We train the model with the 500 training samples and then generate the metrics in the tables below using the 350 test examples. For each table we use the following constant values for all parameters that are not the parameter being tested:

Table 4

# Parameter	Value
Max Neighborhood Size	40
BFS Search Depth	2
Clique Size	12
Node Features	50

Constant Parameter Values

The following results display the sensitivity of the model to each of these hyperparameters:

Table 5

# Features	AUC	F1 Score	Precision	Recall
20	0.992	0.971	0.943	1.000
50	1.000	0.985	0.980	0.990
100	0.994	0.990	0.980	1.000

Impact of Node Feature Vector Dimensionality on Citeseer

Table 6

Clique Size	AUC	F1 Score	Precision	Recall
10	0.989	0.980	0.962	1.000
12	1.000	0.985	0.980	0.990
15	0.999	0.990	0.980	1.000

Impact of Anomaly Clique Size on Citeseer

Table 7

Depth	AUC	F1 Score	Precision	Recall
1	0.990	0.976	0.952	1.000
2	0.990	0.943	0.893	1.000
3	0.997	0.985	0.971	1.000

Impact of Search Depth during Neighborhood Sample Generation on Citeseer

Table 8

Size	AUC	F1 Score	Precision	Recall
30	0.992	0.980	0.962	1.000
40	0.990	0.943	0.893	0.987

Impact of Maximum Neighborhood Size (#nodes) on Citeseer

8.3 Additional Testing Results

Along with the clique-based anomalies, we also experimented with more robust anomaly types. The first method we used was to replace neighborhood structures with a foreign structure. Assume we have an attributed undirected network $\mathcal{G}$. We can then sample a neighborhood $G \subseteq \mathcal{G}$ and let $k = |G|$. We then let A denote the adjacency matrix of G and X the feature matrix of G. Now consider another undirected network $\mathcal{G}'$, which need not be attributed. We can sample a neighborhood $G' \subseteq \mathcal{G}'$ with the condition that $|G'| = k = |G|$. Let A' represent the adjacency matrix of G'. It follows trivially that both A and A' are $k \times k$ adjacency matrices. Because of this, we can now swap the structure of G with the structure of G' by just replacing the adjacency matrix of G with A' whilst maintaining the original node features X. Ideally, if $\mathcal{G}$ and $\mathcal{G}'$ are non-similar networks, this would give the sample neighborhood G a connectivity pattern that is inconsistent with the underlying distribution of neighborhoods in $\mathcal{G}$. Thus we can label G as a new type of anomaly and generate a set of anomalous neighborhoods with this construction.

For implementing these structure-swap anomalies we used an additional network dataset, ego-Facebook [ML13]. Ego-Facebook is a social network where nodes represent users and edges represent friendship connections between two users.

Table 9

AUC	F1 Score	Precision	Recall
0.775	0.418	0.308	0.650

Results of Structure-Swap Anomalies for Citeseer swapped with ego-Facebook

The other type of anomaly that we generated were completely feature-based anomalies. Assume $\mathcal{G}$ is an undirected attributed network with adjacency matrix A, node feature matrix X, and node class labels C. As we would normally generate a sample, randomly select a node $v \in V(\mathcal{G})$ and generate an ego neighborhood G centered around v. Now let c_v denote the class label of v. We now repeatedly select random nodes $u \in V(\mathcal{G})$ until we find a u such that $c_u \neq c_v$. That is that u has a different class label as v. This implies that u and v should have significantly distinct feature vectors. Using this, we then replace the feature vector X_v with X_u within the sample neighborhood G. Ideally, this would result in a neighborhood in which the central node has features that are inconsistent with its neighbors, implying that v is an anomaly under these modifications. We can then repeat this process to form a set of anomalous neighborhoods for testing.The results below were generated with 2-hop neighborhoods with a maximum of 20 nodes and 500 features per node.

Table 10

AUC	F1 Score	Precision	Recall
0.513	0.213	0.132	0.530

Results of Class based Feature-Swap Anomalies on Citeseer

Our model is clearly deficient at detecting this type of anomaly. However, the AnomalyDAE model produces very similar poor results which may suggest that the node features of the Citeseer and Cora datasets are too sparse for this anomaly formation technique to be valid (Citeseer has a feature density of 0.0086). The following is an additional test ran to validate that our model is learning with the node features. Instead of swapping out the feature vector of v with a randomly selected feature vector from a node of another class, we instead fill the node vector of v with a random binary feature vector of density 0.25. This density is much greater than the average Citeseer feature density, and thus should be a glaring anomaly. The results below validate that our model is using node features to learn. Although this is the case, these results do not necessarily imply that our model is not just learning to simply reward sparse features. These results were generated with 2-hop neighborhoods with a maximum of 20 nodes and 500 features per node.

Table 11

AUC	F1 Score	Precision	Recall
0.996	0.971	0.943	1.000

Results of Swapping Feature Vectors with Random Higher Density Vectors on Citeseer

References

[FZL20] Haoyi Fan, Fengbin Zhang, and Zuoyong Li. *AnomalyDAE: Dual autoencoder for anomaly detection on attributed networks*. 2020. arXiv: 2002.03665 [cs.LG].

[HCT21] Emiel Hoogeboom, Taco Cohen, and Jakub Mikolaj Tomczak. "Learning Discrete Distributions by Dequantization". In: *Third Symposium on Advances in Approximate Bayesian Inference*. 2021. URL: https://openreview.net/forum?id=a0EpGhKt_R.

[Li+19] Yuening Li et al. *SpecAE: Spectral AutoEncoder for Anomaly Detection in Attributed Networks*. 2019. arXiv: 1908.03849 [cs.LG].

[ML13] Julian McAuley and Jure Leskovec. *Discovering Social Circles in Ego Networks*. 2013. arXiv: 1210.8182 [cs.SI].

[Sch+17] Michael Schlichtkrull et al. *Modeling Relational Data with Graph Convolutional Networks*. 2017. arXiv: 1703.06103 [stat.ML].

[Shi+20] Chence Shi et al. "GraphAF: a Flow-based Autoregressive Model for Molecular Graph Generation". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=S1esMkHYPr.

[ZML20] Dali Zhu, Yuchen Ma, and Yinlong Liu. "DeepAD: A Joint Embedding Approach for Anomaly Detection on Attributed Networks". In: *Computational Science – ICCS 2020* 12138 (2020), pp. 294–307.