Effective Decoding in Graph Auto-Encoder Using Triadic Closure

Han Shi,1 Haozheng Fan,1,2 James T. Kwok1
1Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong
2Amazon
{hshiac, jamesk}@cse.ust.hk, fanhaozh@amazon.com

Abstract
The (variational) graph auto-encoder and its variants have been popularly used for representation learning on graph-structured data. While the encoder is often a powerful graph convolutional network, the decoder reconstructs the graph structure by only considering two nodes at a time, thus ignoring possible interactions among edges. On the other hand, structured prediction, which considers the whole graph simultaneously, is computationally expensive. In this paper, we utilize the well-known triadic closure property which is exhibited in many real-world networks. We propose the triad decoder, which considers and predicts the three edges involved in a local triad together. The triad decoder can be readily used in any graph-based auto-encoder. In particular, we incorporate this to the (variational) graph auto-encoder. Experiments on link prediction, node clustering and graph generation show that the use of triads leads to more accurate prediction, clustering and better preservation of the graph characteristics.

Introduction
With the proliferation of online social networks, an enormous number of people are now connected digitally. Besides people, almost everything is also increasingly connected either physically or by all sorts of relationships, leading to the recent popularity of the internet of things and knowledge graphs. In today’s big data era, it is thus important for organization to derive the most value and insight out of this colossal amount of data entities and inter-relationships.

In general, nodes in the graph represent data entities, while edges represent all sorts of fine-grained relationships. For example, in a social network, the nodes are users that are connected by edges denoting pairwise friendships. In an author collaboration network, the edges denote co-authorship relationships. Besides social networks and knowledge graphs, data in domains such as chemistry and natural language semantics are often naturally represented as graphs.

There are a number of important tasks in graph analytics. A prominent example is link prediction (Wang, Chen, and Li 2017), which predicts whether an edge should exist between two given nodes. Since its early success at LinkedIn, link recommendation has attracted significant attention in social networks, and also in predicting associations between molecules in biology, and the discovery of relationships in a terrorist network. Other popular graph analytics tasks include the clustering of nodes (Wang et al. 2017), and automatic graph generation (Bojchevski et al. 2018). Node clustering aims to partition graph nodes into a set of clusters such that the intra-cluster nodes are much related (densely connected) with each other as compared to the inter-cluster nodes. These cluster structures occur frequently in many domains such as computer networks, sociology, and physics. Graph generation refers to the task of generating similar output graphs given an input graph. It can be used in the discovery of molecule structures and dynamic network prediction.

However, graphs are typically difficult to analyze because they are large and highly sparse. Recently, there is a surge of interest in learning better graph representations (Goyal and Ferrara 2018; Hamilton, Ying, and Leskovec 2017). Different approaches are proposed to embed the structural and attribute information in a graph to a low-dimensional vector space, such that both the neighborhood similarity and community membership are preserved (Zhang et al. 2018).

A particularly successful unsupervised representation learning model on graphs is the variational auto-encoder (VAE) (Kingma and Welling 2014), and its variants such as the variational graph auto-encoder (VGAE), graph auto-encoder (GAE) (Kipf and Welling 2016b), adversarially regularized graph auto-encoder (ARGA), and adversarially regularized variational graph auto-encoder (ARVGA) (Pan et al. 2018). All these auto-encoder models consist of an encoder, which learns the latent representation, and a decoder, which reconstructs the graph-structured data based on the learned representation. The encoder is often based on the powerful graph convolutional network (GCN) (Kipf and Welling 2016a). However, the decoder is relatively primitive, and prediction of a link is based simply on the inner product between the latent representations of the nodes involved.

As nodes in a graph are interrelated, instead of predicting each link separately, all the candidate links in the whole graph should be considered together, leading to a structured prediction problem. However, learning of structured prediction models is often NP-hard (Globerson et al. 2015). A re-
cent model along this direction is the structured prediction energy network (SPEN) (Belanger and McCallum 2016). However, design of the underlying energy function is still an open question, and computation of the underlying Hessian is computationally expensive.

On the other hand, an interesting property exhibited in many real-world networks is the so-called triadic closure, which is first proposed by Simmel (1908). This mechanism states that for any three nodes \(\{i, j, k\}\) in a graph, if there are edges between \((i, j)\) and \((i, k)\), it is likely that an edge also exists between \(j\) and \(k\). It is then popularized by Granovetter (1973), who demonstrated empirically that triadic closure can be used to characterize link connections in many domains. For example, if two people (A and B) in a friendship network have a common friend, it is likely that A and B are also friends. If two papers in a citation network cite the same paper (suggesting that they belong to the same topic), it is likely that one also cites the other. Besides, the triadic closure is also fundamental to the understanding and prediction of network evolution and community growth (Caplow 1968; Bianconi et al. 2014; Zhou et al. 2018).

In this paper, we utilize this triadic closure property as an efficient trick to break structured prediction (which considers the whole graph simultaneously but is expensive) and individual link prediction (which is simple but ignores interactions among edges). Specifically, we propose the triad decoder, which predicts the three edges involved in a triad together. The triad decoder can readily replace the vanilla decoder in any graph-based auto-encoder. In particular, we incorporate this into VGE and GAE, leading to the triad variational graph auto-encoder (TVGA) and triad graph auto-encoder (TGA). Experiments are performed on link prediction, node clustering and graph generation using a number of real-world data sets. The prediction and clustering results are more accurate, and the graphs generated preserve more characteristics of the input graph, demonstrating the usefulness of triads in graph analytics.

**Related Work: Graph Auto-Encoder**

The variational graph auto-encoder (VGE) and its non-variational variant graph auto-encoder (GAE) are introduced in (Kipf and Welling 2016b). Using the variational auto-encoder (VAE) framework (Kingma and Welling 2014), they are based on unsupervised deep learning model consisting of an encoder and a decoder.

**GCN Encoder**

Let the graph be \(G = (V, E)\), where \(V\) is a set of \(N\) nodes, and \(E\) is the set of edges. Let its adjacency matrix be \(A\). The encoder is a graph convolutional network (GCN) (Kipf and Welling 2016a), which is a deep learning model for graph-structure data. For a \(L\)-layer GCN, the layer-wise propagation rule is given by:

\[
H^{(l+1)} = f(H^{(l)}, A) = \text{relu}(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)})
\]

where \(\tilde{A} = A + I\) is the identity matrix, \(\tilde{D}\) is a diagonal matrix with \(\tilde{D}_{ii} = \sum_{j=1}^{N} A_{ij}\). \(H^{(l)}\) and \(W^{(l)}\) are the feature map and weight at the \(l\)th layer, respectively, and relu(·) is the ReLU activation function. \(H^{(0)}\) is the matrix of node feature vectors \(X\), and \(H^{(L)}\) is the (deterministic) graph embedding matrix \(Z\). Often, \(L = 2\) (Kipf and Welling 2016b), leading to the following encoder:

\[
H^{(1)} = f(X, A), \quad Z = f(H^{(1)}, A).
\]

In VGE, the embedding (encoder output) is probabilistic. Let \(z_i\) be the embedding of node \(i\). It is assumed to follow the normal distribution:

\[
q(z_i|X, A) = \mathcal{N}(\mu_i, \text{diag}(\sigma^2)),
\]

where \(\mu_i\) and \(\log \sigma\) are outputs from two GCNs that share the first-layer weights. The distribution for all the embedding vectors is then \(q(Z|X, A) = \prod_{i=1}^{N} q(z_i|X, A)\).

**Inner Product Decoder**

The decoder is often a vanilla model based on the inner product between the latent representations of two nodes. For GAE, the graph adjacency matrix \(\hat{A}\) is reconstructed from the inner product of two node embeddings as:

\[
\hat{A} = \sigma(ZZ^T),
\]

where \(\sigma\) is the sigmoid activation function. For the VGE, the decoder is also based on inner products, but is probabilistic:

\[
p(\hat{A}|Z) = \prod_{i=1}^{N} \prod_{j=1}^{N} p(\hat{A}_{ij}|z_i, z_j),
\]

where \(p(\hat{A}_{ij}|z_i, z_j) = \sigma(z_i^T z_j)\).

**Triad Decoder**

Based on triadic closure, the presence of a particular edge in a triad is dependent on whether the other two edges are present. Specifically, consider the triad \(\Delta = (i, j, k)\). Let \(I_{ij}\) be the indicator function representing whether nodes \(i\) and \(j\) are connected (i.e., \(I_{ij} = 1\) if \(i, j\) are connected; and 0 otherwise). As the presence of edges \((i, j), (i, k), (j, k)\) are interrelated, we propose to predict the three edge probabilities \(\{P(I_{ij}|\Delta), P(I_{ik}|\Delta), P(I_{jk}|\Delta)\}\) (denoted \(\{e_{ij}(\Delta), e_{ik}(\Delta), e_{jk}(\Delta)\}\)) together, using as inputs the three embeddings \(z_i, z_j, z_k\), where \(z_i\)’s are the latent representations of the nodes learned by the graph encoder.

**Structure**

The structure of the proposed triad decoder is shown in Figure 1. First, we perform \(1 \times 1\) convolution, which corresponds to a linear transform, on the three node embeddings \(z_i, z_j, z_k\). This is followed by the rectified linear (ReLU) nonlinearity, producing the vector \(z_{\text{triplet}}\). Its output for the \(l\)-th dimension is ReLU(\(w_i z_{il} + w_j z_{jl} + w_k z_{kl}\)). Note that this contains information from the whole triad. Vector \(z_{\text{triplet}}\) is then further nonlinearly transformed by the fully connected (FC) layer and another ReLU nonlinearity. The whole block, which is denoted \(F\), is finally merged with the three inner products constructed from \(z_i, z_j, z_k\). This additional inner
product connections serve similarly as the residual connection in residual networks (He et al. 2016). The output of the triad decoder is:

\[
[e_{ij}(\Delta), e_{ik}(\Delta), e_{jk}(\Delta)] = \sigma(F(z_i, z_j, z_k) + [z_i^T z_j, z_i^T z_k, z_j^T z_k]).
\] (4)

When \(F\) outputs a zero mapping, it reduces to the standard inner product decoder.

### Model Training

The triad decoder can be readily used to replace the decoder in any graph-based auto-encoder. In this paper, we incorporate this into the VGAE and GAE, leading to the triad variational graph auto-encoder (TVGA) and the triad graph auto-encoder (TGA), respectively (Figure 2).

The whole network can be trained end-to-end, and the training procedure is shown in Algorithm 1. The total number of triads in the graph is \(\binom{N}{3}\), which is large. As is common in deep learning, we use stochastic gradient descent (SGD) or its variants for better scalability. In particular, the popular Adam optimizer (Kingma and Ba 2014) will be employed in the experiments. In each iteration, we sample \(B\) triads \(\{\Delta_1, \ldots, \Delta_B\}\) from the graph to form a mini-batch \(\mathcal{B}\). The probability for the presence of a particular edge \((i, j)\) in the mini-batch is

\[\tilde{e}_{ij} = \frac{e_{ij}}{M_{ij}}.\] (5)

where \(E_{ij} = \sum_{m=1}^{B} e_{ij}(\Delta_m)\), with \(e_{ij}(\Delta_m) = 0\) if \((i, j) \notin \Delta_m\). \(M_{ij}\) is the total number of times \((i, j)\) observed in the mini-batch (i.e., \(M_{ij} = \sum_{m=1}^{B} I_{ij}(\Delta_m)\), where \(I_{ij}(\Delta_m) = 1\) if \((i, j) \in \Delta_m\); and 0 otherwise).

### Optimization Objective

First, we consider TVGA. Let \(A_{ij}\) be the observed graph adjacency matrix. As in (Kipf and Welling 2016b), this variational model is trained by maximizing the variational lower bound:

\[
\sum_{\text{triad} \in \mathcal{B}} \sum_{(i, j) \in \text{triad}} (A_{ij} \log \tilde{e}_{ij} + (1 - A_{ij}) \log(1 - \tilde{e}_{ij})) - D_{KL}[q(Z|X, A)||p(Z)]
\] (6)

w.r.t. the encoder parameters \((W^{(1)}, W^{(2)}\) of the GCN) and decoder parameters. Here, the first term is the negative of the graph reconstruction error, while the second term is the Kullback-Leibler divergence between the encoder output distribution for the embeddings \(q(\cdot)\) in (2)) and some prior distribution \(p(\cdot)\). In this paper, we assume that the latent dimensions are i.i.d., and follow the standard normal distribution: \(p(Z) = \prod_{i=1}^{N} p(z_i) = \prod_{i=1}^{N} \mathcal{N}(0, I).\)

As for the non-variational version TGA, the Kullback-Leibler divergence is removed from (6) as in (Pan et al. 2018), and only the following graph reconstruction error is minimized:

\[
\sum_{\text{triad} \in \mathcal{B}} \sum_{(i, j) \in \text{triad}} (A_{ij} \log \tilde{e}_{ij} + (1 - A_{ij}) \log(1 - \tilde{e}_{ij})).
\] (7)

### Sampling the Training Triads

Real-world networks are typically sparse. If triads are sampled randomly from the graph in step 6 (of Algorithm 1), it is likely that most node pairs in the training triads are disconnected. This resultant high class imbalance can lead to severe performance deterioration for most classifiers.

To alleviate this problem, we construct a more balanced training data set by sampling each triad \(\Delta = \{i, j, k\}\) as follows. First, a node \(i\) is randomly sampled from the graph. Let \(\mathcal{N}(i)\) be the set containing all neighbors of \(i\). With probability \(p\), we sample the next node \(j\) from \(\mathcal{N}(i)\); and with probability \(1 - p\), sample \(j\) from a faraway node not in \(\mathcal{N}(i)\). After sampling \(j\), the last node \(k\) in the triad is similarly sampled from \(\mathcal{N}(j)\) with probability \(p\); and from a node not in \(\mathcal{N}(j)\) with probability \(1 - p\).

The class imbalance can be controlled by appropriately setting \(p\). First, note that the expected number of connected edges in a triad \(\Delta\) is \(E[I_{ij} + I_{jk} + I_{ik}] = E[I_{ij}] + E[I_{jk}] + E[I_{ik}]\).
the total number of triads. For all three nodes in the triad are connected to each other) to clustering coefficient (Wasserman and Faust 1994), which pairs $P$ exists between nodes $i$ and/or $(j, k)$ are connected (which are independent based on the sampling scheme). Thus, $E[I_{ij} + I_{jk} + I_{ik}] = p + p + p^2 P(I_{ik}|\text{observed 2 edges}) + 2p(1 - p)P(I_{ik}|\text{observed 1 edge}) + (1 - p)^2 P(I_{ik}|\text{observed 0 edge}).$

$P(I_{ik}|\text{observed 2 edges})$ can be estimated by the global clustering coefficient (Wasserman and Faust 1994), which is defined as the ratio of the number of closed triads (i.e., all three nodes in the triad are connected to each other) to the total number of triads. For $P(I_{ik}|\text{observed 1 edge})$ and $P(I_{ik}|\text{observed 0 edge})$, intuitively, the effect of the presence of one or zero edge on $I_{ik}$ is small. Thus, we simply assume that both probabilities are the same as the prior probability $P(I_{ik})$, which can be estimated from the graph density (Lawler 2001). Hence,

$$E[I_{ij} + I_{jk} + I_{ik}] = 2p + p^2 \cdot (\text{clustering coefficient}) + (1 - p^2) P(I_{ik}).$$

To construct a balanced training set, the desired $p$ can be obtained by setting the above to $3/2$.

**Space Complexity** VGAE/GAE (Kipf and Welling 2016b) and ARVGA/ARGA (Pan et al. 2018) take the whole graph as input. The space complexities are both $O(N^2)$, where $N$ is the number of nodes. For large graphs, the adjacency matrix may not even be able to fit into memory. In contrast, the proposed algorithm is trained on mini-batches of triads. The space complexity is $O(\max(N, B))$, which are much smaller than $N^2$ and thus much more scalable.

**Inference**

After training, the learned model can be used on a variety of graph learning tasks. In this section, we focus on link prediction, node clustering and graph generation.

**Link Prediction**

In link prediction, one wants to predict whether an edge exists between nodes $i$ and $j$. Recall that the proposed trial decoder predicts all three edges in the whole triad simultaneously, and so the three nodes need to be inputted together. In constructing these triads during inference, intuitively a node faraway from the node pair $(i, j)$ carries little information. Hence, instead of using both neighboring and faraway nodes to construct triads as in training, we only aggregate predictions from nodes $k$ that are in $N(i) \cup N(j)$. For each such $k$, the decoder predicts the probability for each $(i, j)$ edge using (4). The average probability over all these $k$’s is taken as the final probability for the presence of an edge.

On using NetGAN for link prediction, one has to first generate a number of random walks and accumulate the corresponding transition counts to an $N \times N$ matrix, which is then thresholded to produce the predicted links. Thus, the space and time complexities are both $O(N^2)$. In contrast, the proposed algorithm predicts the edge directly by (5) in $O(1)$ time. With $N_{\text{neighbor}}$ nodes in the union of neighborhoods of $i$ and $j$ (typically, $N_{\text{neighbor}} \ll N$), the total complexity is $O(N_{\text{neighbor}})$. Hence, the proposed algorithm is much more efficient. On the other hand, VGAE/GAE and ARVGA/ARGA only take $O(1)$ space and time. However, as will be seen in the experiments, their link prediction results are much inferior.

**Node Clustering**

With the learned node embedding, one can apply a standard clustering algorithm to cluster the nodes. Recall that the encoder and decoder in a graph-based auto-encoder are trained together in an end-to-end manner. Hence, though the decoder is not explicitly used in node clustering, an improved decoder (such as the proposed triad decoder) can guide the learning of better node representations in the embedding space.

**Graph Generation**

TVGA, which is based on the VAE framework, can be used to generate graphs. TGA, on the other hand, is not probabilistic and cannot be used for graph generation.

Let $N'$ be the target number of nodes to be generated. We first randomly sample $N'$ $z_i$’s from the posterior (normal) distribution of TVGA in the latent space. We then randomly sample $K$ triads from these $z_i$’s. For each triad, (4) is used to predict the probabilities for the three constituent edges. Finally, the predictions are averaged over all $K$ triads, and stored in an estimated adjacency matrix $\hat{A}$. To ensure symmetry, we replace each entry $A_{ij}$ of $\hat{A}$ by $(\hat{A}_{ij} + \hat{A}_{ji})/2$.

We assume that the generated graph has to be connected. Inspired by NetGAN, the following strategy is used to generate such a graph from $\hat{A}$. First, for every node $i$, we sample an edge $e_{ij}$ to node $j$ with probability $p_{ij} = \hat{A}_{ij}/\sum_{k=1}^{N} \hat{A}_{ik}$, and add it to the graph if it is new. Afterwards, we continue adding edges to the graph until the...
target number of edges have been generated. However, unlike NetGAN which uses sampling, we simply add edges in descending probability $p_{ij} = \hat{A}_{ij} / \sum_{u=1}^{N} \sum_{v=1}^{N} \hat{A}_{uv}$. Empirically, this has better performance as the whole local triad information is used in the proposed algorithm, and so $p_{ij}$ is more reliable.

### Experiments

In this section, we demonstrate the performance of the proposed algorithm on link prediction, node clustering and graph generation. Experiments are performed on three standard benchmark citation graph data sets\(^1\) (Sen et al. 2008): Cora, Citeseer, and Pubmed (Table 1). Each node represents an article, and has a boolean feature vector whose entries indicate whether a specific word occurs in the article. Each node also has a label, indicating the class it belongs to. The edges are citation links. We treat the graphs as undirected graphs. We remove all self-loops and undirected edges are citation links. We treat the graphs as undirected nodes also have a label, indicating the class it belongs to. The node labels do not indicate whether a specific word occurs in the article. Each node also has a label, indicating the class it belongs to. The edges are citation links. We treat the graphs as undirected graphs. We remove all self-loops and undirected edges are citation links. We treat the graphs as undirected graphs.

#### Table 1: Statistics for the (largest connected component of) graph data sets used.

<table>
<thead>
<tr>
<th></th>
<th>Cora</th>
<th>Citeseer</th>
<th>PubMed</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of nodes</td>
<td>2,485</td>
<td>2,120</td>
<td>19,717</td>
</tr>
<tr>
<td>number of edges</td>
<td>5,069</td>
<td>3,679</td>
<td>44,324</td>
</tr>
<tr>
<td>number of classes</td>
<td>7</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>feature dim</td>
<td>1,433</td>
<td>3,703</td>
<td>500</td>
</tr>
<tr>
<td>clustering coefficient</td>
<td>0.2376</td>
<td>0.1696</td>
<td>0.0602</td>
</tr>
<tr>
<td>graph density</td>
<td>0.0016</td>
<td>0.0016</td>
<td>0.0002</td>
</tr>
</tbody>
</table>

#### Table 2: Link prediction accuracy (%). TVGA(rand) and TGA(rand) are variants of TVGA and TGA, respectively, that use random triad sampling.

<table>
<thead>
<tr>
<th></th>
<th>Cora AUC</th>
<th>Cora AP</th>
<th>Citeseer AUC</th>
<th>Citeseer AP</th>
<th>PubMed AUC</th>
<th>PubMed AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>node2vec</td>
<td>86.9</td>
<td>88.7</td>
<td>88.1</td>
<td>89.2</td>
<td>92.2</td>
<td>92.3</td>
</tr>
<tr>
<td>SC</td>
<td>87.8</td>
<td>91.5</td>
<td>86.2</td>
<td>89.2</td>
<td>97.1</td>
<td>96.1</td>
</tr>
<tr>
<td>NetGAN</td>
<td>90.9</td>
<td>92.7</td>
<td>92.9</td>
<td>94.6</td>
<td>88.2</td>
<td>88.1</td>
</tr>
<tr>
<td>SEAL</td>
<td>93.3</td>
<td>94.6</td>
<td>92.8</td>
<td>93.4</td>
<td>95.8</td>
<td>96.3</td>
</tr>
<tr>
<td>VGAE</td>
<td>94.4</td>
<td>95.9</td>
<td>93.4</td>
<td>95.2</td>
<td>96.2</td>
<td>96.4</td>
</tr>
<tr>
<td>GAE</td>
<td>93.1</td>
<td>95.0</td>
<td>91.5</td>
<td>92.6</td>
<td>97.1</td>
<td>97.4</td>
</tr>
<tr>
<td>ARVGA</td>
<td>93.8</td>
<td>94.8</td>
<td>94.4</td>
<td>95.7</td>
<td>98.0</td>
<td>98.2</td>
</tr>
<tr>
<td>ARGA</td>
<td>94.2</td>
<td>95.6</td>
<td>93.5</td>
<td>95.0</td>
<td>95.5</td>
<td>96.0</td>
</tr>
<tr>
<td>TVGA(rand)</td>
<td>68.5</td>
<td>68.5</td>
<td>60.5</td>
<td>65.3</td>
<td>82.9</td>
<td>81.2</td>
</tr>
<tr>
<td>TVGA</td>
<td>96.0</td>
<td>96.3</td>
<td>96.2</td>
<td>96.5</td>
<td>98.5</td>
<td>98.7</td>
</tr>
<tr>
<td>TGA(rand)</td>
<td>76.3</td>
<td>76.9</td>
<td>69.2</td>
<td>69.1</td>
<td>89.6</td>
<td>88.7</td>
</tr>
<tr>
<td>TGA</td>
<td>95.6</td>
<td>96.2</td>
<td>96.5</td>
<td>97.0</td>
<td>98.2</td>
<td>98.4</td>
</tr>
</tbody>
</table>

#### Link Prediction

In this experiment, 85% of the edges and non-edges (unconnected nodes) from each graph are randomly selected to form the training set, another 10% is used as the validation set, and the remaining 5% as testing set. The proposed algorithm uses a mini-batch size of 5,000. Adam (Kingma and Ba 2014) is the optimizer, with a learning rate of 0.0005. Both the hidden layer and embedding layer of the encoder have 4 hidden units. The convolution layer in the triad decoder has 4 filters (i.e., the dimension of triplet is $1 \times 32 \times 4$).

#### Triad Sampling Scheme

First, we study how the proposed sampling scheme alleviates the class imbalance problem. Figure 3 shows the proportion of existent edges in the triads, with different sampling probabilities $p$.

With random sampling (yellow dot in the figure), the ratio of existent edges is very small, and the data set is highly imbalanced. As expected, using a larger $p$ means that node $j$ is more likely to be connected to node $i$, and node $k$ is more likely to be connected to node $j$, leading to a higher proportion of edges being observed in the sampled triads. The red dot corresponds to the $p$ value obtained by the proposed sampling scheme. As can be seen, the ratios of existent edges are all close to 0.5 on the three data sets, indicating that the proposed sampling scheme has effectively alleviated the class imbalance problem.

#### Comparison with the State-of-the-Art

The proposed algorithm (using both random triad sampling and the proposed sampling scheme) is compared with the following baselines: (i) node2vec (Grover and Leskovec 2016); (ii) spectral clustering (SC) (Bruna et al. 2014); (iii) NetGAN\(^2\) (Bojchevski et al. 2018); (iv) SEAL (Zhang and Chen 2018); (v) variational graph auto-encoder (VGAE) (Kipf and Welling 2016b); (vi) graph auto-encoder (GAE) (Kipf and Welling 2016b); (vii) adversarially regularized variational graph auto-encoder (ARVGA) (Pan et al. 2018); and (viii) adversarially regularized graph auto-encoder (ARGA) (Pan et al. 2018). SC and node2vec are only used to generate node embeddings. The inner product decoder in (3) is then used to obtain edge probabilities. NetGAN generates random walks and accumulates the corresponding transition counts, which are used to measure how likely there is an edge between two nodes. Note that node2vec, SC and NetGAN do not make use of node features. The other four baselines use the graph convolutional network (Kipf and Welling 2016a) as encoder. In particular, ARVGA and ARGA are modified from VGAE and GAE, respectively, by adding a discriminator that are trained adversarially.

For performance evaluation, we use the area under the ROC curve (AUC) and average precision (AP) as in (Kipf and Welling 2016b).

Results are shown in Table 2. As can be seen, node2vec, SC and NetGAN, which do not utilize node features, have the worst performance. TVGA and TGA outperform the other VAE-based methods, showing that triad information leads to more accurate predictions. Moreover, the proposed triad sampling scheme performs significantly better than random triad sampling (TVGA(rand) and TGA(rand)). Hence, we will only experiment with the proposed triad sampling scheme.

\(^1\)http://www.cs.umd.edu/~sen/lbc-proj/LBC.html

\(^2\)NetGAN has two early stopping schemes. Here, we use VAL-CRITERION as it is more related to generalization properties.
Figure 3: Ratio of existent edges in each triad, with different sampling probabilities $p$.

Table 3: Node clustering performance on Cora.

<table>
<thead>
<tr>
<th>Method</th>
<th>acc</th>
<th>NMI</th>
<th>F1</th>
<th>precision</th>
<th>adj-RI</th>
</tr>
</thead>
<tbody>
<tr>
<td>node2vec</td>
<td>0.674</td>
<td>0.475</td>
<td>0.658</td>
<td>0.689</td>
<td>0.422</td>
</tr>
<tr>
<td>SC</td>
<td>0.299</td>
<td>0.077</td>
<td>0.088</td>
<td>0.294</td>
<td>0.001</td>
</tr>
<tr>
<td>TADW</td>
<td>0.604</td>
<td>0.438</td>
<td>0.553</td>
<td>0.613</td>
<td>0.320</td>
</tr>
<tr>
<td>VGAE</td>
<td>0.662</td>
<td>0.482</td>
<td>0.639</td>
<td>0.648</td>
<td>0.433</td>
</tr>
<tr>
<td>GAE</td>
<td>0.602</td>
<td>0.460</td>
<td>0.591</td>
<td>0.591</td>
<td>0.389</td>
</tr>
<tr>
<td>ARVGA</td>
<td>0.616</td>
<td>0.457</td>
<td>0.599</td>
<td>0.608</td>
<td>0.378</td>
</tr>
<tr>
<td>ARGA</td>
<td>0.687</td>
<td>0.518</td>
<td>0.677</td>
<td>0.692</td>
<td>0.455</td>
</tr>
<tr>
<td>TVGA</td>
<td><strong>0.753</strong></td>
<td><strong>0.591</strong></td>
<td><strong>0.731</strong></td>
<td><strong>0.781</strong></td>
<td><strong>0.560</strong></td>
</tr>
<tr>
<td>TGA</td>
<td>0.728</td>
<td>0.558</td>
<td>0.711</td>
<td>0.747</td>
<td>0.512</td>
</tr>
</tbody>
</table>

Table 4: Node clustering performance on Citeseer.

<table>
<thead>
<tr>
<th>Method</th>
<th>acc</th>
<th>NMI</th>
<th>F1</th>
<th>precision</th>
<th>adj-RI</th>
</tr>
</thead>
<tbody>
<tr>
<td>node2vec</td>
<td>0.478</td>
<td>0.291</td>
<td>0.461</td>
<td>0.531</td>
<td>0.235</td>
</tr>
<tr>
<td>SC</td>
<td>0.258</td>
<td>0.037</td>
<td>0.090</td>
<td>0.247</td>
<td>0.003</td>
</tr>
<tr>
<td>TADW</td>
<td>0.581</td>
<td>0.371</td>
<td>0.480</td>
<td>0.481</td>
<td>0.354</td>
</tr>
<tr>
<td>VGAE</td>
<td>0.515</td>
<td>0.332</td>
<td>0.486</td>
<td>0.551</td>
<td>0.273</td>
</tr>
<tr>
<td>GAE</td>
<td>0.430</td>
<td>0.245</td>
<td>0.421</td>
<td>0.555</td>
<td>0.118</td>
</tr>
<tr>
<td>ARVGA</td>
<td>0.580</td>
<td>0.337</td>
<td>0.530</td>
<td>0.547</td>
<td>0.322</td>
</tr>
<tr>
<td>ARGA</td>
<td>0.584</td>
<td>0.370</td>
<td>0.536</td>
<td>0.572</td>
<td>0.339</td>
</tr>
<tr>
<td>TVGA</td>
<td>0.591</td>
<td>0.365</td>
<td>0.544</td>
<td>0.565</td>
<td>0.339</td>
</tr>
<tr>
<td>TGA</td>
<td><strong>0.611</strong></td>
<td><strong>0.401</strong></td>
<td><strong>0.564</strong></td>
<td><strong>0.600</strong></td>
<td><strong>0.387</strong></td>
</tr>
</tbody>
</table>

**Node Clustering**

In this section, we consider the unsupervised task of clustering nodes in the graph. We perform clustering on the obtained node embeddings using the $K$-means clustering algorithm, where $K$ is set to be the number of classes in Table 1. The following baselines are compared with the proposed algorithm: (i) node2vec; (ii) spectral clustering (SC); (iii) text-associated DeepWalk (TADW) (Yang et al. 2015); (iv) VGAE; (v) GAE; (vi) ARVGA; and (vii) ARGA. SC is a general clustering algorithm, while TADW is designed specifically for graphs. We do not compare with SEAL and NetGAN, since they do not produce node embeddings for clustering.

The node labels in Table 1 are used as ground-truth clustering labels. For performance evaluation, we follow (Xia et al. 2014) and first match the predicted labels with the ground-truth labels using the Munkres assignment algorithm (Munkres 1957), and then report the (i) accuracy (acc); (ii) normalized mutual information (NMI); (iii) F1-score (F1); (iv) precision; and (v) adjusted rand index (adj-RI). As in (Pan et al. 2018), we only evaluate on the Cora and Citeseer data sets.

Results on Cora and Citeseer are shown in Tables 3 and 4. TVGA and TGA outperform the other methods on both data sets across all metrics. SC does not perform well, as it does not utilize node features and is not designed for graphs.

**Graph Generation**

In this experiment, we train the model on an input graph, and then try to generate similar graphs. Following (Bojchevski et al. 2018), we only experiment on the Cora and Citeseer data sets. Moreover, as NetGAN can only generate graphs with the same number of nodes as the input graph, we set the numbers of nodes and edges to be generated to be equal to those in the input graph.

The following baselines are compared with the proposed model: (i) configuration model (conf. model) (Molloy and Reed 1995); (ii) degree-corrected stochastic block model (DC-SBM) (Karrer and Newman 2011); (iii) NetGAN\(^3\); (iv) GraphRNN (You et al. 2018); (v) variational graph auto-encoder (VGAE); and (vi) adversarially regularized variational graph auto-encoder (ARVGA). The configuration model and DC-SBM are classic graph generation algorithms, which model certain graph statistics directly. NetGAN generates graphs via accumulating random walks. GraphRNN generates nodes sequentially. Besides, note that the configuration model, DC-SBM, NetGAN and GraphRNN do not utilize node features. Moreover, we do not compare with GraphVAE (Simonovsky and Komodakis 2018) and the graph neural network based model in (Li et al. 2018), as they can only be used on very small graphs.

As in (Bojchevski et al. 2018), performance is evaluated by a number of statistics measured on the generated graph. These include the Gini coefficient, maximum degree, num-

\(^3\)Here, we use the early stopping scheme EO-CRITERION. As discussed in (Bojchevski et al. 2018), this gives the user control over graph generation.
The rightmost columns in Tables 5 and 6 show the average rank of each method over all statistics. TVGA, by utilizing the whole local triad information, and characteristic path length (i.e., average number of steps along the shortest paths for all node pairs).

Tables 5 and 6 show the results on Cora and Citeseer. Though the configuration model and DC-SBM excel at some metrics that they directly model (such as “maximum degree” and “power law exponent”), they fail to reproduce others. GraphRNN does not perform well. It has to be trained on small graphs with a maximum of 500 nodes (while the Cora and Citeseer data sets here are much larger).

The rightmost columns in Tables 5 and 6 show the average rank of each method over all statistics. TVGA, by utilizing both pairwise node information and triadic structure, ranks the highest and generates graphs with similar statistics to the original one (Hamilton, Ying, and Leskovec 2017). Figures 4 and 5 show parts of the generated graphs. As can be seen, the graphs generated by TVGA are more similar to the original graphs than the others. For example, in Figure 4, both the input and TVGA-generated graphs have three dominant clusters. However, this is not obvious in the other graphs generated. Similarly, in Figure 5, one can observe three dominant clusters (arranged in a triangle) in both the input and TVGA-generated graphs, but not in the others.

**Table 5: Statistics for the generated graphs on Cora. Results are averaged over 5 runs.**

<table>
<thead>
<tr>
<th>Method</th>
<th>Gini coeff.</th>
<th>max degree</th>
<th>triangle count</th>
<th>assortativity</th>
<th>power law exp.</th>
<th>clustering coeff.</th>
<th>charac. path len.</th>
<th>avg rank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td></td>
</tr>
<tr>
<td>original</td>
<td>0.397 ± 0.000</td>
<td>168 ± 0.000</td>
<td>113.6 ± 10.8</td>
<td>-0.019 ± 0.008</td>
<td>1.885 ± 0.000</td>
<td>3.09e-4 ± 2.94e-5</td>
<td>4.82 ± 0.01</td>
<td>3.57</td>
</tr>
<tr>
<td>conf. model</td>
<td>0.397 ± 0.000</td>
<td>168 ± 0.000</td>
<td>113.6 ± 10.8</td>
<td>-0.019 ± 0.008</td>
<td>1.885 ± 0.000</td>
<td>3.09e-4 ± 2.94e-5</td>
<td>4.82 ± 0.01</td>
<td>3.57</td>
</tr>
<tr>
<td>DC-SBM</td>
<td>0.476 ± 0.003</td>
<td>123 ± 7.66</td>
<td>333 ± 25.5</td>
<td>-0.028 ± 0.007</td>
<td>1.854 ± 0.004</td>
<td>1.75e-3 ± 1.15e-4</td>
<td>4.88 ± 0.02</td>
<td>4.14</td>
</tr>
<tr>
<td>NetGAN</td>
<td>0.372 ± 0.003</td>
<td>131 ± 2.87</td>
<td>874.0 ± 13.5</td>
<td>-0.074 ± 0.003</td>
<td>1.861 ± 0.002</td>
<td>4.63e-3 ± 1.63e-4</td>
<td>5.86 ± 0.02</td>
<td>2.29</td>
</tr>
<tr>
<td>GraphRNN</td>
<td>0.315 ± 0.006</td>
<td>33 ± 6.11</td>
<td>65.0 ± 11.3</td>
<td>0.095 ± 0.055</td>
<td>1.845 ± 0.007</td>
<td>3.48e-3 ± 5.82e-4</td>
<td>5.70 ± 0.08</td>
<td>5.00</td>
</tr>
<tr>
<td>VGAE</td>
<td>0.509 ± 0.002</td>
<td>348 ± 2.58</td>
<td>3731.0 ± 22.4</td>
<td>-0.154 ± 0.001</td>
<td>2.055 ± 0.004</td>
<td>1.04e-3 ± 2.15e-5</td>
<td>5.04 ± 0.04</td>
<td>6.00</td>
</tr>
<tr>
<td>ARVGA</td>
<td>0.563 ± 0.001</td>
<td>239 ± 6.12</td>
<td>7511.0 ± 337.0</td>
<td>-0.141 ± 0.002</td>
<td>2.168 ± 0.005</td>
<td>4.67e-3 ± 2.56e-4</td>
<td>6.02 ± 0.07</td>
<td>5.00</td>
</tr>
<tr>
<td>TVGA</td>
<td>0.389 ± 0.002</td>
<td>152 ± 5.84</td>
<td>1258.6 ± 32.0</td>
<td>-0.053 ± 0.002</td>
<td>1.879 ± 0.002</td>
<td>4.26e-3 ± 3.29e-4</td>
<td>5.42 ± 0.03</td>
<td>2.00</td>
</tr>
</tbody>
</table>

**Table 6: Statistics for the generated graphs on Citeseer. Results are averaged over 5 runs.**

<table>
<thead>
<tr>
<th>Method</th>
<th>Gini coeff.</th>
<th>max degree</th>
<th>triangle count</th>
<th>assortativity</th>
<th>power law exp.</th>
<th>clustering coeff.</th>
<th>charac. path len.</th>
<th>avg rank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td>avg std</td>
<td></td>
</tr>
<tr>
<td>original</td>
<td>0.428 ± 0.000</td>
<td>99 ± 0.000</td>
<td>43.2 ± 8.4</td>
<td>-0.011 ± 0.010</td>
<td>2.071 ± 0.000</td>
<td>5.18e-4 ± 1.01e-4</td>
<td>5.28 ± 0.04</td>
<td>3.43</td>
</tr>
<tr>
<td>conf. model</td>
<td>0.428 ± 0.000</td>
<td>99 ± 0.000</td>
<td>43.2 ± 8.4</td>
<td>-0.011 ± 0.010</td>
<td>2.071 ± 0.000</td>
<td>5.18e-4 ± 1.01e-4</td>
<td>5.28 ± 0.04</td>
<td>3.43</td>
</tr>
<tr>
<td>DC-SBM</td>
<td>0.514 ± 0.006</td>
<td>90 ± 3.56</td>
<td>158.2 ± 11.3</td>
<td>0.020 ± 0.006</td>
<td>1.957 ± 0.006</td>
<td>2.20e-3 ± 1.20e-4</td>
<td>5.09 ± 0.03</td>
<td>4.29</td>
</tr>
<tr>
<td>NetGAN</td>
<td>0.365 ± 0.003</td>
<td>73.2 ± 4.31</td>
<td>592.8 ± 27.5</td>
<td>-0.043 ± 0.005</td>
<td>1.988 ± 0.003</td>
<td>1.54e-2 ± 1.60e-3</td>
<td>7.55 ± 0.13</td>
<td>3.14</td>
</tr>
<tr>
<td>GraphRNN</td>
<td>0.313 ± 0.005</td>
<td>17 ± 1.85</td>
<td>89.2 ± 7.7</td>
<td>0.066 ± 0.015</td>
<td>1.964 ± 0.018</td>
<td>1.52e-2 ± 1.71e-3</td>
<td>7.55 ± 0.25</td>
<td>4.43</td>
</tr>
<tr>
<td>VGAE</td>
<td>0.495 ± 0.003</td>
<td>196 ± 4.17</td>
<td>4037 ± 114.4</td>
<td>-0.035 ± 0.006</td>
<td>2.221 ± 0.006</td>
<td>5.73e-3 ± 3.27e-4</td>
<td>7.03 ± 0.17</td>
<td>5.29</td>
</tr>
<tr>
<td>ARVGA</td>
<td>0.524 ± 0.004</td>
<td>139 ± 8.61</td>
<td>6126.8 ± 194.3</td>
<td>0.017 ± 0.018</td>
<td>2.293 ± 0.012</td>
<td>1.99e-2 ± 2.58e-3</td>
<td>7.90 ± 0.18</td>
<td>4.43</td>
</tr>
<tr>
<td>TVGA</td>
<td>0.428 ± 0.002</td>
<td>86 ± 3.07</td>
<td>1288.2 ± 40.0</td>
<td>0.033 ± 0.007</td>
<td>2.068 ± 0.004</td>
<td>1.78e-2 ± 1.17e-3</td>
<td>6.35 ± 0.06</td>
<td>2.71</td>
</tr>
</tbody>
</table>

**Figure 4:** An original subgraph from Cora, and the corresponding subgraphs generated by various methods.

**Figure 5:** An original subgraph from Citeseer, and the corresponding subgraphs generated by various methods.

Figure 4, both the input and TVGA-generated graphs have three dominant clusters. However, this is not obvious in the other graphs generated. Similarly, in Figure 5, one can observe three dominant clusters (arranged in a triangle) in both the input and TVGA-generated graphs, but not in the others.

**Conclusion**

In this paper, we proposed a novel triad decoder that uses the whole local triad information, and is able to model the triadic closure property that is fundamental in real-world networks. It can be readily used in any graph-based auto-encoder. Experimental results show that the proposed decoder, when used with the (variational) graph auto-encoder, outperforms the state-of-the-art on link prediction, node clustering and graph generation tasks.
**Acknowledgement**

This work was supported in part by Huawei PhD Fellowship.

**References**


913