Effective Decoding in Graph Auto-Encoder Using Triadic Closure

Han Shi,[1] **Haozheng Fan,**[1,2] **James T. Kwok**[1]
[1]Department of Computer Science and Engineering
Hong Kong University of Science and Technology, Hong Kong
[2]Amazon
{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 organizations 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 tradeoff between 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 VGAE 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 (VGAE) 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:

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

where $\widetilde{A} = A + I$, I is the identity matrix, $\widetilde{D}$ is a diagonal matrix with $\widetilde{D}_{ii} = \sum_{j=1}^{N} A_{ij}$, $H^{(l)}$ and $W^{(l)}$ are the feature map and weight at the lth layer, respectively, and relu$(\cdot)$ 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:

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

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

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

where μ_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:

$$\hat{A} = \sigma(ZZ^T), \tag{3}$$

where σ is the sigmoid activation function. For the VGAE, 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 $(i, j), (i, k)$, and (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 1×1 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_{triplet}. Its output for the lth 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_{triplet} is then further nonlinearly transformed by the fully connected (FC) layer and another ReLU nonlinearity. The whole block, which is denoted $\mathcal{F}$, is finally merged with the three inner products constructed from z_i, z_j, z_k. This additional inner

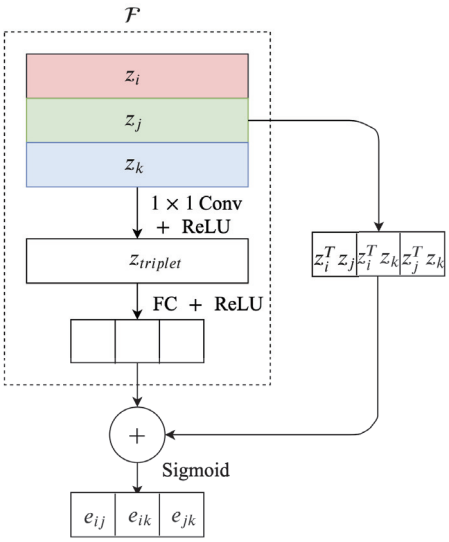


Figure 1: Structure of the triad decoder.

product connections serve similarly as the residual connection in residual networks (He et al. 2016). The output of the triad decoder is:

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

When $\mathcal{F}$ 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

$$\bar{e}_{ij} = E_{ij}/M_{ij}, \quad (5)$$

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:

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

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:

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

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 Δ is $E[I_{ij} + I_{jk} + I_{ik}] = E[I_{ij}] +$

Algorithm 1 Training the triad variational graph auto-encoder (TVGA) and triad graph auto-encoder (TGA) using SGD. Here, the encoder and decoder parameters are combined and denoted by w. $L_w(\{\bar{e}_{ij}\}, A, X)$ is the loss function (for TVGA, it is the negative of (6); for TGA, it is (7)).

1: initialize w_0;
2: $t \leftarrow 0$;
3: **while** w not converged **do**
4: $t \leftarrow t + 1$;
5: encode the graph G to latent embedding Z using (1) for TGA and (2) for TVGA;
6: sample B triads to form mini-batch $\mathcal{B}$;
7: obtain $\bar{e}_{ij}$'s for the triads from (4) and (5);
8: $w_t \leftarrow w_{t-1} - \alpha \nabla_w L_{w_{t-1}}(\{\bar{e}_{ij}\}, A, X)$;
9: **end while**
10: **return** Z.

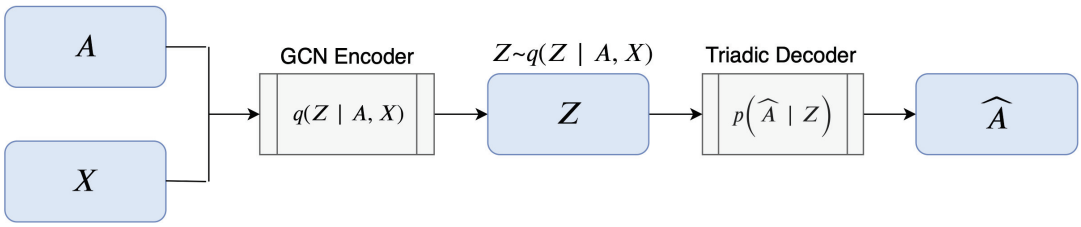


Figure 2: Structure of the graph-based auto-encoder with a triad decoder.

$E[I_{jk}]+E[I_{ik}]$. Using the above sampling scheme, $E[I_{ij}] = E[I_{jk}] = p$. As for $E[I_{ik}]$, this depends on the cases where pairs (i, j) and/or (j, k) are connected (which are independent based on the sampling scheme). Thus,

$$\begin{aligned} E[I_{ij} + I_{jk} + I_{ik}] =& p + p + p^2 P(I_{ik}|\text{observed 2 edges}) \\ &+ 2p(1-p)P(I_{ik}|\text{observed 1 edge}) \\ &+ (1-p)^2 P(I_{ik}|\text{observed 0 edge}). \end{aligned}$$

$P(I_{ik}|\text{observed 2 edges})$ 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,

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

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 $\mathcal{O}(N^2)$, 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 $\mathcal{O}(\max(N, B))$, 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 triad 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 $\mathcal{N}(i) \cup \mathcal{N}(j)$. 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 $\mathcal{O}(N^2)$. In contrast, the proposed algorithm predicts the edge directly by (5) in $\mathcal{O}(1)$ time. With N_{neighbr} nodes in the union of neighborhoods of i and j (typically, $N_{\text{neighbr}} \ll N$), the total complexity is $\mathcal{O}(N_{\text{neighbr}})$. Hence, the proposed algorithm is much more efficient. On the other hand, VGAE/GAE and ARVGA/ARGA only take $\mathcal{O}(1)$ 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 $\hat{A}_{ij}$ 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

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

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

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, and all self-loops are removed. Moreover, we only use the largest connected component in each graph.

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 32 hidden units. The convolution layer in the triad decoder has 4 filters (i.e., the dimension of z_{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.

[1] http://www.cs.umd.edu/~sen/lbc-proj/LBC.html

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

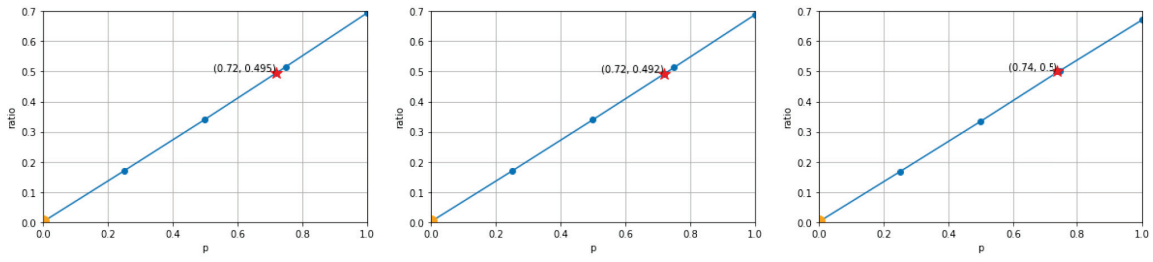
	Cora		Citeseer		PubMed	
	AUC	AP	AUC	AP	AUC	AP
node2vec	86.9	88.7	88.1	89.2	92.2	92.3
SC	87.8	91.5	86.2	89.2	97.1	96.1
NetGAN	90.9	92.7	92.9	94.6	88.2	88.1
SEAL	93.3	94.6	92.8	93.4	95.8	96.3
VGAE	94.4	95.9	93.4	95.2	96.2	96.4
GAE	93.1	95.0	91.5	92.6	97.1	97.4
ARVGA	93.8	94.8	94.4	95.7	98.0	98.2
ARGA	94.2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

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

[2] NetGAN has two early stopping schemes. Here, we use VAL-CRITERION as it is more related to generalization properties.



(a) Cora. (b) Citeseer. (c) PubMed.

Figure 3: Ratio of existent edges in each triad, with different sampling probabilities p.

Table 3: Node clustering performance on Cora.

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

Table 4: Node clustering performance on Citeseer.

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

sampling scheme in the sequel.

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-

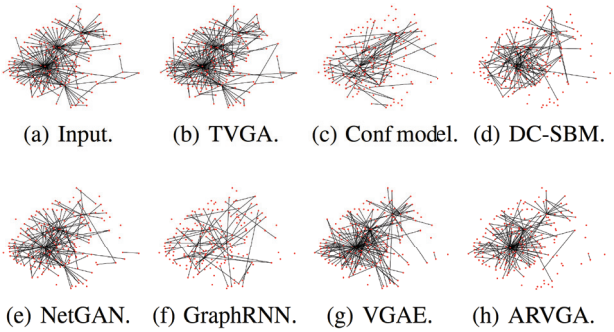
[3]Here, we use the early stopping scheme EO-CRITERION. As discussed in (Bojchevski et al. 2018), this gives the user control over graph generation.

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

	Gini coeff.		max degree		triangle count		assortativity		power law exp.		clustering coeff.		charac. path len.		avg rank
	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std	
original input	0.397		168		1558		-0.071		1.885		4.24e-3		6.31		
conf. model	0.397	± 0.000	168	± 0.00	113.6	± 10.8	-0.019	± 0.008	1.885	± 0.000	3.09e-4	± 2.94e-5	4.82	± 0.01	3.57
DC-SBM	0.476	± 0.003	123	± 7.06	333	± 25.5	-0.028	± 0.007	1.854	± 0.004	1.75e-3	± 1.15e-4	4.88	± 0.02	4.14
NetGAN	0.372	± 0.003	131	± 2.87	874.0	± 13.5	-0.074	± 0.003	1.861	± 0.002	4.63e-3	± 1.63e-4	5.86	± 0.02	2.29
GraphRNN	0.315	± 0.006	33	± 6.11	65.0	± 11.3	0.095	± 0.055	1.845	± 0.007	3.48e-3	± 8.52e-4	5.70	± 0.08	5.00
VGAE	0.509	± 0.002	348	± 2.58	3731.0	± 22.4	-0.154	± 0.001	2.055	± 0.004	1.04e-3	± 2.15e-5	5.04	± 0.04	6.00
ARVGA	0.563	± 0.001	239	± 6.12	7511.0	± 337.0	-0.141	± 0.002	2.168	± 0.005	4.67e-3	± 2.56e-4	6.02	± 0.07	5.00
TVGA	0.389	± 0.002	152	± 5.84	1258.6	± 32.0	-0.053	± 0.002	1.879	± 0.002	4.26e-3	± 3.29e-4	5.42	± 0.03	2.00

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

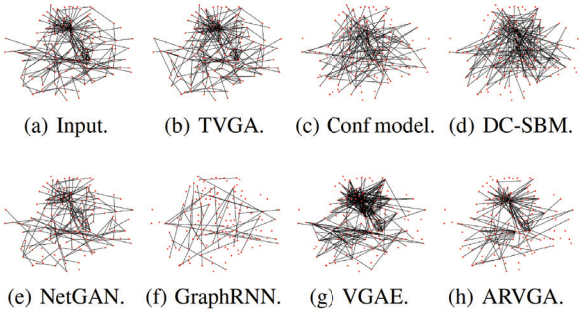
	Gini coeff.		max degree		triangle count		assortativity		power law exp.		clustering coeff.		charac. path len.		avg rank
	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std	
original input	0.428		99		1084		0.008		2.071		1.30e-2		9.33		
conf. model	0.428	±0.000	99	±0.00	43.2	±8.4	-0.011	±0.010	2.071	±0.000	5.18e-4	±1.01e-4	5.28	±0.04	3.43
DC-SBM	0.514	±0.006	90	±3.56	158.2	±11.3	0.020	±0.006	1.957	±0.008	2.20e-3	±1.20e-4	5.09	±0.03	4.29
NetGAN	0.365	±0.003	73.2	±4.31	592.8	±27.5	-0.043	±0.005	1.988	±0.003	1.54e-2	±1.60e-3	7.55	±0.13	3.14
GraphRNN	0.313	±0.005	17	±1.85	89.2	± 7.7	0.066	±0.015	1.964	±0.018	1.52e-2	±1.71e-3	7.55	±0.25	4.43
VGAE	0.495	±0.003	196	±4.17	4037	±114.4	-0.035	±0.006	2.221	± 0.006	5.73e-3	±3.27e-4	7.03	±0.17	5.29
ARVGA	0.524	±0.004	139	±8.61	6126.8	±194.3	0.017	±0.018	2.293	±0.012	1.99e-2	±2.58e-3	7.90	±0.18	4.43
TVGA	0.428	±0.002	86	±3.07	1288.2	±40.0	0.033	±0.007	2.068	±0.004	1.78e-2	±1.17e-3	6.35	±0.06	2.71



(a) Input. (b) TVGA. (c) Conf model. (d) DC-SBM.

(e) NetGAN. (f) GraphRNN. (g) VGAE. (h) ARVGA.

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



(a) Input. (b) TVGA. (c) Conf model. (d) DC-SBM.

(e) NetGAN. (f) GraphRNN. (g) VGAE. (h) ARVGA.

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

ber of triangles, assortativity (i.e., Pearson correlation of degrees of connected nodes), power law exponent (i.e., exponent of the power law distribution for degrees), clustering coefficient (as defined in NetGAN), 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 all breadth-first-search orderings (which can be in the order of $\mathcal{O}(|V|!)$). In (You et al. 2018), GraphRNN has only been 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.

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

Belanger, D., and McCallum, A. 2016. Structured prediction energy networks. In *International Conference on Machine Learning*, 983–992.

Bianconi, G.; Darst, R. K.; Iacovacci, J.; and Fortunato, S. 2014. Triadic closure as a basic generating mechanism of communities in complex networks. *Physical Review E* 042806.

Bojchevski, A.; Shchur, O.; Zügner, D.; and Günnemann, S. 2018. NetGAN: Generating graphs via random walks. In *International Conference on Machine Learning*, 609–618.

Bruna, J.; Zaremba, W.; Szlam, A.; and LeCun, Y. 2014. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*.

Caplow, T. 1968. *Two against one: Coalitions in triads*. Prentice-Hall.

Globerson, A.; Roughgarden, T.; Sontag, D.; and Yildirim, C. 2015. How hard is inference for structured prediction? In *International Conference on Machine Learning*, 2181–2190.

Goyal, P., and Ferrara, E. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 78–94.

Granovetter, M. 1973. The strength of weak ties. *American Journal of Sociology* 1360–1380.

Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *International Conference on Knowledge Discovery and Data Mining*, 855–864.

Hamilton, W.; Ying, R.; and Leskovec, J. 2017. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin* 52–74.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *International Conference on Computer Vision and Pattern Recognition*, 770–778.

Karrer, B., and Newman, M. 2011. Stochastic blockmodels and community structure in networks. *Physical Review E* 83(1).

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Kingma, D., and Welling, M. 2014. Auto-encoding variational bayes. In *International Conference on Learning Representations*.

Kipf, T., and Welling, M. 2016a. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Kipf, T., and Welling, M. 2016b. Variational graph auto-encoders. In *NIPS Workshop on Bayesian Deep Learning*.

Lawler, E. L. 2001. *Combinatorial optimization: Networks and matroids*. Dover.

Li, Y.; Vinyals, O.; Dyer, C.; Pascanu, R.; and Battaglia, P. 2018. Learning deep generative models of graphs. Preprint arXiv:1803.03324.

Molloy, M., and Reed, B. 1995. A critical point for random graphs with a given degree sequence. *Random Structures & Algorithms* 6(2-3):161–180.

Munkres, J. 1957. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics* 32–38.

Pan, S.; Hu, R.; Long, G.; Jiang, J.; Yao, L.; and Zhang, C. 2018. Adversarially regularized graph autoencoder for graph embedding. In *International Joint Conferences on Artificial Intelligence*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI Magazine* 93.

Simmel, G. 1908. *Sociology: Investigations on the forms of sociation*.

Simonovsky, M., and Komodakis, N. 2018. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, 412–422.

Wang, C.; Pan, S.; Long, G.; Zhu, X.; and Jiang, J. 2017. MGAE: Marginalized graph autoencoder for graph clustering. In *International Conference on Information and Knowledge Management*, 889–898.

Wang, Z.; Chen, C.; and Li, W. 2017. Predictive network representation learning for link prediction. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, 969–972.

Wasserman, S., and Faust, K. 1994. *Social network analysis: Methods and applications*. Cambridge University Press.

Xia, R.; Pan, Y.; Du, L.; and Yin, J. 2014. Robust multi-view spectral clustering via low-rank and sparse decomposition. In *AAAI Conference on Artificial Intelligence*, 2149–2155.

Yang, C.; Liu, Z.; Zhao, D.; Sun, M.; and Chang, E. 2015. Network representation learning with rich text information. In *International Joint Conferences on Artificial Intelligence*, 2111–2117.

You, J.; Ying, R.; Ren, X.; Hamilton, W.; and Leskovec, J. 2018. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, 5694–5703.

Zhang, M., and Chen, Y. 2018. Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, 5165–5175.

Zhang, D.; Yin, J.; Zhu, X.; and Zhang, C. 2018. Network representation learning: A survey. *IEEE Transactions on Big Data*.

Zhou, L.; Yang, Y.; Ren, X.; Wu, F.; and Zhuang, Y. 2018. Dynamic network embedding by modeling triadic closure process. In *AAAI Conference on Artificial Intelligence*.