Exploration on Multi-relational Graph Generation Report

Jiao Sun
jiaosun@usc.edu
University of Southern California

Abstract

Graphs are a fundamental structure to model relational data including text, images, knowledge bases, etc. Recent research efforts have shown that graph neural networks are powerful for representing graph-structured data as well as learning deep generative models for graphs. However, there are few works in generative models for multi-relational graphs. We will introduce the possible application scenarios where the multi-relational graph generation may help, and some ideas about how we can accomplish the task. In this report, I will show some preliminary experiment result on using graph generative models on scene graph generation tasks and molecular graph generation.

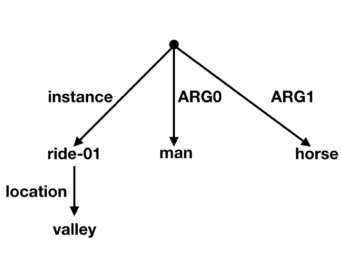

Figure 1: A generated AMR graph, whose corresponding text is "A man is riding a horse in a valley."

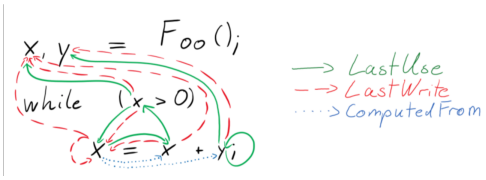

Figure 2: Represent a program with graph

1 Introduction

As one of the most fundamental data structures, graphs can capture the relational information well. There are lots of domains where understanding the graph structure is crucial to solving the problems, such as understanding the evolvement of communities, generating molecular structures for drug discovery, as well as to discover new information in the knowledge graph. In this section, we will introduce the scenarios where multi-relational graph generation will help the most.

1.1 AMR Graph Generation

Abstract Meaning Representation (AMR) (Banarescu et al., 2013) is a semantic formalism that encodes the meaning of a sentence as a rooted, directed graph. We can always model AMR graph as $G = (V, E)$, where V, E denotes the sets of nodes and edges. V represents the entities in the sentence while E represents the relation among different entities. If we can genera AMR graphs, then we can use the graph to generate text (Song et al., 2018), which means we are using the graph-based methods to generate text from scratch. Here we give an example of a generated AMR graph to text.

1.2 Scene Graph and Code Generation

Understanding a visual scene goes beyond recognizing individual objects in isolation. Relationships between objects also constitute rich semantic information about the scene. Xu et al. (2017) use the iterative message passing to generate scene graphs. If we can get generated scene graphs by graph generation techniques, then we can also get generated images (Johnson et al., 2018), as we show in Figure 3. Moreover, it also works for the code generation. We can represent a program as graph as shown in Figure 2 (Allamanis et al., 2018). Representing code as graphs can better capture the dependency among variables compared to capturing its shallow textual structure (Bhoopchand et al., 2016) or parse trees (Bielik

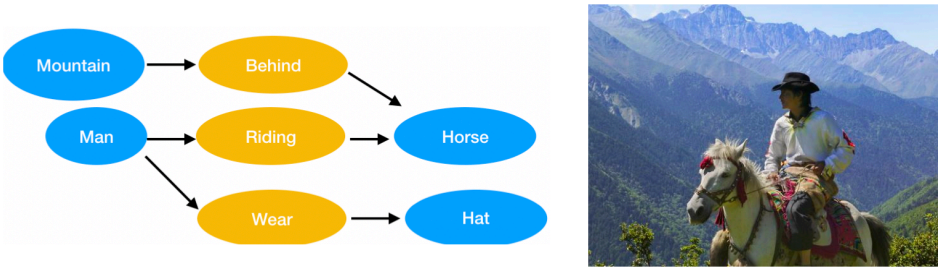

Figure 3: A scene graph and the corresponding image

et al., 2016). If we can generate the corresponding graphs for different programs, then we can get the machine-written code, which is a hot topic nowadays.

2 Related Work

2.1 Representation Learning on Graphs

Graphs are a fundamental abstraction for modeling relational data (Grover et al., 2018). Matrix factorization, random walk are all used for graph representation. Recently, researchers have raised interests on using neural networks to embed the graph information. The idea behind these representation learning approaches is to learn a mapping that embeds nodes, or entire (sub)graphs as points in a low-dimensional vector space (Hamilton et al., 2017). There are lots of graph neural networks operate over graphs using message passing.

Kipf and Welling (2016) developed an efficient variant of convolutional neural network with a layer-wise propagation rule to pass messages. It encodes the information from both the graph structure and node features. Authors demonstrate the efficiency with semi-supervised classification tasks. There are some other similar works to GCN (Li et al., 2016; Battaglia et al., 2016).

Veličković et al. (2017) propose graph attention networks GAT leveraging masked self-attention layers to address the shortcomings of the previous graph convolutional neural networks. It provides more modeling power than its ancestors.

Grover et al. (2018) propose a latent variable generative model for graphs based on variational autoencoding (Kingma and Welling, 2013). It uses graph neural networks for both the inference and generation.

Li et al. (2018b) propose to use graph neural networks to express probabilistic dependencies among a graph's nodes and edges. It claims that the networks can learn distributions over any arbitrary graph.

Based on the deep understanding of the graph structure and node information, researchers also use deep neural networks to generate graphs. You et al. (2018a) propose GraphRNN to both model distributions over graphs and sample from these distributions. It decomposes the graph generation process into node sequences and edge formation. Researchers also adopt the idea of GAN (Goodfellow et al., 2014b) to the graph generation domain. NetGAN (Bojchevski et al., 2018a) learns the distribution by random walks over the graph and mimic the original input.

2.2 Relation Modeling

Researchers used to create auxiliary triples and add to the learning objective factorization model to model the relation (Guu et al., 2015; García-Durán et al., 2015).

Schlichtkrull et al. (2017) introduce R-GCNs, first showing that GCN frames can be used to model relational data and proves the efficiency with link prediction and entity classification tasks.

Knowledge graphs are widely used in lots of domains. Extracting the information among entities in the knowledge base has recently received considerable attention. In the recommendation field, Wang et al. (2019) propose KGNN-LS, which transform the knowledge graph into a user-specific weighted graph and apply a graph neural network. The biggest difference of between this work and other works is that KGNN-LS is designed for heterogeneous knowledge graphs, while others are designed for homogeneous bipartite graphs.

2.3 Text Generation with Graph Neural Networks

Although lots of current works deal with both the graph representation, graph generation and relation modeling, there are fewer works in the natural

language processing domain, which can combine the relation extraction and graph representation together.

Researchers have some attempts about extracting the graph structure from the text information. Abstract Meaning Representation (AMR) graph generation was one of the popular methods. Bojchevski et al. (2018a) first applies the neural network to parse and generate text using AMR. However, they model the AMR structure with sequence encoding instead of operating on the graph structure, which may lose some information.

The most recent work is GraphWriter (Koncel-Kedziorski et al., 2019). Instead of using AMR, authors use a title and a transformed knowledge graph as input, encodes them respectively and generates text using GAT (Veličković et al., 2017) as we mentioned in Section 2.1.

3 Preliminary Methods Exploration

Existing work about graph generation Deep learning based methods have shown their superiorities for graph generation on preserving both the structural information and the node information. You et al. (2018b) proposed Graph-RNN to generate graph from sequence with BFS. Li et al. (2018a) propose DGMG to generate nodes and edges according to the probability of choosing the next step. Bojchevski et al. (2018b) propose NetGAN to generate entire realistic graphs via random walk. It conducts the objective function with the main adversarial idea from GAN (Goodfellow et al., 2014a). Grover et al. (2018) proposes Graphite, a latent variable generative model for graphs. Among all the proposed methods, Li et al. (2018a) is the closet work to ours, since it already embeds the node information when considering the graph generation, while other methods only consider the sequential information of nodes (i.e., the position of node in the graph).

Our task and two possible methodologies The main goal of our task is to generate graph with both the node information and the edge information.

1) Therefore, a simple way to complete our task would be based on the output of DGMG. The output will consist the node information and the graph structure information. Therefore, the task can be transformed as a network representation learning or knowledge representation learning problem. We can also think about it as a link prediction problem, where we need to predict the information of the link connecting two nodes. Researchers have also done lots of work under this topic (Chen et al., 2018).

The preliminary idea addressing the link prediction task would be using reinforcement learning based methods. Xiong et al. (2017) use the reinforcement learning framework for learning multi-hop relational paths with a policy-based agent with continuous states based on knowledge graph embeddings. Lin et al. (2018) reduce the impact of false negative supervision by adopting a pre-trained one-hop embedding model to estimate the reward of unobserved facts.

Therefore, we conclude the methods we propose here as two phases:

- Get the output graph from DGMG, which includes the embedding of the node information, please note that authors of DGMG already take the graph G as input and its node representation h_v to determine if to add the node, therefore we do not need to think about the node information anymore.

- Add the edge information to the generated graph. This progress can be modeled as a Markov decision process: given entity pairs (e_s, e_t), we want to find the most informative link to connect these entity pairs, which can be solved by reinforcement learning. State vector at step t is: $s_t = (e_t, e_{target} - _t)$ where e_t denotes the embedding of the current entity node, and e_{target} denotes the target embedding of the current node. A simple reward mechanism would be giving 1 if the link reaches the target, and -1 if it does not.

By doing so, we convert the task from multi-relational graph generation to graph generation with node information and link prediction with reinforcement learning.

2) The most ideal case would be designing an end-to-end network structure which can get the desired graph structure in the end, but it is more complicated than the first methods, a possible solution is what mentioned in the DGMG work. When they decide if to connect a node with another node, a function f_s maps pair h_u (the hidden state for u) and h_v to a score s_u for connecting u to the new node v. *This can be extended to handle typed edges by making s_u a vector of scores same size as the number of edge types, and taking the softmax*

	Name	Link
Molecular graph	ChEMBL dataset	https://www.ebi.ac.uk/chembl/
Scene graph	T.B.A	http://cvgl.stanford.edu/scene-graph/VG/VG-scene-graph.zip
AMR graph	The Little Prince	https://amr.isi.edu/download.html
Program	T.B.A	https://www.microsoft.com/en-us/download/details.aspx?id=56844

Table 1: The dataset we may try in different scenarios

over all node and edge types. But specifically how to do this still needs more exploration.

4 Experiment

As we mentioned in Section 1, there are different scenarios where multi-relational graph generation may help. Note that here we focus on the small graph generation, like AMR graph graph / scene graph / molecular graph. The number of nodes should be around 10 for each small graph, rather than hundreds of nodes like in the case for knowledge graph. Therefore, we collect some datasets which can be useful for our problem as shown in Table 1. We did the experiment mainly about testing how DGMG performs for relational graph dataset we collected here. Here I will mainly introduce two scenarios that I used for the experiment and my result, one is for molecular scenario and the other is for scene graph.

4.1 Molecular Scenario

The dataset we collect for the molecular scenario is the ENZYMES dataset. ENZYMES is a dataset of protein tertiary structures obtained from (Feragen et al., 2013) consisting of 600 enzymes from the BRENDA enzyme database (Neumann et al., 2016). In this case the task is to correctly assign each enzyme to one of the 6 EC top-level classes.

We can conclude the statistics information of the dataset as: about 600 graphs, and there are about twenty nodes in one graph. For the convenience of our experiment, we reduced the size of the the dataset to 200 graphs, which we call "enzyme small".

For the presentation of the graph, we have 1) the adjacency matrix for all graphs which represent the connectivity of the graph; 2) graph identifiers for all nodes in all graphs; 3) class labels for all graphs in the dataset, which is six in total in our experiment settings; 4) node labels.

Our task would be using generative models to get enzymes which can also fall in these six categories. The graph generative algorithms we used


```
[{'synsets': ['tree.n.01'],
  'h': 557,
  'object_id': 1058549,
  'merged_object_ids': [],
  'names': ['trees'],
  'w': 799,
  'y': 0,
  'x': 0},
 {'synsets': ['sidewalk.n.01'],
  'h': 290,
  'object_id': 1058534,
  'merged_object_ids': [5046],
  'names': ['sidewalk'],
  'w': 722,
  'y': 308,
  'x': 78},
 {'synsets': ['building.n.01'],
  'h': 538,
  'object_id': 1058508,
  'merged_object_ids': [],
  'names': ['building'],
  'w': 222,
  'y': 0,
  'x': 1},
 {'synsets': ['street.n.01'],
  'h': 258,
  'object_id': 1058539,
  'merged_object_ids': [3798578],
  'names': ['street'],
  'w': 359,
  'y': 283,
  'x': 439},
 {'synsets': ['wall.n.01'],
  'h': 535,
  'object_id': 1058543,
  'merged_object_ids': [],
  'names': ['wall'],
  'w': 135,
  'y': 1,
  'x': 0},
 {'synsets': ['tree.n.01'],
  'h': 360,
```


Figure 4: Object file in the scene graph dataset

	relationships	image_id
0	[{'predicate': 'ON', 'object': {'name': 'street', 'h': 262, 'object_id': 5046, 'synsets': ['stre...	1
1	[{'predicate': 'wears', 'object': {'name': 'backpack', 'h': 81, 'object_id': 5071, 'synsets': ['...	2
2	[{'predicate': 'in front of', 'object': {'name': 'monitor', 'h': 155, 'object_id': 1060246, 'syn...	3
3	[{'predicate': 'has', 'object': {'name': 'padding', 'h': 27, 'object_id': 5123, 'synsets': ['pad...	4
4	[{'predicate': 'ON', 'object': {'name': 'floor', 'h': 108, 'object_id': 1060359, 'synsets': ['fl...	5

Figure 5: Relationship file in the scene graph dataset

are GraphRNN (You et al., 2018b) and DGMG (Li et al., 2018a). The metric we used in our experiment is the Maximum Mean Discrepancy (MMD) of two distributions, here we use the degree distribution of two graphs. The result is shown in Table 2.

Suppose that a unit ball in a reproducing kernel Hilbert space (RKHS) H is used as its function class F, and k is the associated kernel, the squared MMD between two sets of samples from distributions p and q can be derived as $\text{MMD}^2 = E_{x,y\sim p}[k(x,y)] + E_{x,y\sim q}[k(x,y)] - 2E_{x\sim p,y\sim q}[k(x,y)]$.

4.2 Scene Graph Generation Scenario

Scene graphs use the graph structure to model the content of a picture. Scene graphs are normally used in the computer vision field, and there are plenty of scene graph datasets available for research. For example, there are Visual Genome dataset (https://visualgenome.org/), which is the most original dataset and also contains the most noisy data. With the development of the computer vision field, researchers propose more and more datasets which eliminate the previous disadvantages of the Visual Genome dataset, such as in Xu et al. (2017) (http://cvgl.stanford.edu/scene-graph/VG/VG-scene-graph.zip) as well as the dataset for the visual question answering task() (https://cs.stanford.edu/people/dorarad/gqa/vis.html). Here we use the one proposed in Xu et al. (2017) for the experiment.

First, let us have a look at what the dataset looks like. Figure 5 shows the json format of the relationship between objects in the picture, and their corresponding predicates. Figure 4 shows the json format of all objects in one image. We can see that every object belongs to a synset, which comes from the WordNet. However, please note that in our setting, we only need the name of the object, and there is normally one single name for one entity.

There are 108 graphs in total, which have about 23 nodes in average for one graph. Note that we may only have 17 unique nodes in one graph. For relationships in the dataset, we have about 21 relationships in average for one graph and about seven unique relationships.

For the simplicity reason, we randomly sampled 200 graphs from all graphs in the dataset. Similarly to the previous setting, we have the adjacency matrix for all graphs, graph identifier for all nodes in all graphs and node labels. Please note that in the previous cases, we did not use the node attributes. Instead, we just used the node label as indicator of the node type, and we already had the information of which label means which class before the experiment. However, here we need to utilize the nodes attribute.

The metric we are using for evaluation is still the MMD (Maximum Mean Discrepancy) for the degree distribution between two graphs, and the algorithms we are using are still GraphRNN and DGMG. We show the result in Table 2.

4.3 Experiment Result Analysis

By just getting the number, we have little idea about how our result is. Here we get the reported result table in the GraphRNN paper as shown in Figure 6. We can see that our result for the molecular dataset basically matches with the reported result in the GraphRNN paper, but the result for the scene graph scenario is quite bad compared to the molecular one.

We can conclude that by using the experiment we conducted above, deep generative models work well for the molecular dataset but not for scene graphs. There are some possible reasons that I could come up with:

1) there are too many node types for the scene graph scenario, and the interaction between node pairs is not too strong, which means for the adja-

Table 2. GraphRNN compared to state-of-the-art deep graph generative models on small graph datasets using MMD and negative log-likelihood (NLL). $(\max(|V|), \max(|E|))$ of each dataset is shown. (DeepVAE and GraphVAE cannot scale to the graphs in Table 1.)

	Community-small (20,83)					Ego-small (18,69)				
	Degree	Clustering	Orbit	Train NLL	Test NLL	Degree	Clustering	Orbit	Train NLL	Test NLL
GraphVAE	0.35	0.98	0.54	13.55	25.48	0.13	0.17	0.05	12.45	14.28
DeepGMG	0.22	0.95	0.40	106.09	112.19	0.04	0.10	0.02	21.17	22.40
GraphRNN-S	**0.02**	0.15	**0.01**	31.24	35.94	0.002	**0.05**	**0.0009**	8.51	9.88
GraphRNN	0.03	**0.03**	**0.01**	28.95	35.10	**0.0003**	**0.05**	**0.0009**	9.05	10.61

Figure 6: The reported result in the GraphRNN paper

cency matrix, it is really sparse that it could not hardly capture the interaction between different nodes.

2) for scene graphs, the information on the edge is quite critical for deciding if to connect two nodes, while for the molecular dataset, the node types make up the most important component in the graph generation process.

5 Rethinking about the problem

After the experiment, we figure out the generative model did not work well for the scene graph setting. While our original idea was to use the two step methodology introduced on the method section. Then we think the original idea may not work since we already did not get the satisfying result in the phase one. Therefore, if we continue our phase two based on the bad result of phase one, we may get into trouble.

We then rethink about the problem, we actually care more about how to get the relationship on the edges in the graph. It would have been ideal if we can both take care of the graph generation process and generating the edge information. However, it seems that we could not get the ideal result during the graph generation process in the multi-relational graph settings. If we only focus on the edge type generation, it could be an independent process apart from the graph generation for the graph structure.

We can rephrase it the research problem as *Structured prediction for edge tagging in multi-relational graphs*. The input for the problem is the node information and the graph structure, and the expected output for this problem would be the edge information.

We can think about some potential use cases for this problem itself, such as the recommendation for Turkers in the crowdsourcing setting, while humans only need to label the graph structure and the detailed information for edges in the graph can be recommended by the system.

There are some potential exploration directions that we can maybe think of.

- Link Prediction based methods. There are many multi-relational graph embedding methods available already, such as TransE, TransH, DisMult, etc. However, they only take into account the head information and tail information, while not caring about the graph structure. One obvious improvement that we could maybe do is to take into account the graph structure, which means improve the method from $f(h_i, t_i) -> r_i$ to $f(h_i, t_i, G) -> r_i$, where (h, t) indicates the head-tail pair, and G here means the Graph structure.

- Sequence tagging with edge ordering. While link-prediction based methods only consider the pair information. The sequence based methods also consider the result of previous steps. The first step is to oder the nodes and edges into a sequence, then we can learn a tagger for tagging the edges. More specifically, we can see the tagging process as a markov process, and there are already lost of work about POS-tagging with BLSTM-CRF models. We can also make it more oder-insensitive by taking M permutations and aggregate the result, the process can be represented as $f([e_1, e_2, ..., e_n], G) -> [y_1, y_2, ..., y_n]$.

- We can also directly do the graph-level edge tagging, the function would be $f(G) -> Y$, where Y means the edge information.

The above solutions are some initial ideas about how we could solve this problem, but more de-

	GraphRNN	DGMG
Molecular graph	0.0006	0.0023
Scene graph	1.753	3.0053

Table 2: MMD for molecular dataset and scene graph dataset

tailed and concrete plan still needs more careful considerations.

6 Conclusion

In this project, I started with the point of wanting to generate multi-relational graphs from scratch by using a two-step strategy: 1) generate graph structure; 2) label edge information in the generated graph structure. However, the result for the first step was not promising, and I come up with some other potential solutions to the second step which is independent from the first step, which will serve as the future directions.

I also want to quickly conclude what I got from the course: although the project does not go well as I expected, I learned important concepts and popular methods and algorithms in the NLP field, which will definitely benefit my future research. From the project, I also grasped the main idea and the structure of the most state-of-the-art graph generative models, which I think will also be beneficial to my potential research.

References

Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. 2018. Learning to represent programs with graphs. In *International Conference on Learning Representations*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, pages 178–186.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. 2016. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510.

Avishkar Bhoopchand, Tim Rocktäschel, Earl Barr, and Sebastian Riedel. 2016. Learning python code suggestion with a sparse pointer network. *arXiv preprint arXiv:1611.08307*.

Pavol Bielik, Veselin Raychev, and Martin Vechev. 2016. Phog: probabilistic model for code. In *International Conference on Machine Learning*, pages 2933–2942.

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018a. Netgan: Generating graphs via random walks. In *ICML*.

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemann. 2018b. Netgan: Generating graphs via random walks. *arXiv preprint arXiv:1803.00816*.

Haochen Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2018. A tutorial on network embeddings. *arXiv preprint arXiv:1808.02590*.

Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. 2013. Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems*, pages 216–224.

Alberto García-Durán, Antoine Bordes, and Nicolas Usunier. 2015. Composing relationships with translations. In *EMNLP*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014a. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. 2014b. Generative adversarial nets. In *NIPS*.

Aditya Grover, Aaron Zweig, and Stefano Ermon. 2018. Graphite: Iterative generative modeling of graphs. *arXiv preprint arXiv:1803.10459*.

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing knowledge graphs in vector space. In *EMNLP*.

William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584.

Justin Johnson, Agrim Gupta, and Li Fei-Fei. 2018. Image generation from scene graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Diederik P. Kingma and Max Welling. 2013. Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. 2019. Text generation from knowledge graphs with graph transformers. *ArXiv*, abs/1904.02342.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. 2016. Gated graph sequence neural networks. *CoRR*, abs/1511.05493.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. 2018a. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*.

Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter W. Battaglia. 2018b. Learning deep generative models of graphs. *ArXiv*, abs/1803.03324.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2018. Multi-hop knowledge graph reasoning with reward shaping. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3243–3253, Brussels, Belgium. Association for Computational Linguistics.

Marion Neumann, Roman Garnett, Christian Bauckhage, and Kristian Kersting. 2016. Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2017. Modeling relational data with graph convolutional networks. In *ESWC*.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. *arXiv preprint arXiv:1805.02473*.

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

Hongwei Wang, Fuzheng Zhang, Mengdi Zhang, Jure Leskovec, Miao Zhao, Wenjie Li, and Zhongyuan Wang. 2019. Knowledge-aware graph neural networks with label smoothness regularization for recommender systems. In *KDD*.

Wenhan Xiong, Thien Hoang, and William Yang Wang. 2017. DeepPath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 564–573, Copenhagen, Denmark. Association for Computational Linguistics.

Danfei Xu, Yuke Zhu, Christopher Choy, and Li Fei-Fei. 2017. Scene graph generation by iterative message passing. In *Computer Vision and Pattern Recognition (CVPR)*.

Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. 2018a. Graphrnn: A deep generative model for graphs. *ArXiv*, abs/1802.08773.

Jiaxuan You, Rex Ying, Xiang Ren, William L Hamilton, and Jure Leskovec. 2018b. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*.