Generating Graphs With Specified Properties And Their Use For Constructing Scene Graphs From Images

Aliaksei Himbitski
Faculty of Applied Mathematics and Computer Science
Belarusian State University
Minsk, Belarus
alekseygimbickiy@gmail.com

Vitali Himbitski
Faculty of Applied Mathematics and Computer Science
Belarusian State University
Minsk, Belarus
gimbitskyvitaly@gmail.com

Vassili Kovalev
Biomedical image analysis department
United Institute of Informatics Problems
Minsk, Belarus
vassili.kovalev@gmail.com

***Abstract*—Graph generation, the process of creating meaningful graphs, plays a vital role in various domains, including social network analysis, bioinformatics, recommendation systems, and network modeling. This article provides three graph generation models and also proposes the idea of constructing a scene graph using graph generation models. The where different models graph generation has been used for purposes such as social network analysis for community discovery, bioinformatics for protein interaction networks, recommendation systems for personalized recommendations, and network modeling for simulating real-world scenarios. In such models, the hidden state matrix of generated objects was used as a feature matrix. This article sets the goal of building a model with the ability to generate various types of graphs, without being tied to a specific area of application, that is, a matrix describing the structural characteristics of graphs will be used as a feature matrix.**

This paper develops three methods for generating graph structures with given properties using generative neural networks.

The developed methods are tested on the set of Hamiltonian graphs. A comparative analysis of the quality of the generated graph structures is performed. A method of scene graph construction using the developed methods is proposed.

Keywords—graph neural networks, generative neural networks, scene graph.

I. Introduction

Modern tasks make extensive use of graph structures, which allow us to represent relationships between elements of some set. Graphs are used in many fields, and their use is particularly useful for building a model of what is happening in an image. Such a graph structure is called a scene graph.

In this paper, three developed methods for generating graph structures with given properties are discussed. Based on these models, a method for constructing a scene graph is developed, which can be useful for the task of image description. In addition, the developed models can be applied to the development of self-driving cars.

Furthermore, the developed methods are integrated into a comprehensive approach for constructing scene graphs, which can be utilized for image description tasks. Given an input image, the proposed method generates a scene graph that captures the objects present, their relationships, and additional contextual information. This enables the automatic generation of descriptive captions or textual representations of visual scenes, facilitating tasks such as image understanding, retrieval, and summarization.

Beyond image description applications, the developed graph generation models have broader implications in the field of computer vision. For instance, in the context of self-driving cars, scene understanding plays a crucial role in perceiving and navigating the surrounding environment. By utilizing scene graphs, autonomous vehicles can better comprehend the relationships between objects on the road, pedestrians, and traffic signs, ultimately enhancing their ability to make informed decisions in complex driving scenarios.

In summary, this paper presents three methods for generating graph structures with given properties. In addition this paper proposes a way to apply these generative graph models to construct a scene graph. The presented method leverage object detection, deep learning, and probabilistic modeling techniques to construct scene graphs. The proposed approach can contribute to advancements in image description and other computer vision applications, with potential implications for autonomous driving systems

II. Generative adversarial neural network for graph generation.

A. *Generative adversarial neural network based on linear layers.*

The proposed model generates new graph structures from the adjacency matrices of the graphs represented in the dataset. Linear layers are used to determine the required graph propertie.

The architecture of the built model is shown below.

As a result, the model generates a matrix of size *n* x *n*, where *n* is the number of vertices in the generated graph. At the intersection of the *i*-th row and *j*-th column in this matrix there is a number from the segment [0, 1] denoting the probability that there is an edge between the *i*-th and *j*-th vertex in the graph. Here is an example of probability matrices generated in this way, represented as square images. In these images, the white color (*i*, *j*) of a cell means a high probability that there is an edge between the *i*-th and *j*-th vertices of the graph, and the black color means a low probability.

TABLE I. Generator architecture

Layers	***Input Shape -> Output Shape***	***Layers Information***
Input Layer	(100)->(400)	Linear BatchNorm LeakyReLU
Hidden Layer	(400)->(200)	Linear BatchNorm LeakyReLU
	(200)->(400)	Linear BatchNorm LeakyReLU
Output Layer	(400)->(64)	Linear
	(64)->(1, 8, 8)	Reshape

TABLE II. DISCRIMANTOR ARCHITECTURE

Layers	***Input Shape -> Output Shape***	***Layers Information***
Input Layer	(1, 8, 8)->(64)	Reshape
	(64)->(512)	Linear LeakyReLU Dropout
Hidden Layer	(512)->(512)	Linear LeakyReLU Dropout
Output Layer	(512)->(1)	Linear

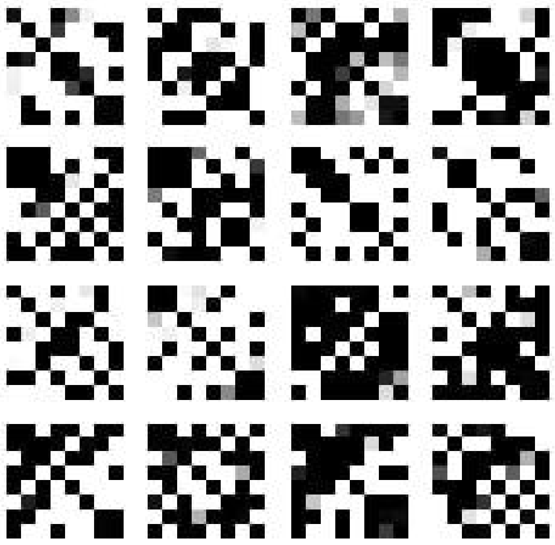
Fig. 1. Probability matrices obtained using a generative network

The following are examples of graphs obtained by generating with this model trained 50 epochs on a dataset consisting of Hamiltonian graphs.

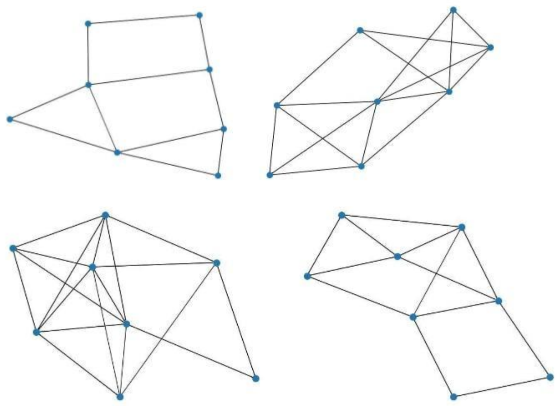
Fig. 2. Graphs obtained using a generative network

B. Graph convolution

The models proposed below utilize graph convolution. This section gives a brief description of this operation and provides some as that will be used to implement the graph convolution layer.

The idea of a graph convolution is that for each vertex of a graph it accumulates the features of all vertices adjacent to it. The graph convolution layer takes an adjacency matrix *A*, of size *n* x *n*, and a feature matrix *X* of size *n* x *k*, where *k* = = 1 + n + l. And returns some matrix *Y* obtained by the formula

$$Y = relu(\hat{A} \times X \times W), \text{ where } \hat{A} = A - D. \qquad (1)$$

Here *A* is the graph adjacency matrix, *D* is a diagonal matrix with the degrees of the graph vertices on the diagonal, *X* is the feature matrix of the graph vertices, *W* is some trainable parameters.

C. Graph autoencoder based on graph convolutions.

In order to preserve the structural features of the graph, the use of graph convolutions has been proposed. To train this autoencoder, graph adjacency matrices as well as feature matrices containing vector representation of each vertex of the graph are used. The architecture of the encoder and decoder is summarized in the table below.

TABLE III. ENCODER ARCHITECTURE

Layers	***Input Shape -> Output Shape***	***Layers Information***
Input Layer	(1, n, m) -> (1, n, m/4)	(GraphConvLayer, ReLU)x2
OutputLayer	(1, n, m/4)->(1, n, m)	(GraphConvLayer, ReLU)x3

TABLE IV. DECODER ARCHITECTURE

Layers	***Input Shape -> Output Shape***	***Layers Information***
Input Layer	(1, n, m) -> (1, 1, n*m)	Linear ReLU
Hidden Layer	(1, 1, n*m)-> (1, 1, $\lfloor \frac{3}{2} * n * m \rfloor$)	Linear ReLU
Output Layer	(1, 1, $\lfloor \frac{3}{2} * n * m \rfloor$)-> (1, 1, n)	Linear ReLU

Here *n* denotes the number of graph vertices, m is the dimension of the graph vertex embedding.

The encoder consists of two consecutive graph convolutional layers with ReLU activation function and collapses the graph into a hidden feature representation. The decoder in turn consists of three fully connected layers with ReLU activation function and translates the feature representation of the graph into a probability vector *p* of size *1* x *n*. We then obtain the probability matrix *P* by multiplying this vector by itself. By applying the sigmoid activation function, threshold function, we get the generated graph adjacency matrix.

$$P = p^T \cdot p. \quad (2)$$

As input, the encoder receives the source graph adjacency matrix and the vertex feature matrix.

Will train the obtained model for 20 epochs. As an optimizer will use Adam optimizer, set $learningrate = 10^{-3}$. The results of the model trained on the dataset of Hamiltonian graphs are shown below.

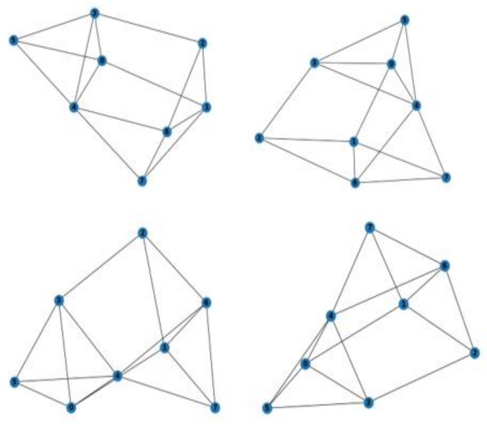
Fig. 3. Graphs obtained after 20 epochs of training the graph autoencoder

D. Generative adversarial neural network based on graph convolutions.

This model extracts graph features using graph convolutions. To train this generative adversarial network, graph adjacency matrices are used, as well as feature matrices containing a vector representation of each vertex in the graph.

An autoencoder is used to obtain the graph feature matrix. The autoencoder takes the graph adjacency matrix, collapses it into a latent space using an encoder, and reconstructs the graph adjacency matrix from the latent representation using a decoder. The matrix of the form, where d is the column of degrees of the graph vertices, A is the graph adjacency matrix, L is the latent representation of the graph adjacency matrix, is used as the feature matrix.

The architecture of the model is shown below.

Will train the obtained model for 20 epochs. Will use Adam optimizer as an optimizer. For the first 10 epochs will set $learningrate = 10^{-3}$. During the last 10 epochs will linearly decrease $learningrate$ to 0.

After 20 epochs of training on the dataset of Hamiltonian graphs, the model will generate the following graphs.

TABLE V. GENERATOR ARCHITECTURE

Layers	***Input Shape -> Output Shape***	***Layers Information***
Input Layer	(1, 8, 18)->(144)	Reshape
Hidden Layer	(144)->(144)	Linear ReLU Linear ReLU Linear ReLU
	(144)->(1, 8, 18)	Reshape
Decoder	(144)->(128)	Linear LeakyReLU
	(128)->(112)	Linear LeakyReLU
	(112)->(96)	Linear LeakyReLU
	(96)->(80)	Linear LeakyReLU
	(80)->(64)	Linear LeakyReLU
	(64)->(1, 8, 8)	Reshape
Output Layer	[(1, 8), (1, 8, 8), (1, 8, 18)] ->(1, 8, 27)	Concatenate

TABLE VI. DISCRIMINATOR ARCHITECTURE

Layers	***Input Shape -> Output Shape***	***Layers Information***
Conv Layer	[(1, 8, 8), (1, 8, 27)]-> (1, 8, 27)	GraphConvLayer GraphConvLayer GraphConvLayer
Classification Layer	(1, 8, 27)->(216)	Reshape

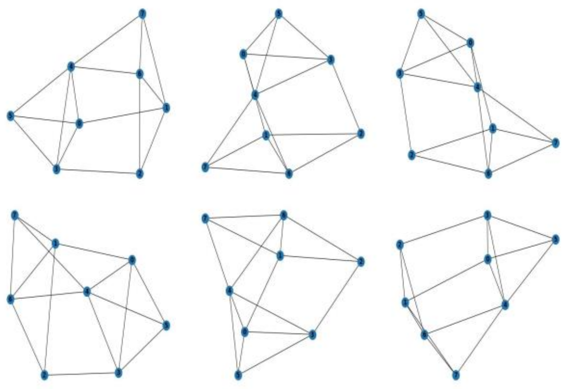
Fig. 4. Graphs obtained after 20 epochs of training the graph GAN

III. Graph-Based Generative Neural Networks in Computer Vision Tasks for Scene Graph Construction

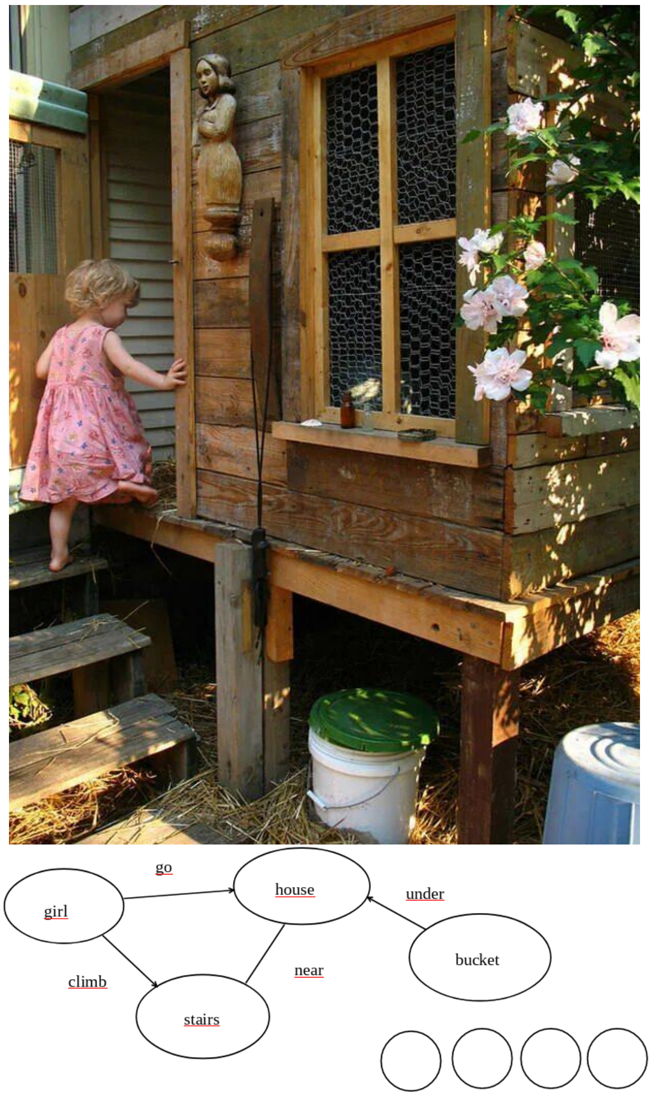

Fig. 5. The principle of scene graph construction

The proposed graph generative neural network architectures can be used to generate scene graphs to describe the actions taking place in the image. The new model will consist of two structural parts: a convolutional neural network to select objects in the image and a graph neural network to construct the scene graph. The graph part of the complex model will be represented by a generator of one of the above mentioned generative adversarial neural networks or a decoder of the above mentioned graph autoencoder.

$$\begin{tikzcd}
\boxed{\text{image}} \arrow[r] & \text{CNN} \arrow[r] & \boxed{X} \arrow[r] & \text{GNN} \arrow[r] \arrow[dr] & \boxed{A} \\
 & & & & \boxed{X}
\end{tikzcd}$$

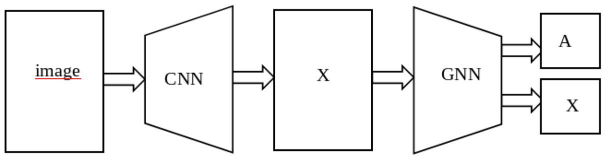

Fig. 6. Model structure for scene graph construction

Figure 5 shows the working principle of this model. The convolutional neural network takes an image as input and returns a matrix *X* of size *k* x *n*, which corresponds to the feature matrix of the scene graph. The graph neural network uses the feature matrix to construct the adjacency matrix of the scene graph of size *n* x *n* x *l*. Here n is the number of vertices in the scene graph, *k* is the length of the feature vector of the graph vertex, *l* is the length of the feature vector of the graph edge.

The proposed model takes an image as input and returns two matrices: matrix *X* and matrix *A*. *X* is a feature matrix whose rows correspond to feature vectors of objects detected in the image. *A* is the adjacency matrix, whose elements correspond to the vectors of features of the ways of interaction between pairs of objects detected in the image. The methods of generating scene graphs from images are not discussed in more detail in this paper.

IV. Conclusion

The were developed and tested. three models for generating graph structures with given properties. The results of generating Hamiltonian graphs where shown in the paper. A model based on the proposed generative graph neural networks for generating scene graphs from an image was also developed. Graph convolution was implemented and used in the development of graph neural network architectures.

References

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. *(references)*

[2] Lingfei Wu, Peng Cui, Jian Pei and Liang Zhao, “Graph Neural Networks Foundations, Frontiers, and Applications,” Springer Singapore, 2022 .

[3] Feature Extraction for Graphs. The Most Useful Graph Features for Machine Learning Models [Electronic resource], URL: https://towardsdatascience.com/feature-extraction-for-graphs-625f4c5fb8cd

[4] An Introduction to Graph Neural Network(GNN) For Analysing Structured Data [Electronic resource], URL: https://towardsdatascience.com/an-introduction-to-graph-neural-network-gnn-for-analysing-structured-data-afce79f4cfdc

[5] Zhiyuan Liu and Jie Zhou, “Introduction to Graph Neural Networks. Synthesis Lectures on Artificial Intelligence and Machine Learning,” Morgan & Claypool Publishers, 2020.