Representation Learning for Scene Graphs

Frits van Paasschen, Sophia Kivelson
{fritsvp, skivelso}@cs.stanford.edu

***Abstract*—Given the rich downstream use-cases for structured scene data in the form of a scene graph, as shown in [31], [30], [1], we are motivated to explore methods in which scene graphs can be generated, represented, and manipulated. Specifically, we present work towards automatically learning lower dimensional representations for scene graphs that might be useful for tasks such as graph comparison and generation. In this milestone, we introduce an attempt at learning latent space encodings for scene graphs from the GQA dataset [15] using a Variational Graph Autoencoder (Graph-VAE) [33] [10] with various Graph-Convolutional operators. Initial experiments on a downstream link-prediction task with the Graph-VAE were encouraging, but structural properties inherent to scene graphs motivated additional experiments with different architectures for graph representation learning and graph generation. As such, we present several applications of various Graph-VAE architectures on the task of learning lower dimensional scene graph representations.**

I. Introduction

Recently, there has been a significant amount of research into the topic of generating scene graphs from various media: most often images, text, or knowledge bases [23], [24], [17], [19], [20], [28], [27]. Initially, we aimed to create an encoder/decoder framework to embed graphs in a latent space and allow for conditional graph generation by experimenting with Graph-VAEs [10], Graph U-Nets [16], Graph Convolutional Neural Networks, recurrent methods for graph generation [18] [24] [19], and graph representation learning [12] [9] [7]. In this paper, we consider the applications of Graph-VAEs and Graph U-Nets to this domain. In combining this with future work on generative models, we hope to develop a novel framework for scene graph representation learning and/or latent space encoding that allows for the generation of rich new scene graphs that can be applied to various interesting use cases. This is particularly interesting because by learning scene graph representations, we are also effectively learning lower dimensional representations for the images whose information they capture.

II. Related Work

A. Scene Graphs

A scene graph [1] [30] $S : \{V, E, A\}$ consists of a multi-set of nodes V, a set of edge triples E, and a set of object attributes A that encode the semantic and organizational structure of an image. The multi-set of nodes $V : \{o \in O\}$ denotes objects present in the reference image. The set of edge triples $E : \{(o_i, r_k, o_j) \in O, R\}$ (where O and R represent the set of possible objects and relations) represent directed edges between objects o_i and o_j of class r_k. The set of attributes $A : \{(a_l, o_i) \in Q, V\}$ (where Q represents the set of possible attributes) represents the self-edges in the scene graph that encode information about their object nodes. We represent this compactly as an adjacency matrix A of a directed graph where attribute types are encoded multinomially on the diagonal, and edge types are encoded multinomially as well. We use this structured representation for all of our experiments.

B. Representation Learning

1) Graph VAE: The Graph-Variational Auto-Encoder [33] has the same general encoder-decoder architecture as a vanilla VAE, but replaces the conventional neural networks in the encoder and decoder with a graph-specific neural network encoder such as Graph Convolutional Networks (GCN) [3], among others, [2] and with inner product decoders. Similarly to VAEs, these models learn latent representations of their input graphs. While this model has, to our knowledge, not been applied to scene graph encoding before, it is a logical model to run experiments with given that it implictly learns a latent encoding of the data. This model also uses the same learning process as a vanilla VAE by optimizing the variational lower bound using the reparameterization trick [33]. More specifically, the model attempts to approximate a joint distribution of latent variables z and observed data x:

$$p_\theta(x, z) = p(x|z)p(z)$$

Which allows us to sample from $p(z)$ and then $p(x|z)$ to generate novel data. The optimization process attempts

to minimize a tractible approximation of the marginal log likelihood of the data over all possible z. This is done by introducing a tractible estimation $q(z)$ parameterized by λ of the true posterior $p(x|z)$. We approximate the log likelihood of the data $logp_\theta(x)$ using Jensen's inequality and $q(z)$ for the purposes of creating an objective called the Evidence Lower Bound (ELBO):

$$ELBO(x, \theta, \lambda) = \frac{1}{k}\sum_{i=1}^{n} log(\frac{p_\theta(x, z^{(i)})}{q_\lambda(z^{(i)})})$$

The Graph VAE effectively learns an encoder/decoder pair for graph representation and generation [10]. Advantages of this framework are that we can naturally incorporate node features, which can be learned through another technique [8] [9], and that we can use a simple link prediction task on our scene graph data to train the Graph VAE to generate meaningful latent representations. This task comprises the bulk of our experiments for the purpose of this milestone In addition, by sampling from the latent distribution of graph representations $p(z)$ and then inputting this into our decoder, we can generate entirely novel scene graphs. We also wish to explore interpolation [32] methods in this latent space to potentially manipulate scene graphs to exhibit different structural and semantic properties.

2) Graph U-Net: Convolutional Neural Networks have been applied to graphs in applications such as node classification and feature identification [2] [3]. A specific subtype of Convolutional Neural Network, the fully-Convolutional U-Net, has also been applied to graph structure [16]. This Graph U-Net architecture uses two main operations `gPool` and `gunPool` to simulate the down-and-upsampling behavior of a U-Net applied to conventional image data. For our application, graph U-Nets are potentially useful because of their ability to downsample graph structures into lower-dimensional feature vectors. In addition, their upsampling mechanism allows for the generation of more complex graphs from these lower-dimensional feature vectors. However, we have reservations about this approach as the `gPool` function has been shown to be much less effective in graphs with low connectivity (as the result of `gPool` may be a disconnected graph). Though scene graphs do not inherently have low connectivity, their small number of edges poses a problem. The relative sparsity of these graphs in terms of nodes and edges means that pooling operations will potentially destroy relevant graph information. As such, while the application of these models to scene graph representation learning was considered and initially tested, the underlying challenges of using a U-Net to learn graph representations caused us to focus on Graph-VAEs for encoding.

3) Node Embedding Techniques: Established node embedding techniques such as `node2vec` [8], DeepWalk [9], GraphSage [11], or TransE [7] are general methods at encoding the semantic information present in the graph. Given the relative simplicity of these methods and their proven performance, these methods are commonly used to encode graph information. However, they suffer from the problem of representational isomorphism, in that a collection of node vectors can be represented validly in any permuted ordering. Therefore, capturing a graph as a set of node vectors is not inherently useful when the structure of the graph itself must be directly encoded. For this application, we relied on the general priciple of generating node embeddings, as in the previous papers, but relied on different methods to capture the entire graph, such as the Graph-VAE.

C. Graph Generation

1) Auto-Regressive Models: One of the authors of this paper has in the past experimented with autoregressive scene graph generation and graph generation in previous projects [25] [21]. However, we aim to research the applications of these recurrent architectures to the task of scene graph generation: either by conditioning on the lower-dimensional scene graph representations we learn from another architecture, or by modeling the overall distribution of scene graphs using a Deep Generative Model. Autoregressive scene graph generation architectures currently perform well, and research has been done on a wide variety of these architectures. Autoregressive Deep Generative Models could be specifically suited for scene graph generation, such as [24], [19], [27], [18]. Along these lines, these models could be conditioned on the scene graph representations we learn from a model such as a Graph-VAE, or could compose the decoder for the Graph-VAE architecture, where the objective is some similarity metric to the original graph. For the Autoregressive Models that simply capture the distribution of scene graphs, such as the GraphRNN [18], these models can be trained through the same conditioning functionality, or simply sample from a trained model to generate scene graphs.

2) Latent Space Interpolation: Given semantically meaningful representations of scene graphs in a lower dimensional space (such as through one of the representation learning experiments listed above), they can be used to do interpolation in this space [32] to possibly manipulate generated graphs. We hypothesize that interpolation between areas in the learned lower dimensional representation might allow us to modify generated scene graphs.

Future work could investigate this by looking at scene graphs generated from interpolated representations, and possibly even images generated by conditioning on these interpolated scene graphs. Clustering algorithms might also giver further insight into the structure and semantic information stored in specific archetypal classes of scene graphs.

III. Data

A. Dataset

The dataset used for this project is GQA [15], which contains a selection of scene graphs from Visual Genome [14] that have been cleaned and pre-processed to generate a high fidelity scene graph dataset. Since we are interested in learning scene graph representations, we do not require our data to be explicitly labeled with external graph class labels or other data. For the purposes of our project, the encodings of the graphs in GQA were translated to corresponding adjacency matrix form with attributes represented as self-edges. The following table shows dataset statistics and presents an example of a data point in GQA that we used to train our models and run our experiments.

# Examples	# Rel.	# Obj.	# Attr.
74942	1702	309	616

An example of a scene graph, and associated image, from the dataset is as follows:

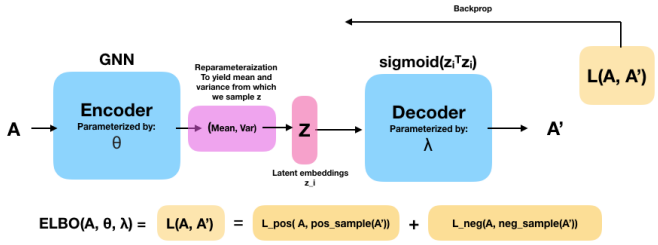
In order to represent and reason with these graphs in code, we wrote a custom dataset in PyTorch-Geometric [35], which contained all of the custom-preprocessed data from GQA. We were then able to layer various PyTorch-Geometric architectures and functionalities on top of this dataset.

IV. Methods and Experiments

A. VGAE implementation

As mentioned above, our experimentation has been geared towards exploring the efficacy of variational graph auto-encoding (VGAE) for learning a latent space for scene graphs. Our implementation is modeled after that described in Kipf et al.'s original paper proposing the method [10]. As detailed in the introduction and data discussions, we represent each graph in our dataset by a matrix A. Suppose Z is a matrix which stores the stochastic latent variables. (Intuitively, A captures the observed data, and we want to learn to map A to some Z which represents high-level or semantic information about A.) This matrix was encoded in a custom PyTorch-Geometric Dataset [35]. We first apply an encoder (inference model) consisting of a 4-layer convolutional architecture with a ReLu non-linearity. The main experimentation that we did using this architecture was in applying different convolutional or graph neural network architectures for this encoder. The bulk of our experiments composed of using different architectures for the encoder. Though it was discussed in more detail in B.1), the inference model is very basically trying to learn parametersθ to maximize $q(Z|A)$, while the generative, decoder learns a separate set of parameters λ to maximize $p(A|Z)$. The decoder works by taking the inner product between latent variables and applying a logistic sigmoid function. We learn then by optimizing the variational lower bound with respect to the parameters(also detailed in the VAE description in B.1)).

Though our goal is to learn a latent space for scene graphs, the VAE model trains on an edge prediction task. To do this, we calculate a loss by combining L_{pos}, based on the model's ability to predict the existence of edges where they ought to be and L_{neg}, which makes sure the model is not predicting the existence of edges where the should not be. The reconstruction loss is calculated as the sum L_{pos} and L_{neg}. The basic structure of the model is shown in the image below. (In various experiments detailed bellow we tried using GCN, GraphSage and GAT layers in the encoder in respective experiments.)



To train the model on an edge prediction task, we train on incomplete versions of the graphs where some

links have been removed. This is done randomly. A major problem we faced was that many of the scene graphs in our dataset were so small, that when edges were randomly removed we ended up with disconnected graphs which were not effective for training. On the other hand, there were cases where our graphs were fully connected, in which case we could not perform negative sampling because there were no no-existent edges. To circumvent this problem, we simply did not consider these graphs during training.

B. GAE Implementation

Our GAE model is very similar to our VGAE model, the main difference is that there is no reparameterization step as we don't try to sample z probabilistically. We calculate the embedding Z and reconstruct A deterministically as described by Kipf et. al. [10].

$$A' = \sigma(ZZ^T), \quad Z = GCN(A)$$

where A' is the reconstructed graph representation of A.

C. Graph Convolutional Network

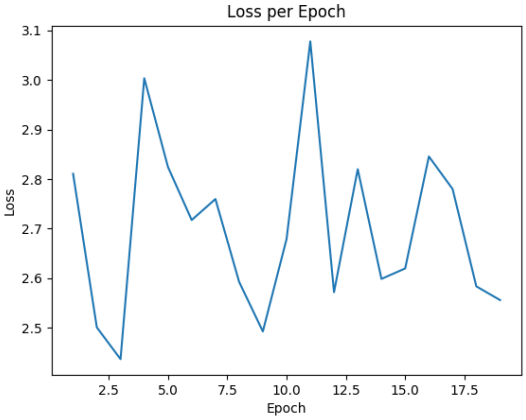
For our first experiment, we utilized a Graph Convolutional Operator from the Graph Convolutional Neural Network [3] as the layers of our encoder. We used a total of 5 layers of graph convolutions, with 32 channels. As in [2], these operators output a score X' given an input X:

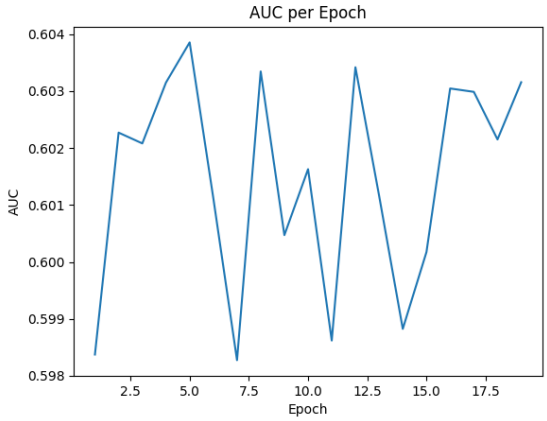
$$X' = \hat{D}^{\frac{-1}{2}}\hat{A}\hat{D}^{\frac{-1}{2}}X\Theta$$

Where $\hat{A}$ is the adjacency matrix with inserted self-loops and D is the graph's degree matrix.

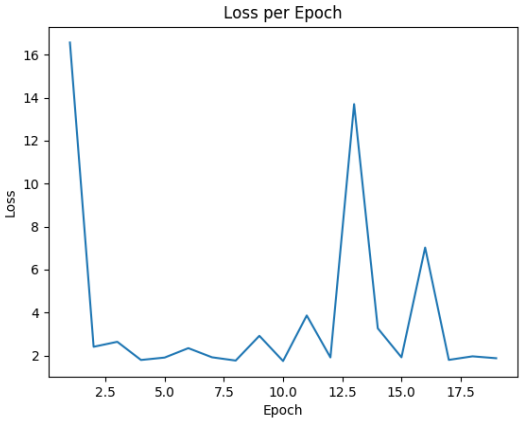
Below we show the AUC and training losses we achieved for our VGAE and GAE using GCN respectively:

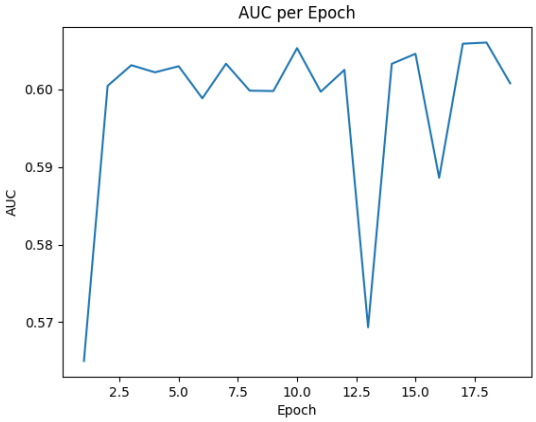
- **GAE with GCN**





- **VGAE with GCN**





Though both the GAE and VGAE achieved similar AUC after 20 epoch (just above 0.6), from the graphs it appears that VGAE is much more stable, and is able to maintain its AUC better. Because of this we find VGAE using the GCN to be a more promising approach.

D. GraphSAGE

The next experiment that we ran was to use the GraphSAGE [5] operator as the main neural network operation in our Graph-VAE Encoder. This architecture learns an aggregation of neighboring node feature vectors

to encode the entire graph as follows, where x' is a learned node feature vector:

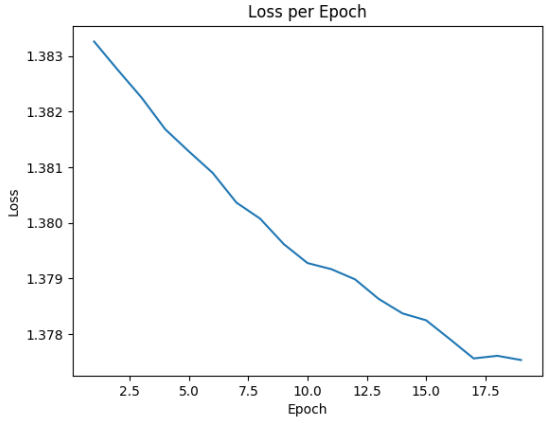
$$x'_i = \Theta mean_{j \in N(i) \cup j}(x_j)$$

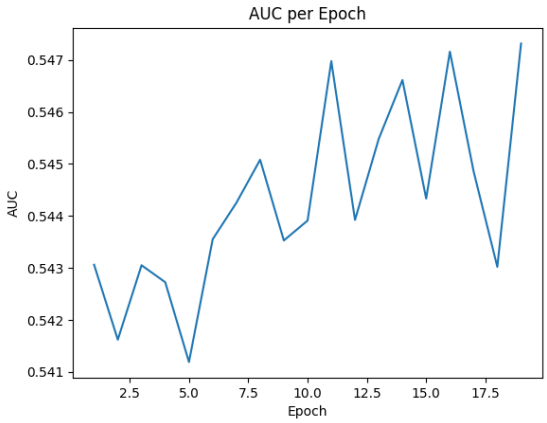
The resulting vectors are then normalized:

$$x'_i = \frac{x'_i}{||x'_i||}$$

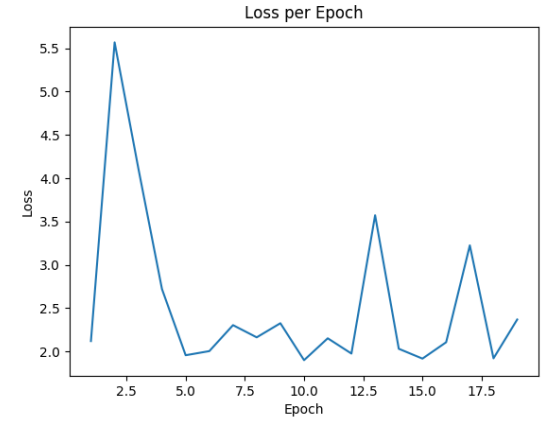
We used the same general architecture as above, but used GraphSAGE operations as the layers in the encoder using PyTorch-Geometric. The results are as follows:

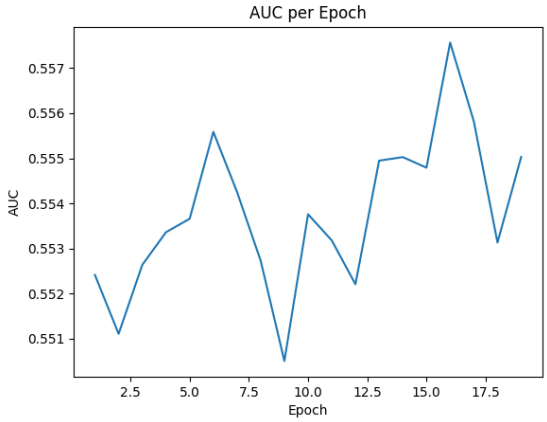
- **GAE with GraphSAGE**





- **VGAE with GraphSAGE**





E. Graph Attention Network

Finally, we used a Graph Attention Mechanism [4] as another experiment in using the Graph-VAE/GAE. We replaced the encoder's convolutional layers with a 1-head graph attention mechanism. This attention mechanism learns a node-level encoding x' as follows:

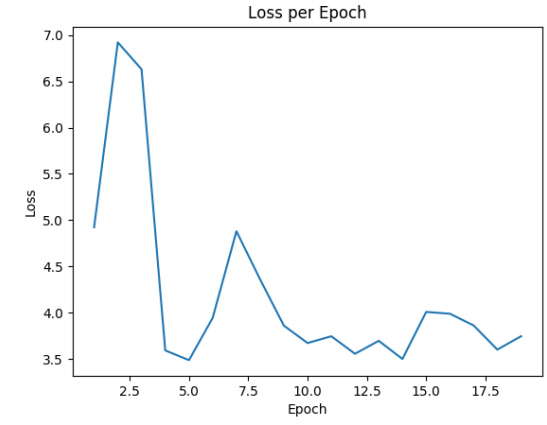
$$x'_i = \alpha_{i,i}\Theta x_i + \sum_{j \in N(i)} \alpha_{i,j}\Theta x_j$$

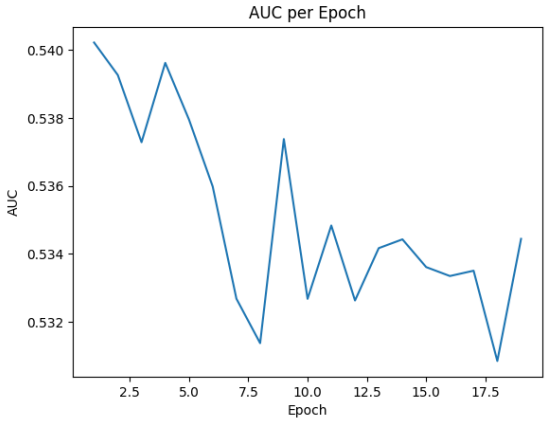
Where:

$$\alpha_{i,j} = \frac{exp(leakyReLU(a^T[\Theta x_i || \Theta x_j]))}{\sum_{k \in N(i) \cup i} exp(leakyReLU(a^T[\Theta x_i || \Theta x_k]))}$$

The results of this experiment for the GAE are as follows:

- **GAE with GAT**





(Note: Because of the number of hyper-parameters and its poor performance with the GAE, we did not do additional experiments with GAT and VGAE.)

F. Evaluation Methods

For the class project, we used the Graph-VAE/GAE's performance on the edge prediction task as an indication of the quality of the learned representation space. However, this is not truly directly indicative of latent space quality and could be improved by more direct evaluation metrics. In further work, we hope to be able to introduce several other evaluation metrics to compare generated graphs with 'real' scene graphs. Ultimately, we hope to evaluate the semantic and structural validity of the graphs generated from the latent space.

Since the goal of our experiment is to generate realistic graphs, we could eventually use the Maximum Mean Discrepancy (MMD) metric as defined in [18]. In addition, we could explore more complex techniques of comparing real and generated graphs using neural comparison methods proposed in [6]. These neural methods rely on Graph Neural Networks to embed generated graphs and Graph Matching networks to detect similarities between real and generated graph pairs. Although this was out of scope for our class project, we could also evaluate the quality of our generated scene graphs by utilizing secondary methods such as image generation or image retrieval with scene graphs [31] [30]. By looking at the relative performance of these downstream use-cases, we could gain an intuition into the performance of our graph generation architecture.

G. Synthesis and Analysis of Experimental Results

When we first trained our models on the full train set, the best auc we achieved was just slightly above 0.5. However, as described in *A. VGAE implementation*, we then began to filter out the graphs which, after randomly removing edges, ended up either disconnected in the positive case, or fully connected in the negative case. Training on partially connected graphs only, greatly increased the auc of all our models.

Overall, our best performing model was the VGAE using GCN layers in the encoder. After training for 20 epochs on the filtered training data the VGAE with GCN consistently achieved an auc above 0.6. Though the GAE with GCN eventually reached similar auc's, we found it to be much less stable. Furthermore, as we are interested in the generative capabilities of our mode, conceptually the VGAE is a better model for what we hope to achieve. Traditional GAE's are used to generate data that is similar to the input data. On the other hand, by embedding the input as a distribution rather than a point, VAE'a allow new data to be more effectively generated by sampling from the embedding space. Thus overall the VGAE is a better model for this task.

In addition to GCN layers, we also experimented with GraphSAGE and GAT layers in our encoder. Both performed less well than the GCN. Scene graphs are so small that doing the matrix multiplication required for GCN's is not that expensive, thus the sampling methods used by GraphSAGE aren't computationally necessary the way they can be for large graphs. It is also possible that the randomness in GraphSAGE accounts for its poorer performance. Again, as the graphs are so small, we want to be as strict as possible about aggregating direct neighbor feature information. For each node, by randomly sampling, even from its 1 or 2 hop neighbors, we may end up loosing what structured information we do have. We did not have a chance to run GAT with the VGAE but it had quite poor performance when run in the GAE. GAT has a larger number of hyper-parameters than the other models, so we are not prepared to make claims about its promise for scene graphs broadly speaking. However, given what time we had, we were not able to make GAT perform effectively.

H. Future Work

There are other model architectures, such as Autoregressive Generative Models and Graph U-Nets, that we think could be interesting to explored more thoroughly for this specific application.

We also could imagine future work investigating possible schemes for processing the data in such a way as to make it more amenable to models like VGAEs and U-nets. Small graphs can make edge prediction hard, as with so few edges, randomly removing some to try and predict later can result in the deletion of all important structural information about the graph. In the case of graph U-nets, the graph-pooling function defined by Gao et al. [16] suggests artificially increasing graph connectivity by randomly adding edges. Both the size of scene graphs, and the delicacy of the semantic meaning captured by scene graph structure limit the efficacy of such a technique. One thing that could be experimented with is coming up with schemes to increase graph size while maintaining semantic meaning. However, this is, perhaps, a long way off. An easier next step may be to try and incorporate edge information into the initial feature representations of nodes.

One way we were thinking of trying this would be to leverage word2vec. We could train word2vec on some corpora. We would then initialize our node features using the word embeddings corresponding to both the node and outgoing edge labels. For instance, suppose you had a scene graph consisting of a nodes "dog," "mountain" and "field," where there was an edge from "mountain" to "dog" labeled "behind" and an edge from "dog" to "field" labeled "on." (The scene is a dog on a field with a mountain in the background.) In this case, your initial embedding for "mountain" could in some way represent "mountain behind," and your embedding for "dog" could be a representation of "dog on." In this way more of the semantics of the scene would be captured by the embeddings.

References

[1] Liu, Ying, et al. "A survey of content-based image retrieval with high-level semantics." Pattern recognition 40.1 (2007): 262-282.

[2] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).

[3] Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering." Advances in neural information processing systems. 2016.

[4] Veličković, Petar, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).

[5] Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in Neural Information Processing Systems. 2017.

[6] Li, Yujia, et al. "Graph Matching Networks for Learning the Similarity of Graph Structured Objects." arXiv preprint arXiv:1904.12787 (2019).

[7] Bordes, Antoine, et al. "Translating embeddings for modeling multi-relational data." Advances in neural information processing systems. 2013.

[8] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2016.

[9] Perozzi, Bryan, Rami Al-Rfou, and Steven Skiena. "Deepwalk: Online learning of social representations." Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014.

[10] Kipf, Thomas N., and Max Welling. "Variational graph auto-encoders." arXiv preprint arXiv:1611.07308 (2016).

[11] Veličković, Petar, et al. "Graph attention networks." arXiv preprint arXiv:1710.10903 (2017).

[12] Peng, Hao, et al. "Building program vector representations for deep learning." International Conference on Knowledge Science, Engineering and Management. Springer, Cham, 2015.

[13] Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pp. 234–241. Springer, 2015.

[14] Krishna, Ranjay, et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123.1 (2017): 32-73.

[15] Hudson, Drew A., and Christopher D. Manning. "Gqa: A new dataset for real-world visual reasoning and compositional question answering." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.

[16] Gao, Hongyang, and Shuiwang Ji. "Graph U-Nets." arXiv preprint arXiv:1905.05178 (2019).

[17] Dornadula, Apoorva, et al. "Visual Relationships as Functions: Enabling Few-Shot Scene Graph Prediction." arXiv preprint arXiv:1906.04876 (2019).

[18] You, Jiaxuan, et al. "GraphRNN: Generating realistic graphs with deep auto-regressive models." International Conference on Machine Learning. 2018.

[19] Liao, Renjie, et al. "Efficient Graph Generation with Graph Recurrent Attention Networks." arXiv preprint arXiv:1910.00760 (2019).

[20] Yang, Jianwei, et al. "Graph r-cnn for scene graph generation." Proceedings of the European Conference on Computer Vision (ECCV). 2018.

[21] Haigh, Alex, Schwager, Sam, and van Paasschen, Frits. "Deep Autoregressive Models for Conditional Graph Generation". CS236 Final Project. 2018.

[22] Newell, Alejandro, and Jia Deng. "Pixels to graphs by associative embedding." Advances in neural information processing systems. 2017.
[23] Schuster, Sebastian, et al. "Generating semantically precise scene graphs from textual descriptions for improved image retrieval." Proceedings of the fourth workshop on vision and language. 2015.
[24] Xu, Danfei, et al. "Scene graph generation by iterative message passing." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. Vol. 2. 2017.
[25] van Paasschen, Frits, and Isaac Kasevich. seq2graph: A Neural Approach to Scene Graph Generation from Natural Language. https://web.stanford.edu/class/cs224n/reports/custom/15786023.pdf.
[26] Chen, Tianshui, et al. "Knowledge-Embedded Routing Network for Scene Graph Generation." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.
[27] Li, Yikang, et al. "Factorizable net: an efficient subgraph-based framework for scene graph generation." Proceedings of the European Conference on Computer Vision (ECCV). 2018.
[28] Gu, Jiuxiang, et al. "Scene graph generation with external knowledge and image reconstruction." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.
[29] Woo, Sanghyun, et al. "Linknet: Relational embedding for scene graph." Advances in Neural Information Processing Systems. 2018.
[30] Johnson, Justin, et al. "Image retrieval using scene graphs." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
[31] Johnson, Justin, Agrim Gupta, and Li Fei-Fei. "Image generation from scene graphs." arXiv preprint (2018).
[32] Wu, Jiajun, et al. "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling." Advances in neural information processing systems. 2016.
[33] Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).
[34] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.
[35] Fey, Matthias, and Jan Eric Lenssen. "Fast graph representation learning with PyTorch Geometric." arXiv preprint arXiv:1903.02428 (2019).