VARSCENE: A Deep Generative Model for Realistic Scene Graph Synthesis

Tathagat Verma
IIT Bombay

Abir De
IIT Bombay

Yateesh Agrawal
IIT Bombay

Vishwa Vinay
Adobe Research

Soumen Chakrabarti
IIT Bombay

Abstract

Scene graphs are powerful abstractions that capture relationships between objects in images by modeling objects as nodes and relationships as edges. Generation of realistic novel scene graphs has applications like scene synthesis and data augmentation for supervised learning. Existing graph generative models are predominantly targeted toward molecular graphs, leveraging the limited vocabulary of atoms and bonds and also the well-defined semantics of chemical compounds. In contrast, scene graphs have much larger object and relation vocabularies, and their semantics are latent. To address this challenge, we propose VARSCENE*, a variational autoencoder for scene graphs, which is optimized for the maximum mean discrepancy (MMD) between the ground truth scene graph distribution and distribution of the generated scene graphs.* VARSCENE *views a scene graph as a collection of star graphs and encodes it into a latent representation of the underlying stars. The decoder generates scene graphs by learning to sample the component stars and edges between them. Our experiments show that our method is able to mimic the underlying scene graph generative process more accurately than several state-of-the-art baselines.*

1. Introduction

Scene graphs [5, 16, 17] have emerged as a powerful data structure to represent images. The nodes in these graphs represent objects and their attributes, and edges capture relationships between them. They are a succinct and human-consumable summary of the content within the image. Their popularity is partially derived from the availability of rich datasets [19, 25] that contain ground-truth scene graphs associated with images. Extracting scene graphs from images continues to be a very active field of research [14, 39, 48]. The success of scene graphs can be seen in the range of visual understanding tasks that utilize them: retrieval [26, 32, 44], editing [10], question answering [40] and captioning [27].

While there has been progress in the field of image generation, the end goal of achieving rich, diverse and complex scenes is far from satisfactory [4, 20]. In the current paper, we propose the novel task of *scene graph generation* as a stepping stone to scene synthesis. Here, our goal is to learn to synthesize scene graphs based on a set of training examples of scene graphs. Our work is relevant to the theme of generating new and complex scenes, where scene graphs are often used as a means for conditioning image synthesis [17]. We decouple the generation of scene semantics (sets of objects and how they relate to each other) from its visual manifestation, and focus specifically on the first part in the current paper. Where required, we will use existing methods to convert our synthesized scene graphs to images [17, 42].

Driven by the success at text and image synthesis, there is much recent interest in deep generative models for graphs, *e.g.*, GraphVAE [33], GraphRNN [49], NeVAE [31], MolGAN [8], GRAN [24], etc. However, most of them are tailored to molecule discovery and cannot be effectively used to generate scene graphs, as we demonstrate in our experiments. Very recently, Garg et al. [11] proposed an unconditional scene graph generator, which, however, is not optimized to capture the underlying data distribution.

1.1. Present work

We propose VARSCENE, a novel variational autoencoder, specifically designed for scene graph generation. (We use graph 'generation' and 'synthesis' interchangeably.) In contrast to prior VAE-based graph generative models which maximize Evidence Lower Bound (ELBO) for model training, VARSCENE is optimized to generate graphs having minimum distribution discrepancy with the ground truth (training) graph distribution, while using its generative capacity to introduce plausible variations. We thus obtain a model from which we can sample graphs representing realistic scenes. Moreover, due to the encoder-decoder architecture, our approach facilitates conditional scene graph generation, which would otherwise not be possible [11].

Generating scene graphs with meaningful semantics is a challenging task. Designing simplistic masking functions, as for molecular graphs, does not work in this context, since the notion of underlying semantics is far more complex than in molecules. Responding to this challenge, VARSCENE views a scene graph as a collection of *stars*, where a star is comprised of a 'hub' node with its incident edges ('spokes'). Our encoder embeds the graph into a collection of latent representation vectors based on its component stars. At the other end, unlike most existing generative models [8, 31, 49], our decoder does not expand the graph by generating a node or an edge at a time, but instead by sampling stars. Specifically, it first sam-

ples stars from a trainable distribution in a sequential manner, and then connects these stars to generate semantically meaningful scene graphs. Such a model allows us to generate novel scene graphs while retaining the semantic correlations between the different components of a scene. In contrast to generative models for molecular graphs, which work with small numbers of node labels and edge labels, VARSCENE is able to generate (and be trained on) scene graphs which consist of a large number of node and edge labels.

In theory, the variational autoencoder described above should be able learn the underlying data distribution, given sufficient training data. However, due to the use of an approximate posterior and lower bound of the true objective, it may not show optimal performance in practice. In response, we further re-train the decoder to directly minimize maximum mean discrepancy (MMD) between the distributions of generated and ground truth scene graphs. Such a construction allows our method to effectively trade off between the underlying MMD and the fidelity to the prior decoder model previously obtained using ELBO maximization.

We evaluate VARSCENE on three real world datasets. Our experiments show that VARSCENE is able to mimic the underlying distribution of scene graphs more accurately than several baselines, substantially assisted by our MMD-optimized decoder.

1.2. Summary of contributions

We summarize our key contributions as follows.

Novel scene graph generative model. We provide a novel VAE specifically designed for scene graph generation. Unlike existing generative models, it does not grow the graph by sampling nodes or edges. Instead, it sequentially samples stars and embeds the scene graph with the latent representations of the stars, computed using a neural encoder. The star-based sampling approach, akin to increasing the order of Markov influences or the maximum clique size for potentials in a graphical model, gives improved approximation to the true generative process. Moreover, this approach allows our model to generate (and be trained on) scene graphs having a large number of node and edge labels.

MMD-optimized decoder design. Our decoder is designed to minimize MMD between the ground truth graph distribution and the generated graph distribution. This optimization problem effectively trades off between MMD minimization and the fidelity of the trained decoder to the base generative model, trained using a simple VAE. Such an approach allows us to capture the ground truth distribution of scene graphs more accurately than a vanilla VAE trained using ELBO maximization alone.

2. Preliminaries

In this section, we first set up notation and then provide an overview of variational autoencoders.

2.1. Notation and definitions

Scene graphs, objects, relations. Given a collection of directed scene graphs $\{G = (V, E)\}$ along with the set of objects T and the set of relations R, we denote t_u as the *object* represented by the node $u \in V$ and r_e as the *relation* represented by $e \in E$. For example, if the node pair (u, v) in the scene graph contains the semantic content: "`car` $\rightarrow$ `moving on` $\rightarrow$ `road`", then we use $t_u = \texttt{car}$, $t_v = \texttt{road}$ and $r_e = \texttt{moving on}$. We denote $\boldsymbol{t}_u$ and $\boldsymbol{r}_e$ as the feature vectors for node u and edge e respectively. Such feature vectors may be obtained by various means, depending on the application. Here we use BERT embeddings of the names of the object t_u and the relation r_e. Features may also be derived of object color and texture. Finally, we define $\text{nbr}(u)$ as the neighbors of node u.

Star and neighbor-stars. Given a graph G and a node u, the *star* s rooted at u is represented by a pair consisting of the node feature vector $\boldsymbol{t}_u$ and the multiset of features of edges incident on u. I.e., the representation of s is $s := \langle \boldsymbol{t}_u, \{\boldsymbol{r}_{(u,v)} \,|\, v \in \text{nbr}(u)\}\rangle$. We denote u as $\text{root}(s)$. Here, the star consists of only one node, *i.e.*, the central node, characterized by the underlying object and the associated open-ended edges characterized by the relations. Thus, the identity of u, or the identity or type of its neighbors are not included in the representation of s. Only the object type of u and the types of incident edges matter. Therefore, it is quite possible that two stars having different root node IDs have identical representation, i.e., $s = s'$ but $\text{root}(s) \neq \text{root}(s')$. Given a dataset of graphs D, we denote the set of stars in its "star vocabulary" as $S(D) = \{s \,|\, s \subset G, G \in D\}$.

Given a star s in graph G, we define its neighbor-stars $\mathcal{N}(s)$ as those stars whose roots are connected to the root of s via an edge. We also define $\gamma(s, s')$ as the relation type r_e of the edge e that connects $\text{root}(s)$ and $\text{root}(s')$. Formally,

$$\mathcal{N}(s) = \big\{(s', \gamma(s, s')) \,|\, (\text{root}(s), \text{root}(s')) \in E(G) \text{ for some } G \in D\big\}.$$

Thus, $\mathcal{N}(s)$ contains pairs of stars s and relation r_e, where s and s' are connected via the edge e. When clear from context, we will drop the edge type and write "$s \in \mathcal{N}(s)$" instead of "$s, \gamma \in \mathcal{N}(s)$".

Maximum mean discrepancy (MMD). We will use MMD to compute the discrepancy between the distribution of the generated graphs and the corresponding true graph distribution [13, 49]. We define MMD between distributions P and Q as:

$$\text{MMD}^2(P, Q) = \mathbb{E}_{G_1, G_2 \sim P}\, k(G_1, G_2) + \mathbb{E}_{G'_1, G'_2 \sim Q}\, k(G'_1, G'_2) - 2\mathbb{E}_{G \sim P, G \sim Q}\, k(G, G') \quad (1)$$

Here, $k(G, G')$ is the kernel induced by a suitable RKHS. We have samples D and D' drawn from p_1 and p_2 respectively. Thus, an unbiased estimator of MMD is given by $\widehat{\text{MMD}}^2(D, D') =$

$$\frac{1}{\binom{|D|}{2}} \sum_{G_1, G_2 \in D} k(G_1, G_2) + \frac{1}{\binom{|D'|}{2}} \sum_{G'_1, G'_2 \in D'} k(G'_1, G'_2) - \frac{2}{|D||D'|} \sum_{G \in D, G' \in D'} k(G, G') \quad (2)$$

In practice, we use the following RBF kernel on some representation vectors $\nu(G)$ and $\nu(G')$ of the graphs, *i.e.*,

$$k(G, G') = \exp(-||\nu(G) - \nu(G')||_2^2/2). \quad (3)$$

2.2. Overview of graph variational autoencoder

Graph VAEs are trainable generative models involving an observable graph G and its latent representations[1] Z, together with three distributions: (i) a prior distribution $p_0(Z)$ over the latent code Z, (ii) the encoder $q(Z \mid G)$ which embeds the observed graphs G into (a distribution over) the latent code $\boldsymbol{z}$, and (iii) the decoder $p(G \mid Z)$ which samples observable graphs G based on the latent representation Z. Computation of the log-likelihood of observables under a VAE model requires marginalization over the distribution of Z, which is usually intractable. Therefore, we typically maximize the evidence lower bound (ELBO) on the log-likelihood:

$$\sum_{G \in D} \Big[\mathbb{E}_{Z \sim q(.|G)} \big[\log p(G|Z) \big] + KL\big(q(Z|G) || p_0(Z)\big) \Big] \quad (4)$$

The encoder $q(Z \mid G)$ is intended to approximate the true posterior of Z. Therefore, the quality of the encoder, and thus, the tightness of the lower bound, depends on the expressivity of q. Typically, it is modeled using a universal distribution approximator, parameterized on G.

3. VARSCENE model

Here we develop VARSCENE in full detail, formulating the underlying MMD optimization problem and presenting its different components and design decisions.

3.1. Overview

Our goal is to design a variational autoencoder that is able to synthesize realistic scene graphs in both unconditional settings (given a corpus of real scene graphs), and conditional settings (given a corpus and also a specific scene graph as a starting point). While we want the synthetic graphs to exhibit stochastic variation beyond training graphs, we want the distribution of various graph properties to remain close to those measured on the training sample. The cornerstone of our approach is to build the decoder that minimizes the MMD between these two distributions, rather than only maximizing ELBO (4).

Given a hypothesis class $\mathcal{P}$, we wish to learn a generative distribution $p^{\text{MMD}}(G) \in \mathcal{P}$ which minimizes MMD between the distribution of real scene graphs and the distribution of the generated graphs. However, training such a generative model without any inductive bias is unlikely to succeed. To tackle this problem, we first train a base variational autoencoder (p_0, q, p) on the training scene graphs. While the trained decoder p is not optimized for minimum MMD, it provides an approximate generative model to start with. We next use this trained decoder p to guide the training of our MMD optimized generative model p^{MMD}. To this end, we design an optimization problem around p^{MMD} which keeps it close to the pre-trained decoder p, while also minimizing MMD with respect to the training data. Finally, we develop a gradient based algorithm to search for p^{MMD}.

[1] Z is usually a set of latent codes associated with embedding vectors of nodes, edges or substructures.

3.2. Design of MMD-optimized decoder p^{MMD}

Learning p^{MMD} given the true generator p_{true}. If we are given the true generator p_{true}, then, in principle, we can directly estimate a generative distribution $p^{\text{MMD}} \in \mathcal{P}$ over scene graphs, which minimizes MMD between p^{MMD} and the true distribution:

$$\underset{p^{\text{MMD}} \in \mathcal{P}}{\text{minimize}} \ \text{MMD}(p_{\text{true}}, p^{\text{MMD}}). \quad (5)$$

The above setup faces two bottlenecks:

(i) It does not have access to the true distribution p_{true}. It can only use the observed scene graphs present in the training dataset.

(ii) Specifying $\mathcal{P}$ without any prior knowledge about the generative process is difficult in practice.

Learning p^{MMD} given observed graphs. Responding to the above challenges, we approximate (5) through several intermediate steps. First, we use the observed scene graphs to estimate $\widehat{MMD}$, as we cannot access the true graph distribution p_{true}. Specifically, given D, a set of observed scene graphs, we replace the objective in the optimization (5) with its sample estimate defined in Eq. (2) as follows:

$$\underset{p^{\text{MMD}} \in \mathcal{P}}{\text{minimize}} \ \mathbb{E}_{D' \underset{iid}{\sim} p^{\text{MMD}}(\cdot)} \widehat{\text{MMD}}(D, D') \quad (6)$$

However, absence of any prior knowledge of the underlying generative process $\mathcal{P}$ makes it difficult to search for p^{MMD}. Now, suppose that we have a trained VAE model $(p_0, q_{\widehat{\phi}}, p_{\widehat{\theta}})$ modeled by neural networks with estimated parameters $\hat{\phi}$ and $\hat{\theta}$, where the encoder $q_{\widehat{\phi}}(\cdot \mid G)$ embeds the graph into a latent representation Z, the decoder $p_{\widehat{\theta}}(\cdot \mid Z)$ generates a graph G conditioned on Z and p_0 is a prior distribution on Z. We can use this trained model to guide the training of p^{MMD}, where we use the same neural network $p_{\widehat{\theta}}$ to parameterize $p^{\text{MMD}} \approx p_\theta^{\text{MMD}}$ with a new parameter vector θ. To this end, given a trained VAE model $(p_0, q_{\widehat{\phi}}, p_{\widehat{\theta}})$ we learn p_θ which minimizes the required MMD, while penalizing its KL divergence with respect to the trained decoder $p_{\widehat{\theta}}$, *i.e.*, we solve:

$$\underset{\theta \in \Theta}{\text{minimize}} \ \mathbb{E}_Z \Bigg[\underset{D(Z) \underset{iid}{\sim} p_\theta^{\text{MMD}}(\cdot \mid Z)}{\mathbb{E}} \widehat{\text{MMD}}\Big(D, D(Z)\Big) + \rho \, \text{KL}\Big(p_\theta^{\text{MMD}}(\cdot \mid Z) \Big\| p_{\widehat{\theta}}(\cdot \mid Z)\Big) \Bigg], \quad (7)$$

where $D(Z)$ is a set of generated graphs by p_θ^{MMD} conditioned on the latent representations Z. Here, $p_{\widehat{\theta}}$ and p_θ^{MMD} share the same parameterized class of generative models $\mathcal{P}(\Theta)$. In the outer expectation, Z can be drawn from the prior distribution p_0 for unconditional graph generation or from the trained encoder q_ϕ for conditional graph generation. Moreover, $\rho > 0$ is a tunable coefficient for the KL regularizer which allows us to trade off between minimizing MMD and the fidelity of p^{MMD} to the original decoder p. Such a KL divergence based regularization allows p_θ^{MMD} to generate plausible scene graphs with meaningful semantics.

Apart from VAEs, such regularization using KL divergence with respect to a prior distribution has also been used in reinforcement learning [41], social welfare based

control [36], property oriented molecule generation [31], etc. However, our key goal here is MMD optimization for scene graph generation, where we must steer the property of the entire set of generated graphs, whereas the above tasks are concerned with property of one instance, e.g., generating molecule with a desired property, re-ranking tweets for misinformation mitigation, etc.

Computation of $\widehat{\text{MMD}}(D, \{G\})$. As suggested by Eq. (2) and (3) , computation of $\widehat{\text{MMD}}(D, \{G\})$ requires us to featurize G into $\nu(G)$. Graph kernels [43] provide some guidance to our choice of ν, but judging whether a synthetic graph is "close to natural" requires the comparison of a diverse bouquet of graph properties, covering complex interactions between topology, node and edge labels. Because our unit of graph generation is a star, we will focus, for the most part, on a definition of $\nu(G)$ in terms of the number of different stars present in the graph. (For evaluation, though, we will include other important properties.) We gather the universe of stars $S(D)$ present in the dataset D, defined as

$$\nu_s(G) = \#\text{ occurrences of } s \text{ in } G \tag{8}$$

and then compute $\nu(G) = \big[\nu_s(G) : s \in S(D)\big]$, which is then used to compute the required MMD using Eq. (3).

Gradient computation for solving (7). We solve optimization (7) using a gradient descent algorithm on the finite sample estimate of the underlying objective, *i.e.*,

$$C_\theta(\{G\}) = \mathbb{E}_Z\Bigg[\widehat{\text{MMD}}(D, D(Z)) + \frac{\rho}{|D(Z)|}\sum_{G\in D(Z)} \log \frac{p_\theta^{\text{MMD}}(G \mid Z)}{p_{\widehat{\theta}}(G \mid Z)}\Bigg]. \tag{9}$$

Computing the gradient of such a quantity may seem difficult, since the samples $\{G\}$ are drawn from the distribution itself which is being trained. To circumvent this problem, we use the log-derivative trick [45] to estimate the gradient $\nabla_\theta C_\theta(\{G\})$ as follows:

$$\sum_{G\in D(Z)} \widehat{\text{MMD}}(D, D(Z)) \nabla_\theta \log p_\theta^{\text{MMD}}(G \mid Z) + \frac{\rho}{|D(Z)|}\sum_{G\in D(Z)}\Bigg[\nabla_\theta \log p_\theta^{\text{MMD}}(G \mid Z) \log \frac{p_\theta^{\text{MMD}}(G \mid Z)}{p_{\widehat{\theta}}(G \mid Z)} + \nabla_\theta \log p_\theta^{\text{MMD}}(G \mid Z)\Bigg], \tag{10}$$

which can be rewritten as

$$\nabla_\theta\Bigg[\sum_{G\in D(Z)} \widehat{\text{MMD}}(D, D(Z)) \log p_\theta^{\text{MMD}}(G \mid Z) + \frac{\rho}{|D(Z)|}\sum_{G\in D(Z)}\Big(\frac{1}{2}\log^2 \frac{p_\theta^{\text{MMD}}(G \mid Z)}{p_{\widehat{\theta}}(G \mid Z)} + \log p_\theta^{\text{MMD}}(G \mid Z)\Big)\Bigg], \tag{11}$$

where the second term on the RHS of the last equality follows from the fact that $\nabla_\bullet \frac{1}{2}(\log(f(\bullet)/a))^2 = \nabla_\bullet \log(f(\bullet)) \log(f(\bullet)/a)$. The term within the above gradient can be viewed as a pseudo loss function which can be easily coded in any ML library to solve the optimization in Eq. (7). In particular, the parameter update step becomes:

$$\theta_{i+1} = \theta_i - \text{learning\_rate}\, \nabla_\theta C_\theta(\{G\})\Big|_{\theta=\theta_i} \tag{12}$$

where graphs $G \in D(Z)$ are drawn from distribution p_{θ_i}.

3.3. Neural architecture of VARSCENE

In this section, we present the neural architecture of the various components of VARSCENE, beginning with a brief outline. Appendix D gives more details.

Outline. VARSCENE consists of a base variational autoencoder $(p_0, q_\theta, p_\theta)$ and the final generative distribution p_θ^{MMD}. The encoder $q_\theta(\cdot)$ views a graph as a set of stars $\{s_0, s_1, \dots\}$, where we call the star s_0 to be the pivot star. Then, we consider various stars $\{s_i \mid \text{DISTANCE}(\text{root}(s_0), \text{root}(s_i)) = \Delta\}$ lying at a given number of hops Δ from the pivot star s_0, and obtain an aggregated representation vector $\boldsymbol{z}_\Delta$ over these stars. Thus, our encoder q_ϕ encodes the stars into the corresponding latent representations $Z = \{\boldsymbol{z}_0, \dots, \boldsymbol{z}_{\Delta_{\max}}\}$, where $\Delta_{\max}$ is the maximum distance of any star from the pivot s_0. The corresponding decoder $p_{\widehat{\theta}}(\cdot)$ generates stars in a sequential manner by sampling from a softmax distribution conditioned on the latent representations Z. Given a set of training graphs along with their object and relation types, *i.e.*, $D = \{G\}$ along with the object types $\{T\}$ and the relation types $\{R\}$, VARSCENE first learns the encoder $q_\phi(\cdot)$ and decoder $p_\theta(\cdot)$ by maximizing ELBO (evidence lower bound) of the likelihood function. More specifically, VARSCENE consists of four components:

1. Prior: $p_0(Z)$ with $\Delta_{\max} \sim \text{Poisson}(\lambda)$
2. Encoder: $q_\phi(Z \mid G, T = \{\boldsymbol{t}_u\}, R = \{\boldsymbol{r}_e\})$
3. Base decoder: $p_\theta(\{s_i\}, \{\gamma_{ij}\} \mid Z)$
4. MMD-optimized decoder: $p_\theta^{\text{MMD}}(\{s_i\}, \{\gamma_{ij}\} \mid Z)$

where, $Z = \{\boldsymbol{z}_0, ..., \boldsymbol{z}_{\Delta_{\max}}\}$ is the set of star representations at different distances and $\gamma_{ij} = \gamma(s_i, s_j)$.

Encoder q_ϕ. Given a graph $G = (V, E)$ along with the object types $T = \{t_u\}$ associated with the nodes and the relation types $R = \{r_e\}$, the encoder q_ϕ aims to characterize the scene graphs as a collection of star graphs $\{s_i\}$ and represent the graph as a collection of embeddings of these stars, *i.e.*, $Z = \{\boldsymbol{z}_\Delta \mid 0 \le \Delta \le \Delta_{\max}\}$. Such an approach is likely to preserve the spatial semantics of the visual content in a scene graph more effectively than existing graph generative models [8, 23, 31] which represent a graph as a collection of node embeddings. In VARSCENE, we compute the graph representation Z in two steps. First, we compute the node and edge embeddings using a graph neural network (GNN) and then combine these to compute star embeddings.

Node and edge embeddings. Given a graph G and the object types $T = \{\boldsymbol{t}_u\}$ and $R = \{\boldsymbol{r}_e\}$, we compute the representations of the nodes $\boldsymbol{x}_u$ and edges $\boldsymbol{x}_e$ using a GNN proposed by Gilmer et al. [12]. More specifically, we start with the initial node representation $\boldsymbol{x}_u(0)$ which transforms the feature vector $\boldsymbol{t}_u$ of the object type associated with node u using a neural network F_ϕ^0.

$$\boldsymbol{x}_u(0) = F_\phi^0(\boldsymbol{t}_u) \tag{13}$$

Then, given a depth limit K, we aggregate structural information from $k = 1, ..., K$ hops from each node into an embedding vector in a recursive manner. Formally:

$$\boldsymbol{x}_{(u,v)}(k-1) = F_\phi^{\text{edge}}\big(\boldsymbol{x}_u(k-1), \boldsymbol{x}_v(k-1), \boldsymbol{r}_{(u,v)}\big) \tag{14}$$

$$\overline{\boldsymbol{x}}_u(k-1) = F_\phi^{\text{agg}}\big(\{\boldsymbol{x}_{(u,v)}(k-1) \mid v \in \text{nbr}(u)\}\big) \tag{15}$$

$$\boldsymbol{x}_u(k) = F_\phi^{\text{node}}\big(\boldsymbol{x}_u(k-1),\ \overline{\boldsymbol{x}}_u(k-1)\big). \tag{16}$$

Here, F_ϕ^{edge} and F_ϕ^{node} are modeled as universal approximator neural networks; and, F_ϕ^{agg} is a symmetric aggregator function.

Star embedding. Given S as the set of stars in G, we represent each star $s \in S$ with depth-K embeddings:

$$\boldsymbol{h}_s = F_\phi^S(\boldsymbol{x}_{\text{root}(s)}, \{\boldsymbol{x}_{(\text{root}(s),v)} \mid v \in \text{nbr}(\text{root}(s))\}) \quad (17)$$

Next, we generate the latent representations $\boldsymbol{\zeta}_s$ from a normal distribution, which parameterizes the mean and the variance using neural networks μ_ϕ and σ_ϕ, *i.e.*,

$$\boldsymbol{\zeta}_s \sim \text{NORMAL}(\mu_\phi(\boldsymbol{h}_s), \sigma_\phi^2(\boldsymbol{h}_s)). \quad (18)$$

In theory, $\boldsymbol{\zeta}_s$ encodes the information about the substructures up to K hops away from s and thus the collection of the representations $\{\boldsymbol{\zeta}_s\}$ should be able to accurately characterize the graph. However, this may not be true in practice, as it does not explicitly specify the relative positions of the stars. To ameliorate this limitation, we first randomly draw the pivot star s_0 from set of all stars S and compute the distances $\{\Delta_s\}$ of all stars $s \in S$ from s_0. Next, we aggregate $\{\boldsymbol{\zeta}_s \mid \Delta_s = \Delta\}$ into a latent vector $\boldsymbol{z}_\Delta$ as follows:

$$\boldsymbol{z}_\Delta = F_\phi^z(\{\boldsymbol{\zeta}_s \mid \Delta_s = \Delta\}) \text{ for } 0 \leq \Delta \leq \Delta_{\max} \quad (19)$$

where $\Delta_{\max}$ is the maximum possible distance from s_0 to any star $s \in S$. Finally, we represent the graph as $Z = \{\boldsymbol{z}_0, ..., \boldsymbol{z}_{\Delta_{\max}}\}$.

Decoders p_θ and p_θ^{MMD}. The decoders p_θ and p_θ^{MMD} share the same neural network structure. They take the latent representations $Z = \{\boldsymbol{z}_0, ..., \boldsymbol{z}_{\Delta_{\max}}\}$ as input and generate a new graph G. To do so, they incrementally draw new stars s_j from a database of stars $S(D)$ and connect them with previously sampled stars $s_i, i \leq j-1$. The decoders first sample a star s_0 and set $\Delta = 0$. Then, at the step $\Delta \geq 1$, they select the stars $\{s_i \mid \text{DISTANCE}(\text{root}(s_0), \text{root}(s_i)) = \Delta - 1\}$ having distance $\Delta - 1$ from s_0 and connect one of them with another star s_j using one of the open edges e, *i.e.*, $\gamma(s_i, s_j) = r_e$. In this context, s_j is either drawn from the database of stars $S(D)$ or chosen from one of the existing stars in the graph.

To connect a new star s_j at an open edge e of a star s_i lying at a distance Δ from s_0, our decoders first featurize all the potential stars $s := \langle \boldsymbol{t}_u, \{\boldsymbol{r}_{(u,v)} \mid v \in \text{nbr}(u)\}\rangle$ as $\boldsymbol{s} = [\boldsymbol{t}_u, \sum_v \boldsymbol{r}_{(u,v)}]$; then represent the pair $[\boldsymbol{s}, \boldsymbol{z}_\Delta]$ using a logit; and finally, feeds the logit into a softmax distribution. This softmax distribution is used to sample the new star s'. Formally, given the snapshot of the scene graph G_Δ at step Δ, we have:

$$p_\theta(G \mid Z) = p_\theta\left(\{s_i\}, \{\gamma(s_i, s_j)\} \mid Z\right)$$
$$= \prod_{\Delta=1}^{\Delta_{\max}} \prod_{\gamma(s_i,s_j):\Delta_{s_i}=\Delta} p_\theta(s_j, \gamma(s_i, s_j) \mid G_\Delta, Z) \quad (20)$$

where, $\Delta_{s_i} = \text{DISTANCE}(\text{root}(s_0), \text{root}(s_i))$. Moreover, we model $p_\theta(s_j, \gamma(s_i, s_j) = r \mid G_\Delta, Z)$ as,

$$\frac{\text{MASK}(s_i, s_j, r)[\![s_j \notin G_\Delta]\!] \exp(M_\theta(\boldsymbol{s_j}, \boldsymbol{z}_\Delta))}{\sum_{s \in S(D) \setminus G_\Delta} \text{MASK}(s, s_j, r) \exp(M_\theta(\boldsymbol{s}, \boldsymbol{z}_\Delta))}$$
$$+ \frac{\text{MASK}(s_i, s_j, r)[\![s_j \in G_\Delta]\!]}{|\{s : \text{MASK}(s, s_j, r) = 1, s \in G_\Delta\}|}. \quad (21)$$

Here, M_θ is a neural network and the masking function MASK aims to ensure that generated scene graphs are semantically meaningful. Specifically, $\text{MASK}(s, s', r) = 1$ encodes that s' is connected to s *via the relation* r *in at least one scene graph in the dataset* D, *i.e.*, $\text{MASK}(s, s', r) = [\![(s', r) \in \mathcal{N}(s)]\!]$. The first term computes the probability when s_j is not present in G_Δ and is drawn from $S(D)$. The second term computes the probability when s_j is already present in G_Δ.

Prior distribution p_0. $p_0(\boldsymbol{z}_0, .., \boldsymbol{z}_{\Delta_{\max}})$ is modeled using a Normal distribution, *i.e.*, $p_0(\boldsymbol{z}_0, .., \boldsymbol{z}_{\Delta_{\max}}) = \text{NORMAL}(0, \mathbb{I}_{\Delta_{\max}})$. Finally, we maximize $\text{ELBO}(q_\phi, p_\theta)$ defined in Eq. (4) with respect to ϕ and θ to obtain $\widehat{\phi}$ and $\widehat{\theta}$.

Dataset	$\lvert D\rvert$	Folds	$\mathbb{E}[\lvert V\rvert]$	$\mathbb{E}[\lvert E\rvert]$	$\lvert T\rvert$	$\lvert R\rvert$
Visual Genome	110000	81:9:9	3.23	2.31	16943	8411
Small-sized Visual Genome	124854	55:12:32	5.31	4.87	150	50
Visual Relationship Detection	7721	74:12:12	4.29	3.78	100	70

Table 1. Dataset statistics where Folds $= |D_{\text{tr}}| : |D_{\text{dev}}| : |D_{\text{test}}|$.

4. Experiments

In this section, we provide a comprehensive evaluation of VARSCENE, addressing the following research questions: **RQ1:** How does VARSCENE compare against state-of-the-art graph generators in terms of its ability to capture the true graph distribution? **RQ2:** Does the MMD-optimized decoder p_θ^{MMD} provide any performance gain as compared to the the base decoder p_θ? **RQ3:** How does the quality of the images created from scene graphs generated by VARSCENE compare against the quality of images created via state-of-the-art graph generators? **RQ4:** What do the generated scene graphs look like? Does the variational code space ensure smooth transitions between scene graphs? Appendix F contains additional experiments.

4.1. Experimental setup

Datasets. We use three datasets: (i) Visual Genome (VG) [19], (ii) Small-sized Visual Genome (SVG) [46] and (iii) Visual Relationship Detection (VRD) [25]. Visual Genome has a total of 16493 object categories and 8411 relationship categories. VARSCENE can deal with this, but some recent systems report results on SVG, derived from VG by reducing the number of object and relationship categories to 150 and 50 respectively, by combining related labels and removing poor-quality labels [11, 46]. Table 1 summarizes details about these datasets. Appendix E provides additional details.

State-of-the-art competitors. We compare VARSCENE against several competitive graph generative methods: two generative models which were developed for molecule synthesis, *viz.*, (1) DeepGMG [23], (2) MolGAN [8]; two domain-agnostic graph generative models, *viz.*, (3) GraphRNN [49], (4) GraphGen [13], and one graph generative model specifically aimed at synthesizing scene graphs, similar to our setup, *i.e.*, (5) SceneGen [11].

Training, validation and testing. Given a dataset of scene graphs D, we split them in training (D_{tr}), validation (D_{dev}) and test (D_{test}) folds, where the exact size of these folds $|D_\bullet|$ varies across datasets due to their varying sizes (see Table 1). We first train the encoder q_ϕ and the base decoder p_θ using the training set D_{tr} and then train the MMD optimized decoder p_θ^{MMD} using the validation set

D_{dev}. Finally, p_θ^{MMD} is evaluated using D_{test}.

Evaluation metrics. We evaluate the quality of a generative model by measuring the similarity between the graphs generated by the trained model and the graphs in the test set. However, unlike normed-distance in Euclidean feature space, a single similarity measure cannot capture the similarity between two graphs in terms of all the properties. Hence, we resort to several similarity metrics.

Star-Sim: The *star distribution similarity* is measured in terms of the cosine similarity between the distributions of the stars present in the test and generated graphs, *i.e.*, $\cos\left(\mathbb{E}_{G'\sim p_\theta^{\text{MMD}}}\nu(G'), \mathbb{E}_{G\sim D_{\text{test}}}\nu(G)\right)$ with $\nu(G) = [\nu_s(G)]$ where $\nu_s(G)$ is the number of occurrences of star s in G, as defined in Eq. (8).

Edge-bigram-Sim: The *edge bigram similarity* is measured in terms of the cosine similarity between the distributions of the edge bigrams of the form of the tuple $e_b = \langle r_{(\bullet,u)}, t_u, r_{(u,\bullet)}\rangle$ across the test and generated graphs, *i.e.*, $\cos\left(\mathbb{E}_{G'\sim p_\theta^{\text{MMD}}}\nu(G'), \mathbb{E}_{G\sim D_{\text{test}}}\nu(G)\right)$ where $\nu(G) = [\nu_{e_b}(G)]$, $\nu_{e_b}(G)$ = # occurrences of edge bigram e_b in G.

Node-bigram-Sim: The *node bigram similarity* is measured in terms of the cosine similarity between the distributions of the node bigrams of the form of the tuple $n_b = \langle t_u, r_{(u,v)}, t_v\rangle$ across the test and generated graphs, *i.e.*, $\cos\left(\mathbb{E}_{G'\sim p_\theta^{\text{MMD}}}\nu(G'), \mathbb{E}_{G\sim D_{\text{test}}}\nu(G)\right)$ where $\nu(G) = [\nu_{n_b}(G)]$, $\nu_{n_b}(G)$ being the number of occurrences of node bigram n_b in G.

SP-Kernel: *Shortest path kernel* [43].

WL-Kernel: *Weisfeiler-Lehman kernel* [43].

NSPD-Kernel: *Neighborhood pairwise distance graph kernel* [6].

In all kernel-based metrics, we compute the similarity as $\mathbb{E}_{G'\sim p_\theta^{\text{MMD}}, G\sim D_{\text{test}}} K(G', G)$. Note that most of these measure the similarity in terms of structures as well as object and relation types. In particular, SP-Kernel and WL-Kernel take object types into account and ignore relationship types, whereas NSPD-Kernel incorporates both object and relationship types.

4.2. Results

Comparison with baseline graph generators. We first address research question **RQ1** by comparing VARSCENE against baselines, in terms of the metrics defined above. We present results for two variants of VARSCENE, *viz.*, unconditional generative model VARSCENE$^{\text{unc}}$ where $Z \sim p_0(\cdot)$ and conditional generative model VARSCENE$^{\text{cond}}$ where $Z \sim q_\phi(\cdot \mid G)$ with $G \in D_{\text{tr}}$, Table 2 summarizes the results. We make the following observations.

(1) VARSCENE outperforms all the competitors in terms of Star-Sim, SP-Kernel and WL-Kernel with a significant boost. There is no consistent winner between VARSCENE$^{\text{unc}}$ and VARSCENE$^{\text{cond}}$. Since VARSCENE$^{\text{cond}}$ generates graphs which are structurally similar to the existing training graphs, one may expect that it is likely to mimic the underlying distribution more closely than VARSCENE$^{\text{unc}}$. However, this need not always be the case, if there is a large empirical distribution shift between finite training and test folds. VARSCENE$^{\text{unc}}$ often captures such drift, as it does not look into the training graphs during new graph generation.

(2) SceneGen outperforms the other baselines by a substantial margin in a majority of the cases. Since it is specifically aimed at scene graph generation, it is able to capture the underlying scene graph distribution more effectively than the baselines which are predominantly domain agnostic or designed for molecule synthesis.

(3) Among the other baselines, DeepGMG shows good performance in terms of WL-Kernel or NSPD-Kernel across Visual Genome and Visual Relationship Detection datasets. Although it was primarily applied for molecule generation, its design choice is quite domain agnostic. The performance of MolGAN is extremely poor due to its design choices that are specifically aimed at molecule generation.

(4) While VARSCENE outperforms SceneGen in terms of edge and node bigram similarity measures for the Visual Genome dataset, SceneGen appears to be the best performer for the two other datasets. However, this apparent superior performance of SceneGen may be misleading — careful investigation revealed that SceneGen generates many isolated nodes whose divergence from the test distribution cannot be captured by bigram similarity.

To shed light on the observation noted in item **(4)**, we next introduce a dummy relation type `NULL` and assign this relation between any isolated node and every other node in a graph. Thus, the set of edge bigrams now include $\langle r, t, r'\rangle$ where r and r' may also assume the `NULL` type in addition to the existing legitimate relation types R. Similarly, the set of node bigrams now include $\langle t, r, t'\rangle$ where r may assume the `NULL` type. Table 3 summarizes the effect of this augmented set of relation types, which shows that SceneGen actually performs not as well as VARSCENE$^{\text{cond}}$.

Effect of MMD-optimized decoder. Next, we address research question **RQ2**. Specifically, we compare the quality of the scene graphs generated by our MMD optimized decoder p_θ^{MMD} with the base decoder p_θ. Table 4 summarizes the results, which shows that our MMD-optimized decoder is able to mimic the true distribution of stars, as well as edge bi-grams, more accurately than the base decoder. Other metrics are reported in Appendix F.

Image quality. Thus far, we have assessed the quality of the generated graphs, rather than the images that might be generated from them. Here, we address research question **RQ3** by evaluating the quality of the images corresponding to the synthesized scene graphs. Specifically, we use `sg2im`[2], a scene graph to image generation system [17] that comes pre-trained on the Small-sized Visual Genome dataset. Similar to Garg et al. [11], we evaluate the quality of the images using Fréchet Inception Distance [15], Inception Score [30], Precision and Recall [29]. Both Inception Score (IS) and Fréchet Inception Distance (FID) aim to provide proxies for qualitative human evaluation. They assess the images from two perspectives: (a) the quality of the images via the output of an image classification model like InceptionV3 and (b) diversity across

[2] https://github.com/google/sg2im

Model	Star-Sim	Edge-bigram-Sim	Node-bigram-Sim	SP-Kernel	WL-Kernel	NSPD-Kernel
Visual Genome (VG)						
DeepGMG	0.6996	0.4668	0.1579	0.0137	0.0994	0.0144
MolGAN	0.0000	0.0000	0.0000	0.0002	0.0439	0.0131
GraphGen	0.6696	0.3760	0.1145	0.0037	0.0378	0.0126
GraphRNN	0.6352	0.0000	0.0332	0.0003	0.0339	0.0121
SceneGen	0.7378	0.5050	0.3275	0.0276	0.0804	**0.0159**
VARSCENE$^{\text{unc}}$	0.5963	0.4576	0.4055	**0.2293**	**0.1176**	0.0147
VARSCENE$^{\text{cond}}$	**0.8660**	**0.5268**	**0.6226**	0.0840	0.0779	0.0135
Small-sized Visual Genome (SVG)						
DeepGMG	0.8097	0.6401	0.4961	0.1092	0.3589	0.0538
MolGAN	0.0000	0.0000	0.0000	0.0496	0.4084	0.0879
GraphGen	0.5223	0.6461	0.3713	0.0545	0.2117	0.0432
GraphRNN	0.2590	0.0775	0.0920	0.3833	0.5991	**0.0705**
SceneGen	0.8644	**0.8828**	**0.9311**	0.6876	0.5480	0.0663
VARSCENE$^{\text{unc}}$	**0.9235**	0.7049	0.8335	**1.0046**	**0.6556**	0.0685
VARSCENE$^{\text{cond}}$	0.9182	0.6964	0.8111	0.9621	0.6417	0.0633
Visual Relationship Detection (VRD)						
DeepGMG	0.7459	0.7338	0.6035	0.9977	1.4125	0.2038
MolGAN	0.0000	0.0000	0.0001	0.0149	0.9721	0.2121
GraphGen	0.6478	0.7586	0.6431	0.3184	0.7964	0.1770
GraphRNN	0.5455	0.2941	0.7163	0.2125	0.7639	0.1888
SceneGen	0.8112	**0.9486**	**0.9552**	0.6069	1.1274	0.2130
VARSCENE$^{\text{unc}}$	**0.9194**	0.9374	0.9403	1.0327	1.5619	**0.2301**
VARSCENE$^{\text{cond}}$	0.9140	0.9372	0.9377	**1.4588**	**1.9218**	0.2275

Table 2. Performance of different graph generative models which include two variants of our model, *i.e.*, VARSCENE$^{\text{unc}}$ (unconditional generation, $Z \sim p_0(\cdot)$), VARSCENE$^{\text{cond}}$ (conditional generation, $Z \sim q_\phi(\cdot \mid G)$), and the baselines, *viz.*, DeepGMG [23], MolGAN [8], GraphGen [13], GraphRNN [49], SceneGen [11], measured in terms of cosine similarity between stars (Star-Sim), edge bigrams (Edge-bigram-Sim), node bigram (Node-bigram-Sim), Shortest path kernel (SP-Kernel) [43], Weisfeiler Lehman Kernel (WL-Kernel) [43], Neighborhood Subgraph Pairwise Distance Kernel (NSPD-Kernel) [6]. In all cases, the encoder q_ϕ and the base decoder p_θ were trained using the training set D_{tr} and the MMD optimized decoder was obtained using the validation set D_{dev}. In all cases, the weight of the KL divergence term in Eq. (7) was set as $\rho = 1000$. Numbers in **bold** (underline) indicate the (second) best performer.

	VG		SVG		VRD	
	Edge	Node	Edge	Node	Edge	Node
VARSCENE$^{\text{cond}}$	**0.5268**	**0.6226**	**0.6964**	0.8111	**0.9372**	**0.9377**
SceneGen, NULL$\in R$	0.0890	0.1121	0.2661	**0.8207**	0.4598	0.8873

Table 3. Performance measured in terms of Edge-bigram-Sim ('Edge') and Node-bigram-Sim ('Node') for VARSCENE$^{\text{cond}}$ and the variant of SceneGen where we introduce a new relation type NULL between an isolated node and any other node. Numbers in bold indicate the best performer. With this modification, SceneGen shows rather poor performance compared to Table 2, where it is not penalized for isolated nodes. Real graphs in our data sets do not have isolated nodes.

	VG		SVG		VRD	
	Star	Edge	Star	Edge	Star	Edge
p_θ^{MMD}	**0.8660**	**0.5268**	**0.9182**	**0.6964**	**0.9140**	**0.9372**
p_θ	0.5867	0.2588	0.7120	0.4195	0.8988	0.9339

Table 4. Performance measured in terms of Star-Sim ('Star') and Edge-bigram-Sim ('Edge') for the scene graphs generated by the MMD optimized decoder p_θ^{MMD} and the base decoder p_θ. Numbers in bold indicate the best performer. We observe that p_θ^{MMD} outperforms p_θ. In all cases, we used conditional graph generation.

the set of the images. We formally define FID, IS, Precision and Recall in Appendix E. Table 5 summarizes the results for SVG, which shows that, (1) the images corresponding to the scene graphs provided by the variants of VARSCENE have substantially better quality than all other baselines; (2) DeepGMG outperforms the other baselines in majority of the cases.

Visualization of generated scene graphs. Finally, we address research question **RQ4**, where we visualize the

Model	FID (↓)	IS (↑)	Precision (↑)	Recall (↑)
DeepGMG	9.8344	4.3566	0.9891	0.9800
MolGAN	240.3760	1.1707	0.0137	0.0986
GraphGen	13.2806	4.2802	0.9780	0.9607
GraphRNN	18.8156	4.6561	0.9432	0.9707
SceneGen	19.2634	4.0283	0.9230	0.9513
VARSCENE$^{\text{unc}}$	6.8591	**5.1505**	**0.9894**	0.9872
VARSCENE$^{\text{cond}}$	**6.0195**	5.0211	**0.9894**	**0.9874**

Table 5. Evaluation of various generative models by assessing the quality of the the images obtained from the corresponding scene graphs. Performance is measured in terms of Fréchet Inception Distance [15], Inception Score [30] and Precision & Recall [29]. Numbers in **bold** (underline) show the best (second best) performer. ↓ means smaller is better; ↑ means larger is better.

graphs G' generated by conditioning on an existing graph G. Specifically, we draw $G' \sim p_\theta^{\text{MMD}}(\bullet \mid Z)$, where $Z \sim q_\phi(\bullet \mid G)$. Figure 6 provides some samples, which show that VARSCENE is able to generate graphs which provides similar images to the image corresponding to scene graph $\boxed{G}$ used for conditioning.

5. Conclusion

We presented VARSCENE, a variational autoencoder tailored to synthesize scene graphs from a seed set of 'gold' graphs. VARSCENE uses a novel sampling vocabulary of *stars* annotated with node and edge types from large vocabularies typical in image collections. It directly minimizes distributional discrepancies for features observed in the gold and generated graphs. These two strengths enable VARSCENE to outperform recent graph

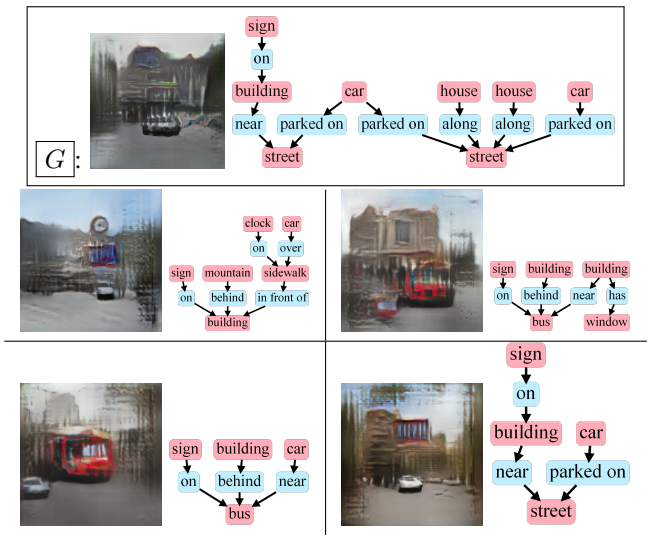

Figure 6. Four scene graphs and the corresponding images, generated using $G \sim p_\theta^{\text{MMD}}(\bullet \mid Z)$, where $Z \sim q_\phi(\bullet \mid \boxed{G})$. Here, $\boxed{G}$ is the graph used for conditioning, which is chosen from Small-sized Visual Genome dataset. The images corresponding to the scene graphs G' are close to the image corresponding to $\boxed{G}$.

generators on three data sets. Additional experiments provide further insight into VARSCENE and prior systems. It would be interesting to use third-party knowledge (e.g., knowledge graphs) to enhance to semantic structure of the generated scenes.

VARSCENE: A Deep Generative Model for Realistic Scene Graph Synthesis (Supplementary Material)

Remarks

1. Eq. (21) was incorrectly typeset and should be
$$\frac{\textsc{mask}(s_i, s_j, r)[\![s_j \notin G_\Delta]\!] \exp(M_\theta(\boldsymbol{s_j}, \boldsymbol{z}_\Delta))}{\sum_{s \in S(D) \setminus G_\Delta} \textsc{mask}(s_i, s, r) \exp(M_\theta(\boldsymbol{s}, \boldsymbol{z}_\Delta))} + \frac{\textsc{mask}(s_i, s_j, r)[\![s_j \in G_\Delta]\!]}{|\{s : \textsc{mask}(s_i, s, r) = 1, s \in G_\Delta\}|}. \tag{21$'$}$$
Specifically, $\textsc{mask}(s, s_j, r)$ in each denominator in Eq. (21) should be replaced with $\textsc{mask}(s_i, s, r)$. Our implementation is correct and not affected.
2. In line 316, the part of the sentence "sample estimate defined in **Eq. (3)** as follows" should be replaced with "sample estimate defined in **Eq.** (2) as follows". The cross-reference was incorrect.

A. Broader impact

Generating realistic and complex scenes is a key task in image retrieval, image editing, question-answering, etc. In this work, we show that it is possible to sample such scene graphs from a distribution estimated from a set of real scene graphs, without explicitly inspecting training images. Any bias (e.g., gender, race) in the training scene graphs may propagate to the synthetic images produced by our system. Our model has sufficient flexibility to attempt to reduce or mitigate such bias. Specifically, one can replace the first term quantifying MMD in our objective function in Eq. (7) with a suitable alternative which can mitigate the biases in the generated graphs.

B. Limitations

There are a few potential limitations of VARSCENE, which we leave to be addressed as ongoing and future work.

1. The MMD computation depends heavily on the choice of graph kernel. It is likely that the decoder p_θ^{MMD} optimized for one specific choice of graph kernel may not show good performance for other types of kernels. Multitask objectives may be of interest here.
2. Currently, our model is trained in two stages. This increases the time and complexity of training. It would be of interest to train a single stage variational autoencoder, which would be also optimized for a given MMD measure (or combination of MMD measures).
3. Our generative mechanism based on star agglomeration allows us to incorporate salient semantic structures from training scenes alone. Knowledge graphs representing facts (such as WikiData) or common sense may provide valuable additional "realism prior" or "world bias" for better training with less supervision.

C. Related work

Our work is related to graph generative models and scene graph extraction from images.

C.1. Graph generative models

Driven by the success of text and image generative models, there is much recent interest in deep generative models for graphs, primarily designed for molecule discovery, *e.g.*, GraphVAE [33], GraphRNN [49], NeVAE [31], MolGAN [8], GRAN [24], etc. However, the edge labels in molecular graphs are restricted by the types of bonds present in chemical compounds, which does not exceed 10. On the other hand, the node labels correspond to the unique elements, which is typically $<$15 in the ZINC [34] and QM9 [2, 28] datasets. In contrast, the distinct number of node and edge labels in scene graph datasets are significantly higher than those in molecular graphs. E.g., Visual Genome [19] has 75,729 unique node labels and 40,480 unique edge labels. Even the smaller Visual Relationship Detection dataset [25] has 100 unique node labels and 70 unique edge labels. Apart from the smaller number of node and edge labels, molecules and protein compounds follow well-defined graph semantics, e.g., valency rules of atoms, non-existence of strained bridges and rings, and occurrence of specific motifs in molecules. These place constraints on the set of plausible graphs, which can be exploited by generative models for molecular graphs. Codifying knowledge about the visual world to aid the training process may not be straightforward in the context of scene graphs.

C.2. Scene graph extraction from images

A graph where the nodes represent objects with edges being relationships between them encapsulates the semantic content of an image in a succinct manner. We refer to the task of constructing such a graph from the visual modality as scene graph extraction (although it is sometimes called 'generation', which can be confusing). Literature associated with this (essentially inference) task is vast [22, 39, 46, 48], with alternatives characterized by the modeling approaches — e.g., do they use conditional random fields or recurrent/convolutional network variations, are the models trained in an adversarial manner, and so on. Most of these models' inputs are derived from the current image alone — these would be visual features computed from regions of interest that help detect the object type or identify the relationship between previously identified objects [7, 18, 21, 51]. More recently, inference methods aided by external knowledge have been developed. This knowledge can take the form of word embeddings [25] and knowledge graphs [50]. The task considered in the current paper is that of scene graph *synthesis*, where we do not have access to the image modality. In the absence of this visual information, toward synthesizing realistic and plausible scene graphs, we mirror the existing practice of utilizing information from the linguistic domain in the form of word embeddings of the objects and relationship names.

C.3. Image generation from scene graphs

While there has been progress in the field of image generation, the end goal of achieving rich, diverse and complex scenes of high quality still needs work [3]. In the current paper, we propose that an existing collection of scene graphs can be utilized to train a generative model of such graphs. Sampling from such a model provides us with the semantic content for a novel scene. The sampled graph can then be used as an input into a conditional image generation model [1, 17, 42]. Conditional image generation models that operate on other input modalities also exist. For example, textual phrases [37, 47], semantic segmentation masks [38], and object positions and layouts [35].

D. Additional details about VARSCENE

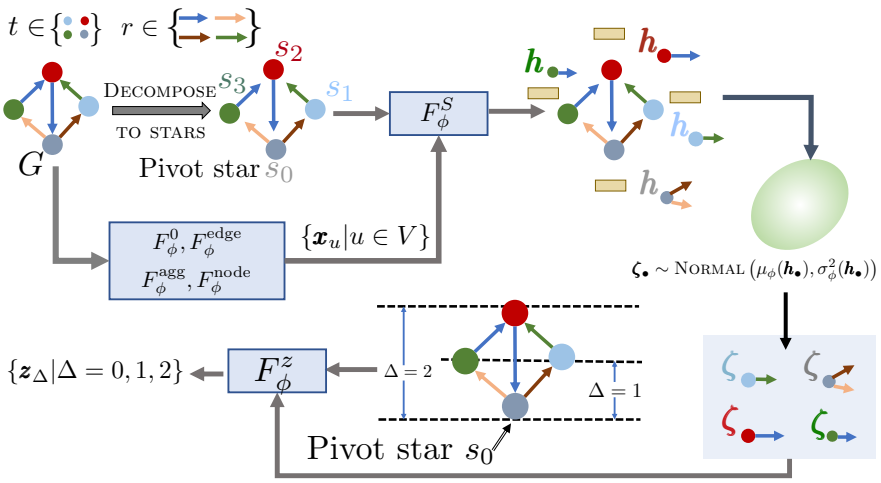

Figure 7. Neural architecture of VARSCENE encoder. Node colors represent object types; edge colors represent relation labels. Stars are defined by hub node type and relation labels but not neighbor node types. Given a graph $G = (V, E)$ along with its object types $\{t\}$ and relation types $\{r\}$, we first decompose it into the set of stars $\{s_0, s_1, s_2, s_3\}$ where s_0 is a pivot star. Next, we feed the graph into a graph neural network $(F_\phi^0, F_\phi^{\text{edge}}, F_\phi^{\text{node}}, F_\phi^{\text{agg}})$ that is used to derive node embeddings $\boldsymbol{x}_u$. These node embeddings along with the stars are fed into a neural module F_ϕ^S to compute the star embeddings $\{\boldsymbol{h}_s\}$, which are then used to generate the latent random codes $\{\boldsymbol{\zeta}_s\}$. Finally, they are aggregated based on the distance from pivot star s_0 to provide the embeddings $\{\boldsymbol{z}_\Delta\}$.

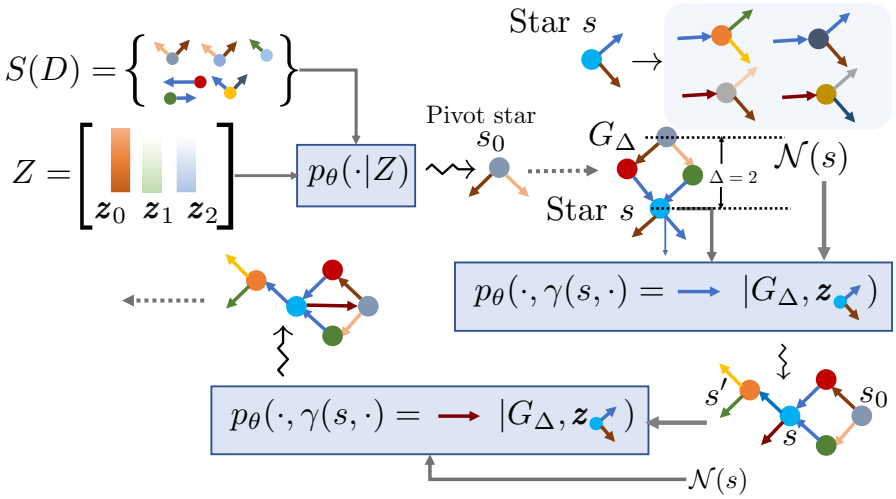

Figure 8. Neural architecture of VARSCENE decoder. Note that the base decoder p_θ and p_θ^{MMD} follow same neural architecture. Given the latent codes $Z = \{z_\Delta | \Delta = 0, 1, \ldots\}$, the decoder first samples the pivot star S_0. Then it keeps adding stars at increasing distance Δ from s_0. Given a snapshot G_Δ, the decoder p_θ draws a new star s' which is going to be connected with an existing star s at distance Δ. In this case, s' can be drawn either from the neighbor stars $\mathcal{N}(s)$ (first term in Eq. (21); the star s' in this figure, connected on the edge $\rightarrow$) or an already existing star (second term in Eq. (21); the star s_0 in this figure, connected on the edge $\rightarrow$).

We summarize the neural architecture in Figure 7 and 8. We provide more details as follows.

***t* and *r*.** Given an object type t and relation type r, we use BERT embedding [9] to obtain the corresponding representations $\boldsymbol{t}$ and $\boldsymbol{r}$. Here, we used the code from https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2 and thus have dimension(t) = dimension(r) = 384.

Specifications of the encoder. We summarize the components of the encoder as follows:

1. The GNN module of q_ϕ contains four networks: F_ϕ^0, F_ϕ^{edge}, F_ϕ^{agg} and F_ϕ^{node}, which follow similar architecture as in [12]. We summarize their architecture in details as follows:
   - F_ϕ^0: It consists of a single layer neural network with linear activation function which outputs a 64 dimensional node features.
   - F_ϕ^{edge}: It is a single layer neural network with linear activation function. It takes the pair of node features and the corresponding relation type of the edge between them as input and then outputs and a 64 dimensional edge embedding vector.
   - F_ϕ^{agg}: It is a sum-pool aggregator.

- F_ϕ^{node}: It is a GRU which sequentially takes $\boldsymbol{x}_u(k-1)$ and $\overline{\boldsymbol{x}}_u(k-1)$ as input and outputs a 64 dimensional node embedding.

2. F_ϕ^S is kept the same as F_ϕ^{node}. It is a GRU which is supplied only one element, the concatenation of $\boldsymbol{x}_{\text{root}(s)}$ and $\text{sum}(\{\boldsymbol{x}_{(\text{root}(s),v)} \mid v \in \text{nbr}(\text{root}(s))\})$. It outputs a 64 dimensional vector.
3. μ_ϕ in Eq. (18) is a two layer neural network which consists of a 128 dimensional hidden layer, ReLU activations and outputs a 64 dimensional mean vector.
4. σ_ϕ in Eq. (18) is a two layer neural network which consists of a 128 dimensional hidden layer, ReLU activations and outputs a 64x64 matrix.
5. F_ϕ^z in Eq. (19) is a sum aggregator.

Specification of the decoder. Here, M_θ in Eq. (21) is a two layer neural network which consists of a 32 dimensional hidden layer, ReLU activations and outputs a 16 dimensional mean vector.

E. Details about experiments

E.1. Pre-processing of the datasets

For the SVG dataset, we start with pre-processed graphs that were provided with the code of [11], that we obtained from the authors. For VG and VRD datasets, we construct graphs directly from the `json` formatted scene graphs. After obtaining these graphs, we split them into weakly connected components, i.e., components in which each node is reachable from all other nodes considering edges to be undirected. This step is done for all datasets. For the VG dataset, the total number of graphs after splitting exceeded ∼800K. Due to the large number of graphs, we randomly sampled 90K train graphs, 10K validation graphs and 10K test graphs for the VG dataset. For the SVG dataset, train-test split was provided in [11], hence we used the same set of test graphs and randomly sampled graphs from the train set to create validation graphs. For the VRD dataset, train-test graphs were provided, but the number of graphs was quite small. Due to this, we randomly sampled graphs from the train and test sets to create the validation graphs.

E.2. Implementation details of baselines

DeepGMG. We used the version released with the official PyTorch implementation of Goyal et al. [13][3].

MolGAN. We used an open-source PyTorch implementation [8][4].

GraphRNN. We used the label adapted version released with the official PyTorch implementation of Goyal et al. [13][5].

GraphGen. We used the official PyTorch implementation [13][6].

SceneGen. We obtained the code from the authors [11].

We summarize the total number of parameters in our model and the baselines in Table 9, which shows that our model uses minimum number of parameters and still outperforms all the baselines. Moreover, note that, our number of parameters is independent of the dataset size, whereas the complexity of baseline models widely varies across most of the datasets.

	Parameter Count		
	VG	SVG	VRD
DeepGMG	9,096,436	653,771	641,836
MolGAN	623,102,433	18,895,885	11,670,574
GraphRNN	19,561,966	529,421	510,150
GraphGen	25,056,650	2,865,855	2,781,760
SceneGen	277,012	222,781	228,019
VARSCENE	221,249	221,249	221,249

Table 9. Number of parameters in each trainable model.

E.3. Hyperparameters

VARSCENE. For the base decoder (p_θ), we trained the model for 1000 epochs. For the optimized decoder (p_θ^{MMD}), we again trained for 1000 epochs. The batch size $|\mathcal{B}| = 1024$. For training p_θ, the learning rate was kept as 10^{-4}. For p_θ^{MMD}, the learning rate was kept as 10^{-3}. Moreover, we used the Adam optimizer with a 10^{-5} weight decay factor.

SceneGen. We trained their model for 300 epochs. The batch size was $|\mathcal{B}| = 64$. A step learning rate scheduler was used for both nodes and edges, with initial learning rate 10^{-3}, final learning rate 10^{-4} and decay factor 0.95. The Adam optimizer was used for updates. These hyper-parameters were the default values.

DeepGMG, GraphRNN, GraphGen. We trained the model for 10000 epochs. Batch size $|\mathcal{B}| = 2048$. Initial learning rate was 0.003 and decayed by a factor of 0.3 after epochs 100, 200, 400 and 800. The Adam optimizer was used for parameter updates. These hyper-parameters were the default values.

MolGAN. For VRD and SVG datasets, we trained the model for 12500 epochs. For the VG dataset, we trained the model for 800 epochs (since the model took a lot of time for training). Batch size $|\mathcal{B}| = 16$. Learning rate was kept as 10^{-4}. The Adam optimizer was used for parameter updates with $\beta_1 = 0.5, \beta_2 = 0.999$. These hyper-parameters were the default values.

Infrastructure details. We performed all experiments on an Intel Xeon server with 1 TB RAM running Ubuntu 18.04.5 LTS. Our code ran on any of a TITAN RTX GPU with 24 GB RAM and two TITAN X (Pascal) GPUs with 12 GB RAM each.

[3] https://github.com/idea-iitd/graphgen
[4] https://github.com/yongqyu/MolGAN-pytorch
[5] https://github.com/idea-iitd/graphgen
[6] https://github.com/idea-iitd/graphgen

F. Additional experiments

Effect of MMD-optimized method. Table 4 compared p_θ^{MMD} against p_θ using Star-Sim and Edge-bigram-Sim measurements. Here, in Table 10, we present additional similarity and kernel measurements.

Model	Star-Sim	Edge-bigram-Sim	Node-bigram-Sim	SP-Kernel	WL-Kernel	NSPD-Kernel
Visual Genome (VG)						
p_θ^{MMD}	**0.8660**	**0.5268**	**0.6226**	0.0840	0.0779	0.0135
p_θ	0.5867	0.2588	0.2212	**0.1183**	**0.1105**	**0.0143**
Visual Relationship Detection (VRD)						
p_θ^{MMD}	**0.9140**	**0.9372**	**0.9377**	**1.4588**	**1.9218**	**0.2275**
p_θ	0.8988	0.9339	0.9306	0.9426	1.4560	0.2156
Small-sized Visual Genome (SVG)						
p_θ^{MMD}	**0.9182**	**0.6964**	**0.8111**	0.9621	0.6417	**0.0633**
p_θ	0.7120	0.4195	0.5616	**1.9098**	**0.6944**	0.0586

Table 10. Performance for the scene graphs generated by the MMD optimized decoder p_θ^{MMD} and the base decoder p_θ. Numbers in bold indicate the best performer. We observe that p_θ^{MMD} outperforms p_θ in a majority of cases. In all cases, we used conditional graph generation.

Effect of optimizing other similarity metrics. In Eq. (7), we focus on minimizing MMD between the generated and test scene graphs in terms of the RBF kernel (3). Here, we replace this MMD metric with negative of the six similarity measures used for evaluation, *i.e.*, Star-Sim, Edge-bigram-Sim, Node-bigram-Sim, SP-Kernel, WL-Kernel and NSPD-Kernel. Tables 11–13 summarize the results for different datasets, and show that the superiority of our method is consistent across them.

Model	Star-Sim	Edge-bigram-Sim	Node-bigram-Sim	SP-Kernel	WL-Kernel	NSPD-Kernel
Visual Genome (VG)						
DeepGMG	0.6996	0.4668	0.1579	0.0137	0.0994	0.0144
MolGAN	0.0000	0.0000	0.0000	0.0002	0.0439	0.0131
GraphGen	0.6696	0.3760	0.1145	0.0037	0.0378	0.0126
GraphRNN	0.6352	0.0000	0.0332	0.0003	0.0339	0.0121
SceneGen	0.7378	0.5050	0.3275	0.0276	0.0804	**0.0159**
VarScene$^{\text{unc}}$	0.5963	0.4576	0.4055	0.2293	**0.1176**	0.0147
VarScene$^{\text{cond}}$	**0.8660**	0.5268	**0.6226**	0.0840	0.0779	0.0135
VarScene$^{\text{unc}}$ (Edge)	0.6999	0.3892	0.3640	0.0789	0.0806	0.0132
VarScene$^{\text{cond}}$ (Edge)	0.5930	0.1677	0.2789	0.1129	0.0899	0.0132
VarScene$^{\text{unc}}$ (Node)	0.7702	0.5497	0.4433	0.0558	0.0680	0.0129
VarScene$^{\text{cond}}$ (Node)	0.7393	0.3636	0.3726	0.0622	0.0674	0.0126
VarScene$^{\text{unc}}$ (WL)	0.7350	0.4709	0.4039	0.0865	0.0753	0.0133
VarScene$^{\text{cond}}$ (WL)	0.6819	0.4088	0.3422	0.0989	0.0784	0.0131
VarScene$^{\text{unc}}$ (SP)	0.6134	0.4246	0.3240	0.1334	0.0868	0.0135
VarScene$^{\text{cond}}$ (SP)	0.5883	0.3193	0.2960	0.1720	0.0985	0.0136
VarScene$^{\text{unc}}$ (NSPD)	0.6704	0.4831	0.3934	0.1399	0.0887	0.0134
VarScene$^{\text{cond}}$ (NSPD)	0.5932	0.3046	0.3766	**0.3390**	0.1121	0.0136
VarScene$^{\text{unc}}$ (Node, Edge)	0.7880	**0.6319**	0.5571	0.0672	0.0743	0.0133
VarScene$^{\text{cond}}$ (Node, Edge)	0.7592	0.3620	0.4573	0.0816	0.0745	0.0133
VarScene$^{\text{unc}}$ (Star, Node, Edge)	0.8013	0.5094	0.5711	0.1065	0.0887	0.0145
VarScene$^{\text{cond}}$ (Star, Node, Edge)	0.6813	0.2477	0.4131	0.1476	0.0957	0.0142
VarScene$^{\text{unc}}$ (SP, Node, Edge)	0.8066	0.5606	0.5476	0.0647	0.0746	0.0135
VarScene$^{\text{cond}}$ (SP, Node, Edge)	0.7344	0.3062	0.4343	0.0774	0.0749	0.0132

Table 11. Performance of different graph generative models (analogous to Table 2) for different optimization objectives in the second stage of our learning for Visual Genome dataset. The metrics against VarScene , in parentheses, imply that in Eqn. (7) we replace $\widehat{\text{MMD}}$ with negative of these similarity metrics. Here VarScene (Metric) means that in Eq. 7 we replace $\widehat{\text{MMD}}$ with $-$Metric. Moreover, for last few rows, we maximize the sum of multiple similarities. For example, VarScene $(\text{Metric}_1, \text{Metric}_2)$ indicates that in Eq. 7, we replace $\widehat{\text{MMD}}$ with $-(\text{Metric}_1 + \text{Metric}_2)$. We write Star for Star-Sim, Edge for Edge-bigram-Sim, Node for Node-bigram-Sim, SP for SP-Kernel, WL for WL-Kernel and NSPD for NSPD-Kernel. Recall that VarScene as-it-is stands for optimizing the discrepancy between the star distributions of the generated and the test set.

Small-sized Visual Genome (SVG)						
Model	Star-Sim	Edge-bigram-Sim	Node-bigram-Sim	SP-Kernel	WL-Kernel	NSPD-Kernel
DeepGMG	0.8097	0.6401	0.4961	0.1092	0.3589	0.0538
MolGAN	0.0000	0.0000	0.0000	0.0496	0.4084	0.0879
GraphGen	0.5223	0.6461	0.3713	0.0545	0.2117	0.0432
GraphRNN	0.2590	0.0775	0.0920	0.3833	0.5991	**0.0705**
SceneGen	0.8644	**0.8828**	**0.9311**	0.6876	0.5480	0.0663
VARSCENE$^{\text{unc}}$	0.9235	0.7049	0.8335	**1.0046**	**0.6556**	0.0685
VARSCENE$^{\text{cond}}$	0.9182	0.6964	0.8111	0.9621	0.6417	0.0633
VARSCENE$^{\text{unc}}$ (Edge)	0.9400	0.7451	0.8620	0.7177	0.6100	0.0673
VARSCENE$^{\text{cond}}$ (Edge)	0.9302	0.7538	0.8362	0.7140	0.6078	0.0624
VARSCENE$^{\text{unc}}$ (Node)	0.9351	0.7452	0.8599	0.7193	0.6135	0.0677
VARSCENE$^{\text{cond}}$ (Node)	0.9210	0.7446	0.8287	0.6726	0.6058	0.0623
VARSCENE$^{\text{unc}}$ (WL)	**0.9469**	0.7268	0.8546	0.9840	0.6342	0.0679
VARSCENE$^{\text{cond}}$ (WL)	0.9300	0.7043	0.8223	0.9395	0.6005	0.0614
VARSCENE$^{\text{unc}}$ (SP)	0.9328	0.7246	0.8576	0.8961	0.6444	0.0691
VARSCENE$^{\text{cond}}$ (SP)	0.9286	0.7171	0.8280	0.8721	0.6170	0.0636
VARSCENE$^{\text{unc}}$ (NSPD)	0.9349	0.7231	0.8498	0.8096	0.6215	0.0669
VARSCENE$^{\text{cond}}$ (NSPD)	0.9334	0.7204	0.8289	0.8038	0.6266	0.0628
VARSCENE$^{\text{unc}}$ (Node, Edge)	0.9254	0.6918	0.8443	0.9661	0.6373	0.0679
VARSCENE$^{\text{cond}}$ (Node, Edge)	0.9160	0.7044	0.8128	0.7688	0.5996	0.0621
VARSCENE$^{\text{unc}}$ (Star, Node, Edge)	0.9308	0.7039	0.8346	0.9234	0.6397	0.0679
VARSCENE$^{\text{cond}}$ (Star, Node, Edge)	0.9204	0.7028	0.8127	0.8861	0.6196	0.0627
VARSCENE$^{\text{unc}}$ (SP, Node, Edge)	0.9394	0.7322	0.8577	0.8346	0.6329	0.0684
VARSCENE$^{\text{cond}}$ (SP, Node, Edge)	0.9332	0.7206	0.8366	0.8087	0.6116	0.0629

Table 12. Performance of different graph generative models (analogous to Table 2) for different optimization objectives in the second stage of our learning for Small-sized Visual Genome dataset. The metrics against VARSCENE , in parentheses, imply that in Eqn. (7) we replace $\widehat{\text{MMD}}$ with negative of these similarity metrics. Here VARSCENE (Metric) means that in Eq. (7) we replace $\widehat{\text{MMD}}$ with $-$Metric. Moreover, for last few rows, we maximize the sum of multiple similarities. For example, VARSCENE $(\text{Metric}_1, \text{Metric}_2)$ indicates that in Eq. (7), we replace $\widehat{\text{MMD}}$ with $-(\text{Metric}_1 + \text{Metric}_2)$. We write Star for Star-Sim, Edge for Edge-bigram-Sim, Node for Node-bigram-Sim, SP for SP-Kernel, WL for WL-Kernel and NSPD for NSPD-Kernel. Recall that VARSCENE as-it-is stands for optimizing the discrepancy between the star distributions of the generated and the test set.

Visual Relationship Detection (VRD)						
Model	Star-Sim	Edge-bigram-Sim	Node-bigram-Sim	SP-Kernel	WL-Kernel	NSPD-Kernel
DeepGMG	0.7459	0.7338	0.6035	0.9977	1.4125	0.2038
MolGAN	0.0000	0.0000	0.0001	0.0149	0.9721	0.2121
GraphGen	0.6478	0.7586	0.6431	0.3184	0.7964	0.1770
GraphRNN	0.5455	0.2941	0.7163	0.2125	0.7639	0.1888
SceneGen	0.8112	**0.9486**	**0.9552**	0.6069	1.1274	0.2130
VARSCENE$^{\text{unc}}$	0.9194	0.9374	0.9403	1.0327	1.5619	0.2301
VARSCENE$^{\text{cond}}$	0.9140	0.9372	0.9377	1.4588	1.9218	0.2275
VARSCENE$^{\text{unc}}$ (Edge)	0.9216	0.9290	0.9363	1.1150	1.6175	0.2349
VARSCENE$^{\text{cond}}$ (Edge)	0.9177	0.9219	0.9343	1.4194	1.9164	0.2290
VARSCENE$^{\text{unc}}$ (Node)	0.9063	0.9296	0.9240	1.0671	1.5865	0.2321
VARSCENE$^{\text{cond}}$ (Node)	0.9062	0.9387	0.9268	1.2975	1.7934	0.2238
VARSCENE$^{\text{unc}}$ (WL)	0.8989	0.9281	0.9217	1.1509	1.6318	0.2345
VARSCENE$^{\text{cond}}$ (WL)	0.8979	0.9249	0.9251	1.4902	1.9244	0.2294
VARSCENE$^{\text{unc}}$ (SP)	0.9122	0.9317	0.9328	1.1432	1.6431	0.2379
VARSCENE$^{\text{cond}}$ (SP)	0.9108	0.9290	0.9354	1.5389	1.9726	0.2334
VARSCENE$^{\text{unc}}$ (NSPD)	0.8893	0.9011	0.9044	1.0247	1.5454	0.2276
VARSCENE$^{\text{cond}}$ (NSPD)	0.8898	0.8858	0.9065	1.2732	1.7864	0.2204
VARSCENE$^{\text{unc}}$ (Node, Edge)	0.8941	0.9254	0.9190	1.1111	1.6225	0.2332
VARSCENE$^{\text{cond}}$ (Node, Edge)	0.9029	0.9298	0.9261	1.4332	1.8953	0.2256
VARSCENE$^{\text{unc}}$ (Star, Node, Edge)	**0.9223**	0.9324	0.9425	1.1342	1.6316	0.2357
VARSCENE$^{\text{cond}}$ (Star, Node, Edge)	0.9222	0.9267	0.9455	1.4657	1.9168	0.2292
VARSCENE$^{\text{unc}}$ (SP, Node, Edge)	0.9103	0.9248	0.9364	1.2269	1.6996	**0.2398**
VARSCENE$^{\text{cond}}$ (SP, Node, Edge)	0.9173	0.9264	0.9416	**1.5882**	**2.0036**	0.2339

Table 13. Performance of different graph generative models (analogous to Table 2) for different optimization objectives in the second stage of our learning for Visual Relationship Detection dataset. The metrics against VARSCENE , in parentheses, imply that in Eqn. (7) we replace $\widehat{\text{MMD}}$ with negative of these similarity metrics. Here VARSCENE (Metric) means that in Eq. (7) we replace $\widehat{\text{MMD}}$ with $-$Metric. Moreover, for last few rows, we maximize the sum of multiple similarities. For example, VARSCENE $(\text{Metric}_1, \text{Metric}_2)$ indicates that in Eq. (7), we replace $\widehat{\text{MMD}}$ with $-(\text{Metric}_1 + \text{Metric}_2)$. We write Star for Star-Sim, Edge for Edge-bigram-Sim, Node for Node-bigram-Sim, SP for SP-Kernel, WL for WL-Kernel and NSPD for NSPD-Kernel. Recall that VARSCENE as-it-is stands for optimizing the discrepancy between the star distributions of the generated and the test set.

Effect of using RBF kernel. Recall that we use RBF kernel in Eq. (3) in our experiments. Here, we compare such an approach against using a simple cosine-similarity based kernel computation, that is, $k(G, G') = \cos(\nu(G), \nu(G'))$. Table 14 summarizes the results, which shows that using RBF kernel provides the performance boost for a majority of the similarity metrics.

Efficiency analysis. Here, we analyze efficiency of different methods. Table 15 summarizes the results, which shows that training the base VAE model of VARSCENE is fastest on Visual Genome which is the largest dataset. For smaller datasets, *i.e.*, Small-sized Visual Genome and Visual Relationship Detection, the training time is comparable with baselines. However, we note that training the MMD optimized decoder is slower in all datasets. The reason for more time in MMD optimization is that we need to sample a set of graphs (we sample 1000 graphs) at each iteration for computing

Visual Genome (VG)						
Model	Star-Sim	Edge-bigram-Sim	Node-bigram-Sim	SP-Kernel	WL-Kernel	NSPD-Kernel
VARSCENE$^{\text{unc}}$	0.5963	0.4576	0.4055	**0.2293**	**0.1176**	**0.0147**
VARSCENE$^{\text{cond}}$	**0.8660**	0.5268	**0.6226**	0.0840	0.0779	0.0135
VARSCENE$^{\text{unc}}$ (Star)	0.7779	**0.5289**	0.4732	0.0863	0.0794	0.0134
VARSCENE$^{\text{cond}}$ (Star)	0.6589	0.2630	0.3659	0.1204	0.0869	0.0133
Small-sized Visual Genome (SVG)						
Model	Star-Sim	Edge-bigram-Sim	Node-bigram-Sim	SP-Kernel	WL-Kernel	NSPD-Kernel
VARSCENE$^{\text{unc}}$	0.9235	0.7049	0.8335	**1.0046**	**0.6556**	0.0685
VARSCENE$^{\text{cond}}$	0.9182	0.6964	0.8111	0.9621	0.6417	0.0633
VARSCENE$^{\text{unc}}$ (Star)	**0.9339**	**0.7339**	**0.8505**	0.7679	0.6104	0.0667
VARSCENE$^{\text{cond}}$ (Star)	0.9235	0.7307	0.8322	0.7654	0.6153	0.0622
Visual Relationship Detection (VRD)						
Model	Star-Sim	Edge-bigram-Sim	Node-bigram-Sim	SP-Kernel	WL-Kernel	NSPD-Kernel
VARSCENE$^{\text{unc}}$	**0.9194**	**0.9374**	**0.9403**	1.0327	1.5619	0.2301
VARSCENE$^{\text{cond}}$	0.9140	0.9372	0.9377	**1.4588**	**1.9218**	0.2275
VARSCENE$^{\text{unc}}$ (Star)	0.9067	0.9294	0.9275	1.0799	1.5911	**0.2306**
VARSCENE$^{\text{cond}}$ (Star)	0.9074	0.9210	0.9301	1.4106	1.8871	0.2242

Table 14. Performance provided by two variants of VARSCENE, *viz.*, VARSCENE(RBF) which is the default variant having k as the RBF kernel as defined in Eq. (3) and VARSCENE(Star-Sim) which uses $k(G, G') = \cos(\nu(G), \nu(G'))$. Numbers in bold indicate the best performer.

$\widehat{\text{MMD}}$ in the objective as per Eqn. 9. For SVG and VRD datasets specifically, the graph generation process takes a higher amount of time. This is because its decoders sample stars beyond a limit (we set this limit as 50). In such cases, we sample graphs repeatedly until we get a valid graph or until we have repeated the procedure for a certain number of times (we set this limit as 20).

	VG	SVG	VRD
DeepGMG	2.0642	3.6121	2.5645
MolGAN	0.1274	0.1128	0.1414
GraphRNN	1.1582	0.0530	0.0435
GraphGen	0.0635	0.0429	0.0319
SceneGen	0.1945	0.0650	0.0510
VARSCENE (p_θ)	0.0334	0.0650	0.4188
VARSCENE (p_θ^{MMD})	5.3866	22.1826	24.9556

Table 15. Time (in seconds) for one iteration for a fixed batch size $|\mathcal{B}| = 64$.

References

[1] Oron Ashual and Lior Wolf. Specifying object attributes and relations in interactive scene generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4561–4569, 2019. 10

[2] L. C. Blum and J.-L. Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.*, 131:8732, 2009. 10

[3] Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. Coco-stuff: Thing and stuff classes in context. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1209–1218, 2018. 10

[4] Arantxa Casanova, Michal Drozdzal, and Adriana Romero-Soriano. Generating unseen complex scenes: are we there yet? *arXiv preprint arXiv:2012.04027*, 2020. 1

[5] Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alex Hauptmann. Scene graphs: A survey of generations and applications. *arXiv preprint arXiv:2104.01111*, 2021. 1

[6] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *ICML*, 2010. 6, 7

[7] Bo Dai, Yuqi Zhang, and Dahua Lin. Detecting visual relationships with deep relational networks. In *Proceedings of the IEEE conference on computer vision and Pattern recognition*, pages 3076–3086, 2017. 10

[8] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018. 1, 4, 5, 7, 10, 13

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 11

[10] Helisa Dhamo, Azade Farshad, Iro Laina, Nassir Navab, Gregory D Hager, Federico Tombari, and Christian Rupprecht. Semantic image manipulation using scene graphs. In *CVPR*, pages 5213–5222, 2020. 1

[11] Sarthak Garg, Helisa Dhamo, Azade Farshad, Sabrina Musatian, Nassir Navab, and Federico Tombari. Unconditional scene graph generation. *ICCV*, 2021. URL https://arxiv.org/abs/2108.05884. 1, 5, 6, 7, 13

[12] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 4, 11

[13] Nikhil Goyal, Harsh Vardhan Jain, and Sayan Ranu. Graphgen: A scalable approach to domain-agnostic labeled graph generation. *Proceedings of The Web Conference*, Apr 2020. doi: 10.1145/3366423.3380201. URL http://dx.doi.org/10.1145/3366423.3380201. 2, 5, 7, 13

[14] Jiuxiang Gu, Handong Zhao, Zhe Lin, Sheng Li, Jianfei Cai, and Mingyang Ling. Scene graph generation with external knowledge and image reconstruction. In *CVPR*, June 2019. 1

[15] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *arXiv preprint arXiv:1706.08500*, 2017. 6, 7

[16] Justin Johnson, Ranjay Krishna, Michael Stark, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Image retrieval using scene graphs. *CVPR*, pages 3668–3678, 2015. URL http://hci.stanford.edu/publications/2015/scenegraphs/JohnsonCVPR2015.pdf. 1

[17] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *CVPR*, pages 1219–1228, 2018. 1, 6, 10

[18] Matthew Klawonn and Eric Heim. Generating triples with adversarial networks for scene graph construction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018. 10

[19] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. 2016. URL https://arxiv.org/abs/1602.07332. 1, 5, 10

[20] Wenbo Li, Pengchuan Zhang, Lei Zhang, Qiuyuan Huang, Xiaodong He, Siwei Lyu, and Jianfeng Gao. Object-driven text-to-image synthesis via adversarial training. In *CVPR*, June 2019. 1

[21] Yikang Li, Wanli Ouyang, Bolei Zhou, Kun Wang, and Xiaogang Wang. Scene graph generation from objects, phrases and region captions. In *Proceedings of the IEEE international conference on computer vision*, pages 1261–1270, 2017. 10

[22] Yikang Li, Wanli Ouyang, Bolei Zhou, Jianping Shi, Chao Zhang, and Xiaogang Wang. Factorizable net: an efficient subgraph-based framework for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 335–351, 2018. 10

[23] Yujia Li, Oriol Vinyals, Chris Dyer, Razvan Pascanu, and Peter Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018. 4, 5, 7

[24] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Charlie Nash, William L. Hamilton, David Duvenaud, Raquel Urtasun, and Richard S. Zemel. Efficient graph generation with graph recurrent attention networks. *CoRR*, abs/1910.00760, 2019. URL http://arxiv.org/abs/1910.00760. 1, 10

[25] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *ECCV*, 2016. 1, 5, 10

[26] Paridhi Maheshwari, Ritwick Chaudhry, and Vishwa Vinay. Scene graph embeddings using relative similarity supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2328–2336, 2021. 1

[27] Victor Milewski, Marie-Francine Moens, and Iacer Calixto. Are scene graphs good enough to improve image captioning? 2020. URL https://arxiv.org/abs/2009.12313. 1

[28] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld. Fast and accurate modeling of molecular atomization energies with machine learning. *Physical Review Letters*, 108:058301, 2012. 10

[29] Mehdi SM Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. *Advances in Neural Information Processing Systems*, 2018. URL https://arxiv.org/abs/1806.00035. 6, 7

[30] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in Neural Information Processing Systems*, 29:2234–2242, 2016. 6, 7

[31] Bidisha Samanta, Abir DE, Gourhari Jana, Pratim Kumar Chattaraj, Niloy Ganguly, and Manuel Gomez Rodriguez. Nevae: A deep generative model for molecular graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):1110–1117, 2019. doi: 10.1609/aaai.v33i01.33011110. 1, 4, 10

[32] Brigit Schroeder and Subarna Tripathi. Structured query-based image retrieval using scene graphs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020. 1

[33] Martin Simonovsky and Nikos Komodakis. Graphvae: Towards generation of small graphs using variational autoencoders. *CoRR*, abs/1802.03480, 2018. URL http://arxiv.org/abs/1802.03480. 1, 10

[34] Teague Sterling and John J. Irwin. Zinc 15 – ligand discovery for everyone. *Journal of Chemical Information and Modeling*, 55(11):2324–2337, 2015. doi: 10.1021/acs.jcim.5b00559. URL https://doi.org/10.1021/acs.jcim.5b00559. PMID: 26479676. 10

[35] Wei Sun and Tianfu Wu. Learning layout and style reconfigurable gans for controllable image synthesis. *IEEE transactions on pattern analysis and machine intelligence*. 10

[36] Behzad Tabibian, Vicenç Gomez, Abir De, Bernhard Schölkopf, and Manuel Gomez Rodriguez. On the design of consequential ranking algorithms. In *Conference on Uncertainty in Artificial Intelligence*, pages 171–180. PMLR, 2020. 4

[37] Fuwen Tan, Song Feng, and Vicente Ordonez. Text2scene: Generating compositional scenes from textual descriptions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6710–6719, 2019. 10

[38] Hao Tang, Dan Xu, Yan Yan, Philip HS Torr, and Nicu Sebe. Local class-specific and global image-level generative adversarial networks for semantic-guided scene generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7870–7879, 2020. 10

[39] Kaihua Tang, Yulei Niu, Jianqiang Huang, Jiaxin Shi, and Hanwang Zhang. Unbiased scene graph generation from biased training. In *CVPR*, pages 3716–3725, 2020. 1, 10

[40] Damien Teney, Lingqiao Liu, and Anton van Den Hengel. Graph-structured representations for visual question answering. In *CVPR*, pages 1–9, 2017. 1

[41] Emanuel Todorov. Efficient computation of optimal actions. *Proceedings of the national academy of sciences*, 106(28):11478–11483, 2009. 3

[42] Hung-Yu Tseng, Hsin-Ying Lee, Lu Jiang, Ming-Hsuan Yang, and Weilong Yang. Retrievegan: Image synthesis via differentiable patch retrieval. In *ECCV*, pages 242–257. Springer, 2020. 1, 10

[43] S Vishy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, 2010. 4, 6, 7

[44] Sijin Wang, Ruiping Wang, Ziwei Yao, Shiguang Shan, and Xilin Chen. Cross-modal scene graph matching for relationship-aware image-text retrieval. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1508–1517, 2020. 1

[45] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992. 4

[46] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 5, 10

[47] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1316–1324, 2018. 10

[48] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018. 1, 10

[49] Jiaxuan You, Rex Ying, Xiang Ren, William L. Hamilton, and Jure Leskovec. Graphrnn: A deep generative model for graphs. *CoRR*, abs/1802.08773, 2018. URL http://arxiv.org/abs/1802.08773. 1, 2, 5, 7, 10

[50] Alireza Zareian, Svebor Karaman, and Shih-Fu Chang. Bridging knowledge graphs to generate scene graphs. In *ECCV*, 2020. 10

[51] Ji Zhang, Mohamed Elhoseiny, Scott Cohen, Walter Chang, and Ahmed Elgammal. Relationship proposal networks. In *CVPR*, pages 5678–5686, 2017. 10