Continuous Graph Flow

Zhiwei Deng [*1,2] **Megha Nawhal** [*2,3] **Lili Meng** [2] **Greg Mori** [2,3]

Abstract

In this paper, we propose *Continuous Graph Flow*, a generative continuous flow based method that aims to model complex distributions of graph-structured data. Our proposed model learns a joint probability density over a set of related random variables by formulating it as first order ordinary differential equation system with shared and reusable functions that operate over the graph structure. This leads to a reversible *continuous message passing* over time resulting in continuous transformations of probability distributions of the variables. We evaluate our model on a diverse set of generation tasks: graph generation, image puzzle generation, and layout generation from scene graphs. Experimental results show that CGF-based models outperform state-of-the-art graph generative models.

1. Introduction

Modeling and generating graph-structured data has important applications in various scientific fields such as building knowledge graphs (Lin et al., 2015; Bordes et al., 2011), inventing new molecular structures (Gilmer et al., 2017) and generating images from scene graphs (Johnson et al., 2018).

Traditional graph generative methods (Erdős & Rényi, 1959; Leskovec et al., 2010; Albert & Barabási, 2002; Airoldi et al., 2008) are based on rigid structural assumptions and lack the capability to learn from observed data. Modern deep learning frameworks based on variational autoencoders (VAE) (Kingma & Welling, 2014) enable learning structured latent space models from data (Lin et al., 2018; He et al., 2018; Kipf & Welling, 2016). Nevertheless, their capacity is still limited mainly because of the assumptions placed on the form of distributions. Another class of graph generative models are based on autoregressive methods (You et al., 2018; Kipf et al., 2018). However, due to the sequential nature of the generation process, these models suffer from the inability to maintain long-term dependencies in larger graphs. Therefore, existing methods for graph generation are yet to realize the full potential of their generative power, particularly, the ability to model complex distributions with the flexibility to address variable data dimensions.

Alternatively, for modeling the relational structure in data, graph neural networks (GNNs) or message passing neural networks (MPNNs) (Scarselli et al., 2009; Gilmer et al., 2017) have been shown to be effective in learning generalizable representations over graph-structured data. These models rely on neural message passing wherein the node representations are updated iteratively for a fixed number of steps. Hereafter, we use the term message passing to refer to neural message passing in GNNs. We leverage this representational ability towards graph generation.

In this paper, we introduce a new class of models – *Continuous Graph Flow* (CGF): a graph generative model based on continuous normalizing flows (Chen et al., 2018; Grathwohl et al., 2019) that generalizes the message passing mechanism to continuous time. Specifically, to model continuous time dynamics of the graph variables, we adopt a neural ordinary differential equation (ODE) formulation. Our CGF model has both the flexibility to handle variable data dimensions (by using GNNs) and the ability to model arbitrarily complex data distributions due to the free-form model architectures enabled by the neural ODE formulation. Inherently, the ODE formulation also imbues the model with properties such as reversibility and exact likelihood computation. Recent research attempts on normalizing flow based graph generative models (Liu et al., 2019; Madhawa et al., 2019; Shi et al., 2020; Honda et al., 2019) propose a reversible graph neural network using normalizing flows. However, they are based on fixed number of transformations as compared to our CGF that models continuous time dynamics.

2. Continuous graph flow

Given a set of random variables $\boldsymbol{V}$ containing n related variables, the goal is to learn the joint distribution $p(\boldsymbol{V})$ of the set of variables $\boldsymbol{V}$. Each element of set $\boldsymbol{V}$ is $\boldsymbol{v}_i \in$

[*]Equal contribution [1]Department of Computer Science, Princeton University, USA [2]Borealis AI, Canada [3]School of Computing Science, Simon Fraser University, Canada. Correspondence to: Zhiwei Deng <zhiweid@princeton.edu>, Megha Nawhal <mnawhal@sfu.ca>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020.

$\mathbb{R}^m$ where $i = 1, 2 \ldots, n$ and m represents the number of dimensions of the variable. For continuous time dynamics of the set of variables $\boldsymbol{V}$, we formulate an ordinary differential equation (ODE) system as follows:

$$\begin{bmatrix} \dot{\boldsymbol{v}}_1(t) \\ \dot{\boldsymbol{v}}_2(t) \\ \vdots \\ \dot{\boldsymbol{v}}_n(t) \end{bmatrix} = \begin{bmatrix} f^1(\boldsymbol{V}(t)) \\ f^2(\boldsymbol{V}(t)) \\ \vdots \\ f^n(\boldsymbol{V}(t)) \end{bmatrix}, \tag{1}$$

where $\dot{\boldsymbol{v}}_{\boldsymbol{i}} = d\boldsymbol{v}_i/dt$ and $\boldsymbol{V}(t)$ is the set of variables at time t. The random variable $\boldsymbol{v}_i$ at time t_0 follows a simple base distribution (*e.g.* Gaussian). The function f^i implicitly defines the interaction among the variables. Following this formulation, the transformation of the variable $\boldsymbol{v}_{\boldsymbol{i}}$ from time t_0 to time t_1 is defined as

$$\boldsymbol{v}_i(t_1) = \boldsymbol{v}_i(t_0) + \int_{t_0}^{t_1} f^i(\boldsymbol{V}(t))dt. \tag{2}$$

2.1. Continuous message passing

The form in Eq. 2 represents a generic multi-variate update where interaction functions are defined over all the variables in the set $\boldsymbol{V}$. However, the functions do not take into account the relational structure between the graph variables.

To address this, we define a neural message passing process that operates over a graph by defining the update functions of variables according to the graph structure relying on information gathered from other neighboring variables. This process begins from time t_0 where each variable $\boldsymbol{v}_i(t_0)$ contain local information only. For such updates, the function f^i in Eq. 2 is defined as:

$$f^i(\boldsymbol{V}(t)) = g(\{\hat{f}_{ij}(\boldsymbol{v}_i(t), \boldsymbol{v}_j(t)) | j \in \mathcal{S}(i)\}), \tag{3}$$

where $\hat{f}_{ij}(\cdot)$ is a reusable message function that passes information between variables $\boldsymbol{v}_{\boldsymbol{i}}$ and $\boldsymbol{v}_{\boldsymbol{j}}$, $\mathcal{S}(i)$ is the set of neighboring variables that interact with variable $\boldsymbol{v}_{\boldsymbol{i}}$, and $g(\cdot)$ is a function that aggregates the information passed to a variable. The above formulation describes the case of pairwise message functions, though it can be generalized to higher order interactions.

We formulate the message passing as a continuous process, thereby, eliminating the requirement of having a predetermined number of message passing steps. By further pushing the message passing process to update at infinitesimally smaller steps for an arbitrarily large number of steps, variable updates can be represented using shared and reusable functions as the following ODE system.

$$\begin{bmatrix} \dot{\boldsymbol{v}}_1(t) \\ \dot{\boldsymbol{v}}_2(t) \\ \vdots \\ \dot{\boldsymbol{v}}_n(t) \end{bmatrix} = \begin{bmatrix} g(\{\hat{f}_{1j}(\boldsymbol{v}_1(t), \boldsymbol{v}_j(t)) | j \in \mathcal{S}(1)\}) \\ g(\{\hat{f}_{2j}(\boldsymbol{v}_2(t), \boldsymbol{v}_j(t)) | j \in \mathcal{S}(2)\}) \\ \vdots \\ g(\{\hat{f}_{nj}(\boldsymbol{v}_n(t), \boldsymbol{v}_j(t)) | j \in \mathcal{S}(n)\}) \end{bmatrix}, \tag{4}$$

where $\dot{\boldsymbol{v}}_{\boldsymbol{i}} = d\boldsymbol{v}_i/dt$. Performing message passing to derive final states is equivalent to solving an initial value problem for an ODE system. Following the ODE formulation, the final states of the i_{th} node can be computed as follows:

$$\boldsymbol{v}_i(t_1) = \boldsymbol{v}_i(t_0) + \int_{t_0}^{t_1} g\left(\left\{\hat{f}_{ij}(\boldsymbol{v}_i(t), \boldsymbol{v}_j(t)) | j \in \mathcal{S}(i)\right\}\right). \tag{5}$$

2.2. Continuous density transformations

Continuous graph flow leverages the continuous message passing mechanism and formulates the message passing as implicit density transformations of the variables. Given a set of variables $\boldsymbol{V}$ with dependencies among them, the goal is to learn a model that captures the distribution from which the data were sampled. Assume the joint distribution $p(\boldsymbol{V})$ at time t_0 has a simple form such as independent Gaussian distribution for each variable $\boldsymbol{v}_i(t_0)$. The continuous message passing process allows the transformation of the set of variables from $\boldsymbol{V}(t_0)$ to $\boldsymbol{V}(t_1)$. Moreover, this process also converts the distributions over variables from simple base distributions to complex data distributions. Building on the *independent variable* continuous time dynamics described in (Chen et al., 2018), we define the dynamics corresponding to related *graph variables* as:

$$\frac{\partial \log p(\boldsymbol{V}(t))}{\partial t} = -Tr\left(\frac{\partial F}{\partial \boldsymbol{V}(t)}\right), \tag{6}$$

where F represents a set of reusable functions incorporating aggregated messages. Therefore, the joint distribution of set of variables $\boldsymbol{V}$ can be expressed as:

$$\log p(\boldsymbol{V}(t_1)) = \log p(\boldsymbol{V}(t_0)) - \int_{t_0}^{t_1} Tr\left(\frac{\partial F}{\partial \boldsymbol{V}(t)}\right). \tag{7}$$

In this paper, we use two types of density transformations described as follows.

Generic message transformations. Transformations with generic update functions where trace in Eq. 7 can be approximated instead of computing it by brute force (Grathwohl et al., 2019). The likelihood is defined as:

$$\log p(\boldsymbol{V}(t_1)) = \log p(\boldsymbol{V}(t_0)) - \mathbb{E}_{p(\boldsymbol{\epsilon})} \int_{t_0}^{t_1} \left[\boldsymbol{\epsilon}^T \frac{\partial F}{\partial \boldsymbol{V}(t)} \boldsymbol{\epsilon} dt\right], \tag{8}$$

where $\boldsymbol{\epsilon}$ is a noise vector and usually can be sampled from standard Gaussian or Rademacher distributions.

Multi-scale message transformations. As a generalization of generic message transformations, we design a model with multi-scale message passing to encode different levels of information in the variables. Similar to (Dinh et al., 2016), we construct our multi-scale CGF model by stacking several blocks wherein each flow block performs message passing based on generic message transformations. After passing the input through a block, we factor out a portion of the output and feed it as input to the subsequent block. The likelihood is defined as:

Table 1. **Results on graph generation.** Left of vertical line: Quantitative evaluation on MMD measures between test set and generated graphs (lower is better). The second set shows GRAPHRNN evaluation with node distribution matching (averaged over 5 different models with 3 trials). The third set shows evaluation for the test set for all 1024 generated graphs (averaged over 5 models). Right of vertical line: Visualization of graphs generated using our method.

Method	COMMUNITY-SMALL			EGO-SMALL		
	DEGREE	CLUSTERING	ORBIT	DEGREE	CLUSTERING	ORBIT
GRAPHVAE	0.35	0.98	0.54	0.13	0.17	0.05
DEEPGMG	0.22	0.95	0.4	0.04	0.10	0.02
GRAPHRNN	0.08	0.12	0.04	0.09	0.22	0.003
GNF + AE	0.20	0.20	0.11	0.03	0.10	0.001
CGF	0.10	0.30	0.08	0.02	0.11	0.001
GRAPHRNN (1024)	0.03	**0.01**	0.01	0.04	0.05	0.06
GNF + AE (1024)	0.12	0.15	0.02	0.01	0.03	0.0008
CGF (1024)	**0.02**	0.02	**0.001**	**0.002**	**0.007**	**0.0002**

$$\log p(\boldsymbol{V}(t_b)) = \log p(\boldsymbol{V}(t_{b-1})) - \mathbb{E}_{p(\boldsymbol{\epsilon})} \int_{t_{b-1}}^{t_b} \left[\boldsymbol{\epsilon}^T \frac{\partial F}{\partial \boldsymbol{V}(t_{b-1})} \boldsymbol{\epsilon} dt \right], \quad (9)$$

where $b = 1, 2, \ldots, (B-1)$ with B as the total number of blocks in the design of the multi-scale architecture. Assume at time t_b ($t_0 < t_b < t_1$), $\boldsymbol{V}(t_b)$ is factored out into two. We use one of these (denoted as $\tilde{\boldsymbol{V}}(t_b)$) as the input to the $(b+1)^{\text{th}}$ block. Let $\tilde{\boldsymbol{V}}(t_b)$ be the input to the next block, the density transformation is formulated as:

$$\log p(\tilde{\boldsymbol{V}}(t_{b+1})) = \log p(\tilde{\boldsymbol{V}}(t_b)) - \mathbb{E}_{p(\boldsymbol{\epsilon})} \int_{t_b}^{t_{b+1}} \left[\boldsymbol{\epsilon}^T \frac{\partial F}{\partial \tilde{\boldsymbol{V}}(t_b)} \boldsymbol{\epsilon} dt \right]. \quad (10)$$

3. Experiments

To demonstrate the effectiveness of our Continuous Graph Flow (CGF), we evaluate our model on three diverse tasks: (1) graph generation, (2) image puzzle generation, and (3) layout generation based on scene graphs. These tasks have high complexity in the distributions of graph variables with diverse message function types, therefore, together these tasks pose a challenging evaluation setup.

3.1. Graph Generation

Datasets & baselines. We evaluate our model on graph generation on two benchmark datasets EGO-SMALL and COMMUNITY-SMALL against four state-of-the-art baselines: GraphVAE (Simonovsky & Komodakis, 2018), GraphRNN (You et al., 2018), DeepGMG (Li et al., 2018), and Graph normalizing flows (GNF) (Liu et al., 2019).

Evaluation. We conduct a quantitative evaluation of the generated graphs using Maximum Mean Discrepancy (MMD) measures following (Liu et al., 2019). Baseline results are directly taken from (Liu et al., 2019).

Results and Analysis. Table 1 shows the results in terms of MMD. Our CGF outperforms the baselines by a wide margin indicating the benefit of free-flow function forms to model graph-structured data.We also visualize the graphs generated by CGF in Table 1 and observe that our model can capture the characteristics of datasets and generate diverse graphs that are not seen during the training.

3.2. Image puzzle generation

Task description. We design image puzzles for image datasets to test model's ability on fitting distributions over graphs with very complex node contents. Given an image of size $W \times W$, we design a puzzle by dividing the original image into non-overlapping unique patches each of size $w \times w$ resulting in $p = W/w$ puzzle patches both horizontally and vertically. Each patch corresponds to a node in the graph. To evaluate the performance of our model on dynamic graph sizes, instead of training the model with all nodes, we sample $\tilde{p}$ adjacent patches where $\tilde{p}$ is uniformly sampled from $\{1, \ldots, P\}$ as input to the model during training and test. In our experiments, we use patch size $w = 16$, $p \in \{2, 3, 4\}$ and edge function for each direction (*left*, *right*, *up*, *down*) within a neighbourhood of a node.

Datasets & baselines. We design this task for three datasets: MNIST (LeCun et al., 1998), CIFAR10 (Krizhevsky et al., 2009), and CelebA (Liu et al., 2015). We split the CelebA dataset into a training set of 27,000 images and test set of 3,000 images following (Kingma & Dhariwal, 2018). We compare our model with six state-of-the-art models: (1) BiLSTM + VAE: a bidirectional LSTM used to model the interaction between latent variables (obtained after serializing the graph) in an autoregressive manner as in (Gregor et al., 2015), (2) StructuredVAE (He et al., 2018), (3) Graphite (Grover et al., 2019), (4) Variational message passing using structured inference networks (VMP-SIN) (Lin et al., 2018), (5) Variational graph autoencoder (GAE) (Kipf & Welling, 2016), and (6) Neural relational inference (NRI) (Kipf et al., 2018).

Results and analysis. We report the negative log likelihood (NLL) in bits/dimension (lower is better). The results in Table 2 indicate that CGF significantly outperforms the baselines. In addition to the quantitative results, we also conduct sampling based evaluation and perform two types of generation experiments: (1) *Unconditional Generation:* Given a puzzle size p, p^2 puzzle patches are generated using a vector z sampled from Gaussian distribution (refer Figure 1(a));

Table 2. **Quantitative results on image puzzle generation and layout generation.** Comparison of our CGF model with baselines in bits/dimension (lower is better). These results are for unconditional generation obtained using multi-scale message transformations for image puzzle generation and generic message transformations for scene graph based layout generation.

Methods	Image Puzzle Generation									Layout Generation	
	MNIST			CIFAR-10			CelebA-HQ			Visual Genome	COCO-Stuff
	2x2	3x3	4x4	2x2	3x3	4x4	2x2	3x3	4x4		
BiLSTM + VAE	4.97	4.77	4.42	6.02	5.20	4.53	5.72	5.66	5.48	-1.20	-1.60
StructuredVAE (He et al., 2018)	4.89	4.65	3.82	6.03	5.02	4.70	5.66	5.43	5.27	-1.05	-1.36
Graphite (Grover et al., 2019)	4.90	4.64	4.02	6.06	5.09	4.61	5.71	5.50	5.32	-1.17	-0.93
VMP-SIN (Lin et al., 2018)	5.13	4.92	4.44	6.00	4.96	4.34	5.70	5.43	5.27	-0.61	-0.85
GAE (Kipf & Welling, 2016)	4.91	4.89	4.17	5.83	4.95	4.21	5.71	5.63	5.28	-1.85	-1.92
NRI (Kipf et al., 2018)	4.58	4.35	4.11	5.44	4.82	4.70	5.36	5.43	5.28	-0.76	-0.91
CGF	**1.24**	**1.21**	**1.20**	**2.42**	**2.31**	**2.00**	**3.44**	**3.17**	**3.16**	**-4.24**	**-6.21**

Figure 1. **Qualitative results for image puzzle generation.** Samples generated using our model for 2x2 MNIST puzzles and 3x3 CelebA-HQ puzzles in (left) *unconditional generation* and (right) *conditional generation* settings. For conditional setting, generated patches (highlighted in green boxes) are conditioned on the remaining patches (from ground truth).

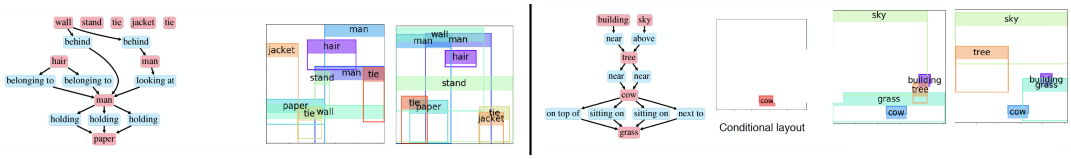

Figure 2. **Visualization for layout generation** on Visual Genome. Our CGF model can generate diverse layouts for the same scene graph. Left: layout samples with unconditional generation. Right: Layout generation conditioned on known layout.

and (2) *Conditional Generation:* Given p_1 patches from an image puzzle having p^2 patches, we generate the remaining $(p^2 - p_1)$ patches of the puzzle using our model (see Figure 1(b)). We believe the task of conditional generation is easier than unconditional generation as there is more relevant information in the input during flow based transformations. For unconditional generation, samples from a base distribution are transformed into learnt data distribution using the CGF model. For conditional generation, we map $\boldsymbol{x}_a \in \boldsymbol{X}_a$ where $\boldsymbol{X}_a \subset \boldsymbol{X}$ to the points in base distribution to obtain $\boldsymbol{z}_a$. Subsequently, we concatenate the samples from Gaussian distribution to $\boldsymbol{z}_a$ and obtain $\boldsymbol{z}'$ that matches the dimensions of desired graph, and generate samples by transforming $\boldsymbol{z}'$ to pixel space using a trained graph flow.

3.3. Layout generation from scene graphs

Task description. Scene graphs represent scenes as directed graphs wherein nodes are objects and edges give relationships between objects. Object layouts are described by sets of corresponding bounding box annotations (Johnson et al., 2018). Our model uses scene graph as inputs. An edge function is defined for each relationship type. The output contains a set of object bounding boxes described by $\{[x_i, y_i, h_i, w_i]\}_{i=1}^{n}$, where x_i, y_i are the top-left coordinates, and w_i, h_i are the bounding box width and height respectively. We use negative log likelihood per node (lower is better) to evaluate the models.

Datasets & baselines. We use two large-scale datasets for evaluation: Visual Genome (Krishna et al., 2017) and COCO-Stuff (Caesar et al., 2018) datasets. We use the same baselines as in Sec. 3.2.

Results and analysis. We show quantitative results in Table 2 against several state-of-the-art baselines. Our CGF model significantly outperforms these baselines in terms of negative log likelihood. Moreover, we show some qualitative results in Figure 2. Our model can learn the correct relations defined in scene graphs for both conditional and unconditional generation. Furthermore, our model learns one-to-many mappings and generates diverse layouts for the same scene graph.

4. Conclusion

In this paper, we presented *continuous graph flow*, a generative model that generalizes the neural message passing in graphs to continuous time. Experimental results on diverse set of generation tasks showed that continuous graph flow achieves significant performance improvement over various state-of-the-art baselines. For future work, we intend to focus on reversible generative modeling for large-scale complex graphs and spatio-temporal graphs to effectively leverage continuous time modeling.

Acknowledgements

We thank Leonid Sigal for the constructive feedback on the draft. This work was supported by Borealis AI, and much of the work was done when Zhiwei Deng and Megha Nawhal were interns at Borealis AI.

References

Airoldi, E. M., Blei, D. M., Fienberg, S. E., and Xing, E. P. Mixed membership stochastic blockmodels. *Journal of machine learning research (JMLR)*, 2008.

Albert, R. and Barabási, A.-L. Statistical mechanics of complex networks. *Reviews of modern physics*, 2002.

Bordes, A., Weston, J., Collobert, R., and Bengio, Y. Learning structured embeddings of knowledge bases. In *AAAI Conference on Artificial Intelligencce*, 2011.

Caesar, H., Uijlings, J., and Ferrari, V. Coco-stuff: Thing and stuff classes in context. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *International Conference on Learning Representations (ICLR)*, 2016.

Erdős, P. and Rényi, A. On the evolution of random graphs. *Publicationes Mathematicae (Debrecen)*, 1959.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, 2017.

Grathwohl, W., Chen, R. T., Betterncourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations (ICLR)*, 2019.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. Draw: A recurrent neural network for image generation. *International Conference on Machine Learning (ICML)*, 2015.

Grover, A., Zweig, A., and Ermon, S. Graphite: Iterative generative modeling of graphs. *International Conference on Machine Learning (ICML)*, 2019.

He, J., Gong, Y., Marino, J., Mori, G., and Lehrmann, A. Variational autoencoders with jointly optimized latent dependency structure. In *International Conference on Learning Representations (ICLR)*, 2018.

Honda, S., Akita, H., Ishiguro, K., Nakanishi, T., and Oono, K. Graph residual flow for molecular graph generation. *arXiv preprint arXiv:1909.13521*, 2019.

Johnson, J., Gupta, A., and Fei-Fei, L. Image generation from scene graphs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances of Neural Information Processing Systems (NeurIPS)*, 2018.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *International Conference on Learning Representations (ICLR)*, 2014.

Kipf, T., Fetaya, E., Wang, K.-C., Welling, M., and Zemel, R. Neural relational inference for interacting systems. *International Conference on Machine Learning (ICML)*, 2018.

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

Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision (IJCV)*, 2017.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research (JMLR)*, 2010.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. In *International Conference on Machine Learning (ICML)*, 2018.

Lin, W., Hubacher, N., and Khan, M. E. Variational message passing with structured inference networks. In *International Conference on Learning Representations (ICLR)*, 2018.

Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. Learning entity and relation embeddings for knowledge graph completion. In *AAAI conference on artificial intelligence*, 2015.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. In *Advances in Neural Information Processing Systems (NIPS)*, 2019.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

Madhawa, K., Ishiguro, K., Nakago, K., and Abe, M. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 2009.

Shi, C., Xu, M., Zhu, Z., Zhang, W., Zhang, M., and Tang, J. Graphaf: a flow-based autoregressive model for molecular graph generation. In *International Conference on Learning Representations (ICLR)*, 2020.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks (ICANN)*, 2018.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. Graphrnn: Generating realistic graphs with deep auto-regressive models. *International Conference on Machine Learning (ICML)*, 2018.

5. Appendix

We provide supplementary materials to support the contents of the main paper. In this part, we describe implementation details of our model.

5.1. Implementation Details

The ODE formulation for continuous graph flow (CGF) model was solved using ODE solver provided by NeuralODE (Chen et al., 2018). In this section, we provide specific details of the configuration of our CGF model used in our experiments on two different generation tasks used for evaluation in the paper.

Graph Generation. For each graph, we firstly generate its line graph with edges switched to nodes and nodes switched to edges. Then the graph generation problem is now generating the current nodes values which represents the adjacency matrix in the original graph. Each node value is binary (0 or 1) and is dequantized to continuous values through variational dequantization, with a global learnable Gaussian distribution as variational distribution. For our architecture, we use two blocks of continuous graph flow with two fully connected layers in Community-small dataset, and one block of continuous graph flow with one fully connected layer in Citeseer-small dataset. The hidden dimensions are all 32.

Image puzzle generation. Each graph for this task comprise nodes corresponding to the puzzle pieces. The pieces that share an edge in the puzzle grid are considered to be connected and an edge function is defined over those connections. In our experiments, each node is transformed to an embedding of size 64 using convolutional layer. The graph message passing is performed over these node embeddings. The image puzzle generation model is designed using a multi-scale continuous graph flow architecture. We use two levels of downscaling in our model each of which factors out the channel dimension of the random variable by 2. We have two blocks of continuous graph flow before each downscaling wth four convolutional message passing blocks in each of them. Each message passing block has a unary message passing function and binary passing functions based on the edge types – all containing hidden dimensions of 64.

Layout generation for scene graphs. For scene graph layout generation, a graph comprises node corresponding to object bounding boxes described by $\{[x_i, y_i, h_i, w_i]\}_{i=1}^{n}$, where x_i, y_i represents the top-left coordinates, and w_i, h_i represents the bounding box width and height respectively and edge functions are defined based on the relation types. In our experiments, the layout generation model uses two blocks of continuous graph flow units, with four linear graph message passing blocks in each of them. The message passing function uses 64 hidden dimensions, and takes the embedding of node label and edge label in unary message passing function and binary message passing function respectively. The embedding dimension is also set to 64 dimensions. For binary message passing function, we pass the messages both through the direction of edge and the reverse direction of edge to increase the model capacity.