Deep Generative Probabilistic Graph Neural Networks for Scene Graph Generation

Mahmoud Khademi
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
mkhademi@sfu.ca

Oliver Schulte
School of Computing Science
Simon Fraser University
Burnaby, BC, Canada
oschulte@sfu.ca

Abstract

We propose a new algorithm, called Deep Generative Probabilistic Graph Neural Networks (DG-PGNN), to generate a scene graph for an image. The input to DG-PGNN is an image, together with a set of region-grounded captions and object bounding-box proposals for the image. To generate the scene graph, DG-PGNN constructs and updates a new model, called a Probabilistic Graph Network (PGN). A PGN can be thought of a scene graph with uncertainty: it represents each node and each edge by a CNN feature vector and defines a probability mass function (PMF) for node-type (object category) of each node and edge-type (predicate class) of each edge. The DG-PGNN sequentially adds a new node to the current PGN by learning the optimal ordering in a Deep Q-learning framework, where states are partial PGNs, actions choose a new node, and rewards are defined based on the ground-truth. After adding a node, DG-PGNN uses message passing to update the feature vectors of the current PGN by leveraging contextual relationship information, object co-occurrences, and language priors from captions. The updated features are then used to fine-tune the PMFs. Our experiments show that the proposed algorithm significantly outperforms the state-of-the-art results on the Visual Genome dataset.

1 Introduction

Visual understanding of a scene is one of the most important objectives in computer vision. Over the past decade, there have been great advances in relevant tasks such as image classification and object detection. However, understanding a scene is not only recognizing the objects in the scene. The interactions between objects also play a crucial role in visual understanding of a scene. To represent the semantic content of an image, previous work proposed to build a graph called scene graph [2, 4, 3], where the nodes represent the objects and the edges show the relationships between them (see Figure 1). This paper therefore introduces a new algorithm, Deep Generative Probabilistic Graph Neural Networks (DG-PGNN), to generate a scene graph for an image.

An efficient scene graph generation model needs to leverage visual contextual information as well as language priors. Previous work proposed to learn a contextualized representation for nodes and edges either by sending messages across a fixed complete graph [9], where each node is potentially an object, or by assuming a fixed linear ordering for the bounding-boxes and applying a bidirectional LSTM [11]. The node and edge representations are then used to infer the node-types and edge-types. However, these models cannot leverage the rich latent graph-structured nature of the input. Unlike these approaches, our method is well-designed for graph-structured input data; we simultaneously build the graph and fine-tune the predictions, as we get new information about the scene. We introduce a new graph neural network model, Probabilistic Graph Network (PGN), to represent a scene graph with uncertainty. The PGN is a probabilistic extension to Graph Network [1]. In a PGN,

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

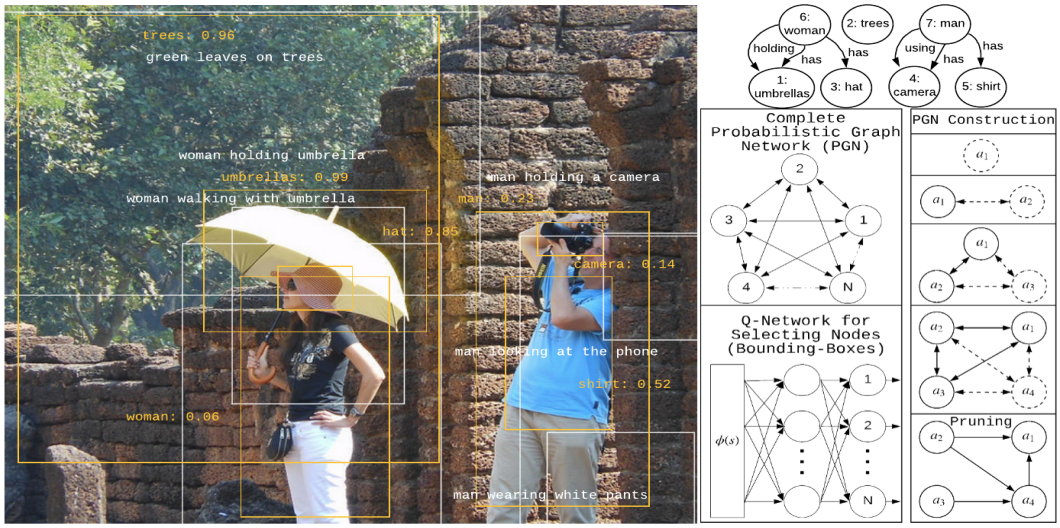

Figure 1: Scene graph generation using DG-PGNN (see the text).

each node and each edge is represented by a CNN feature vector, and the degree of belief about node-type (object category) of a node and edge-type (predicate class) of an edge is represented by a probability mass function (PMF). Our novel DG-PGNN algorithm constructs a PGN as follows. It sequentially adds a new node to the current PGN by learning the optimal ordering in a Deep Q-learning framework, where states are partial PGNs, actions choose a new node, and rewards are defined based on the ground-truth scene graph of the input image. After adding a node, DG-PGNN uses message passing to update the feature vectors of the current PGN by leveraging contextual relationship information, object co-occurrences, and language priors from captions. The updated features are then used to fine-tune the PMFs. Our experiments show that DG-PGNN substantially outperforms the state-of-the-art models for scene graph generation on Visual Genome dataset.

In summary, our contributions are the following: i) A new graph neural network model for representing the uncertainty associated with a scene graph. ii) A new scene graph construction algorithm that combines deep feature learning with probabilistic message passing in a completely differentiable probabilistic framework. The DG-PGNN sequentially adds a new node to the graph in a reinforcement learning (RL) framework. The motivation for using RL is that a graph can be generated through sequentially choosing components, and Markov decision process is an effective framework for sequential decision making. RL provides a scalable and differentiable approach for structure prediction tasks. Although we focus on the scene graph generation task, the DG-PGNN is a generic neural method for feature learning with latent graph structures, when the structure is unknown in advance, e.g. knowledge graph construction from texts. iii) To the best of our knowledge, this is the first work that explicitly exploits textual information of an image to build a scene graph for the image. Textual information is a rich source of information about object attributes and relationships.

2 Related Work

Recently, [4] proposed a model to detect a set of relationships using language priors from semantic word embeddings. [11] proposed to learn a contextualized representation for nodes and edges by assuming a fixed linear ordering for the bounding-boxes and applying a bidirectional LSTM. In [7], authors proposed to train a CNN which takes in an image and produces a scene graph in an end-to-end framework. In [8], authors introduced a model which composes dynamic tree structures that put the objects in an image into a visual context to improve scene graph generation accuracy. In [9], authors developed a model for scene graph generation which uses an RNN and learns to improves its predictions iteratively through message passing across the scene graph.

3 Proposed Algorithm

A scene graph for an image is generated from a Probabilistic Graph Network (PGN). We introduce the PGN, then explain the DG-PGNN algorithm for generating a PGN for an image.

Probabilistic Graph Networks. A Probabilistic Graph Network is defined based on a given (scene) graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes (bounding-boxes), and $\mathcal{E}$ is a set of probabilistic adjacency matrices. Each $|\mathcal{V}| \times |\mathcal{V}|$ matrix $E_k \in \mathcal{E}$ represents a probabilistic adjacency matrix for edge-type (predicate class) k. That is, for each $k \in \{1, \dots, K\}$, $E_k(u, v)$ is the probability that there exists an edge of type k going from node u to node v, where K is the number of edge-types. We write $\mathbf{e}_{u,v} = [E_1(u, v), \dots, E_K(u, v)]^\top \in \mathbb{R}^K$ for the PMF from u to v. The sum of the entries of a PMF must equal 1. We relax this condition for edges, to allow zero or multiple edges with various edge-types from u to v. Each edge $e = (u, v)$ has an edge representation (feature) $\mathbf{h}_{u,v} \in \mathbb{R}^D$. All edges from u to v with different edge-types share the same edge representation. Also, each node v has a node representation $\mathbf{h}_v \in \mathbb{R}^D$ and a node-type PMF $\mathbf{n}_v \in \mathbb{R}^M$, where M is the number of node-types (including "background"), and ith entry of $\mathbf{n}_v$ is the probability that node v has type i. At each step, the DG-PGNN algorithm has two major parts: i) a Q-learning framework for selecting the next node. ii) PGN updates which update the current PGN after adding the new node and edges.

Q-learning for Node Selection. We first construct a complete PGN with N nodes using all N bounding-box proposals of the input image, and all edges between them. The details about constructing the complete PGN are in the supplementary. The complete PGN provides initial node feature vectors, edge feature vectors, and PMFs for the DG-PGNN algorithm. Using deep reinforcement learning, we train a policy to sequentially add new nodes starting from a null graph. An RL state is a PGN, each action chooses a new node, and rewards are defined based on the ground-truth graph. We use two fully-connected layers with a ReLU activation function to implement the Q-network. Let the current state be a partial PGN denoted by s. The input to the Q-network is defined as

$$\phi = \phi(s) = [\mathbf{g}, \mathbf{p}, \mathbf{d}, \mathbf{h}_1, \dots, \mathbf{h}_N, \mathbf{n}_1, \dots, \mathbf{n}_N, \mathbf{o}^{\text{n}}, \mathbf{o}^{\text{e}}] \tag{1}$$

where $\mathbf{g}$ is a global image feature vector extracted from the last layer of 152-layer ResNet, $\mathbf{o}^{\text{n}}$ and $\mathbf{o}^{\text{e}}$ are graph-level representations of the current PGN initialized to $\mathbf{0}$, $\mathbf{p} \in \mathbb{R}^N$ is defined as $\mathbf{p} = [p(1), \dots, p(N)]$, where $p(j)$ is the confidence score of jth box, and vector $\mathbf{d} \in \mathbb{R}^N$ is defined as: $\mathbf{d}(v) = 1$ if v is selected before, otherwise $\mathbf{d}(v) = 0$. The feature vector ϕ provides a history about the nodes, node-types, and edge-types which were selected before. Thus, the Q-network can exploit information about co-occurrence of the objects for selecting the next node. For example, if the current graph has a node with type *street*, it is likely that the image contains object *car*; so the Q-network is more likely to select a bounding-box with a high node-type probability for *car*.

At each step, we select an action from the set $\mathcal{A} = \{v : \mathbf{d}(v) = 0\} \cup \{\texttt{STOP}\}$, where, `STOP` is a special action that indicates the end of graph construction. The reward function for taking action a at state s is defined as: $r(a, s) = 1$ if there exists node v such that $\texttt{IoU}(\texttt{B}(a), \texttt{B}(v)) \geq 0.5$, otherwise $r(a, s) = -1$, where `IoU` stands for Intersection over Union, and $\texttt{B}(x)$ denotes the bounding-box of node x. To update the parameters of the Q-networks, we choose a random minibatch from an experience replay memory [5, 6]. Let $(\hat{\phi}, \hat{s}, \hat{a}, \hat{r}, \hat{\phi}', \hat{s}')$ be a random transition sample. The target Q-value for the network is obtained as

$$\hat{y} = \hat{r} + \gamma \max_{v \in \hat{\mathcal{A}}'} Q\big(\phi(\hat{s}' + v); \hat{\boldsymbol{\theta}}\big) \tag{2}$$

where $\hat{\mathcal{A}}'$ is a set of actions that can be taken in state $\hat{s}'$, and $\hat{s}' + v$ is obtained by adding node v to $\hat{s}'$. Finally, the parameters of the model are updated as follows:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + \alpha\big(\hat{y} - Q\big(\phi(\hat{s} + \hat{a}); \boldsymbol{\theta}\big)\big) \nabla_{\boldsymbol{\theta}} Q\big(\phi(\hat{s} + \hat{a}); \boldsymbol{\theta}\big). \tag{3}$$

To train the model, we use *ϵ-greedy learning*, that is, with probability ϵ a random action is selected, and with probability $1 - \epsilon$ an optimal action is selected, as indicated by the Q-networks. For test images, we construct the graph by sequentially selecting the optimal actions (bounding-boxes) based on the highest Q-values until it selects the `STOP`.

PGN Updates. Whenever we add a new node to the current PGN, we connect the new node to each node of the current PGN using two directed edges. The PMFs between the new node and each node in the current PGN, the features for the new node, and the features for the new edges are initialized by copying the corresponding features and PMFs from the complete PGN (see Figure 1). Then, the node features, edge features, and PMFs of the current graph are updated. A node representation update uses the features of edges linked to the node. An edge representation update uses the node features linked by the edge. Caption information is included in feature updates if available. A node-type PMF update uses the node and graph-level representations. An edge-type probability update uses

the features of the edge and the node representations and node-type probabilities linked by the edge. These iterative updates exploit visual contextual information as well as the region-grounded captions of the image, hence reducing uncertainty about objects and predicate classes. The details about PGN updates are in the supplementary. After constructing the PGN, we eliminate the edges that have a probability less than 0.5 and the nodes with background node-type to obtain the scene graph.

4 Experiments

We introduce the dataset, baselines and evaluation metrics. Then, results are presented and discussed.

Dataset. The Visual Genome (VG) dataset [2] contains $108,077$ images. Annotations provide subject-predicate-object triplets. Following [9], we use the most frequent 150 object categories and 50 predicates for scene graph prediction task. This results in a scene graph of about 11.5 objects and 6.2 relationships per image. The training and test splits contains 70% and 30% of the images.

Metrics. Top-K recall (`Rec@K`) is used as the metric, which is the fraction of the ground truth relationship triplets subject-predicate-object hit in the top-K predictions in an image. Predictions are ranked by the product of the node-type probability of the subject, the node-type probability of the object, and the edge-type probability of the predicate. Following [9], we evaluate our model based on three tasks as follows: i) Predicate classification (PRED-CLS) task: to predict the predicates of all pairwise relationships of a set of objects, where the location and object categories are given. ii) Scene graph classification (SG-CLS) task: to predict the predicate and the object categories of the subject and object in all pairwise relationships, where the location of the objects are given. iii) Scene graph generation (SG-GEN) task: to detect a set of object bounding-boxes and predict the predicate between each pair of the objects, at the same time. An object is considered to be properly detected, if it has at least 0.5 `IoU` overlap with a bounding-box annotation with the same category.

Baseline Models. We compare our model with several models including the state-of-the-art models [8, 11, 7]. [4] uses language priors from semantic word embeddings, while [9] uses an RNN and learns to improves its predictions iteratively through message passing across the scene graph. After a few steps the learned features are classified. Moreover, three ablation models help evaluate the impact of each component of DG-PGNN: i) DG-PGNN$^\circ$ is the initial complete PGN without applying the DG-PGNN algorithm (a node/edge in the initial complete PGN is pruned if its probability is less than 0.5), ii) DG-PGNN$^-$ is the same as DG-PGNN except that it does not use captions. iii) DG-PGNN† is the same as the full model (DG-PGNN), but it uses VGG features instead of ResNet.

	PRED-CLS		SG-CLS		SG-GEN	
Model	R@50	R@100	R@50	R@100	R@50	R@100
SG-LP [4]	27.88	35.04	11.79	14.11	00.32	00.47
SG-IMP [9]	44.75	53.08	21.72	24.38	03.44	04.24
MSDN[3]	63.12	66.41	19.30	21.82	7.73	10.51
GRCNN [10]	54.2	59.1	29.6	31.6	11.4	13.7
PIX2GR [7]	68.0	**75.2**	26.5	30.0	9.7	11.3
MOTIF [11]	65.2	67.1	35.8	36.5	27.2	30.3
VCTREE [8]	66.4	68.1	38.1	38.8	27.9	31.3
DG-PGNN$^\circ$	63.5	65.3	34.2	34.8	26.1	28.8
DG-PGNN$^-$	67.3	70.1	38.6	39.2	30.2	31.8
DG-PGNN†	69.0	72.1	39.3	40.1	31.2	32.5
DG-PGNN	**70.1**	73.0	**39.5**	**40.8**	**32.1**	**33.1**

Table 1: Recall for predicate classification, scene graph classification and generation on VG dataset.

Results and Discussion. Our experimental results are reported in Table 1. The results show DG-PGNN outperforms the state-of-the art models for scene graph generation task. Both DG-PGNN and [9] leverage the power of RNNs to learn a feature vector for each bounding-box. DG-PGNN incorporates captions and contextual information of the image via information propagation to construct precise scene graphs. However, [9] suffers from imbalanced classification problem (often there is no edge between many pairs of objects). DG-PGNN† results (comparing to DG-PGNN) show the effect of backbone CNN is insignificant. This is because our algorithm uses strong contextual information to compensate for the detection errors caused by applying the VGG. Figure 1 illustrates a scene graph generated by our model. The DG-PGNN predicts a rich semantic representation of the given image by recognizing objects, their locations, and relationships between them. For example, DG-PGNN can correctly detect interactions (*man using camera*, and *man has shirt*). Also, DG-PGNN can correctly recognize *woman* (instead of *girl*). More examples are provided in the supplementary.

References

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[2] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *arXiv preprint arXiv:1602.07332*, 2016.

[3] 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 Conference on Computer Vision and Pattern Recognition*, pages 1261–1270, 2017.

[4] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual relationship detection with language priors. In *European Conference on Computer Vision*, pages 852–869. Springer, 2016.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[7] Alejandro Newell and Jia Deng. Pixels to graphs by associative embedding. In *Advances in neural information processing systems*, pages 2171–2180, 2017.

[8] Kaihua Tang, Hanwang Zhang, Baoyuan Wu, Wenhan Luo, and Wei Liu. Learning to compose dynamic tree structures for visual contexts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6619–6628, 2019.

[9] 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*, volume 2, 2017.

[10] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph r-cnn for scene graph generation. *arXiv preprint arXiv:1808.00191*, 2018.

[11] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural motifs: Scene graph parsing with global context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5831–5840, 2018.

[12] Yu Zhang, Guoguo Chen, Dong Yu, Kaisheng Yaco, Sanjeev Khudanpur, and James Glass. Highway long short-term memory rnns for distant speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5755–5759. IEEE, 2016.

Supplementary Materials

1 More Examples for Scene Graph Generation Task

Figure S1 shows three examples of scene graphs generated by DG-PGNN on Visual Genome dataset.

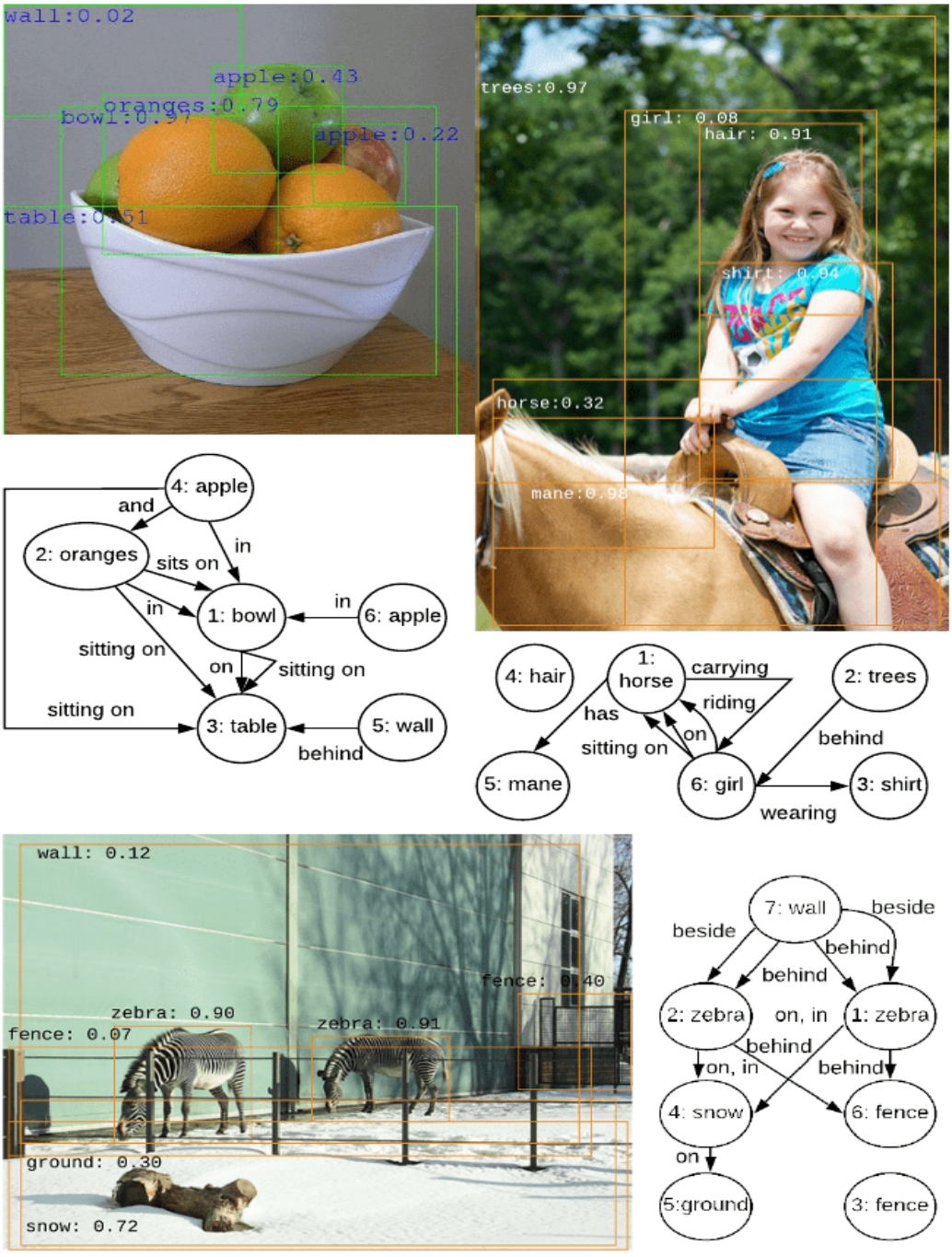

Figure S1: Examples of scene graphs generated by DG-PGNN on Visual Genome dataset using the most frequent 300 object categories. DG-PGNN can correctly detect spatial relationships *apple in bowl*, *trees behind girl*, *wall beside zebra*, and *zebra behind fence*. Also, DG-PGNN can correctly recognize *zebra in snow* (instead of *zebra in grass*).

2 PGN Updates

Node representation update. We compute a new node representation $\mathbf{h}'_v$ for each node v, every time that we extend the graph. This allows to exploit information about co-occurrence of objects. For example, if the current graph has a node with type *building*, it is likely that the graph contains *window*. For each node v in the current PGN, an ideal node representation must take into account the information from every node which has an edge to v based on the strength of the edge. The update equations are as follow

$$\mathbf{a}_v = \sum_{k=1}^{K} \sum_{u \in \mathcal{V}} E_k(u, v) \mathbf{h}_{u,v} \tag{4}$$

$$\mathbf{h}'_v = \texttt{LSTM}(\mathbf{a}_v, \mathbf{h}_v) \tag{5}$$

where, $\mathcal{V}$ is the set of the nodes in the current PGN. Intuitively, $\mathbf{a}_v$ is the aggregation of *massages* from all nodes to v weighted by the strength of their outgoing edge to v. The update mechanism first computes the node activations $\mathbf{a}_v$ for each node v. Then, a (one-step) GRU is used to update node representation for each node by incorporating information from the previous node representation $\mathbf{h}_v$. Note that unlike [3, 9], our message propagation mechanism is based on strength of the edges, not the predefined structure of the graph, since E is probabilistic. Also, our message passing scheme is part of a larger RL-based architecture. Thus, unlike these work which use all candidate boxes, in our DG-PGNN the massages propagate through the partial graph that has been constructed by DG-PGNN, not the complete graph.

Edge representation update. After updating the node representations, we compute a new edge representation $\mathbf{h}'_{u,v}$ for each edge $e = (u, v)$. This allows to exploit contextual relationship information. For example, if the type of node u is *man*, and the type of node v is *horse*, it is likely that the edge-type of $e = (u, v)$ be *riding*. We use a (one-step) GRU to update the edge representations as

$$\mathbf{h}'_{u,v} = \texttt{GRU}([\mathbf{h}_u, \mathbf{h}_v], \mathbf{h}_{u,v}) \tag{6}$$

Caption-guided updates. If a word of a caption referrers to a node-type, then it is useful to update the node representation of a node which is located at the region of the caption based on the encoded caption. For example, for region-grounded caption *a yellow umbrella*, it is useful to update the node representation of a node that is located at the region of the caption, such that the new representation increases the degree of the belief that the node-type of the node is *umbrella*; or *man is riding a horse* increases the chance that an edge that is located at the region of the caption has type *riding*. Given a new node v, if there is a caption c such that $\texttt{IoU}(\texttt{B}(c), \texttt{B}(v)) \geq 0.5$, we update the node representation of v as $\mathbf{h}'_v = \texttt{GRU}([\mathbf{c}, \mathbf{n}_v], \mathbf{h}_v)$, where, $\mathbf{c}$ is the encoded caption. Similarly, if there is a caption c and a node u such that $\texttt{IoU}(\texttt{U}(u, v), \texttt{B}(c)) \geq 0.5$, where U stands for union of two boxes, we update the edge representation of (u, v) and (v, u) as

$$\mathbf{h}'_{u,v} = \texttt{GRU}([\mathbf{c}, \mathbf{e}_{u,v}], \mathbf{h}_{u,v}) \tag{7}$$

$$\mathbf{h}'_{v,u} = \texttt{GRU}([\mathbf{c}, \mathbf{e}_{v,u}], \mathbf{h}_{v,u}) \tag{8}$$

Graph-level representations. We obtain a graph-level representation $\mathbf{o}^{\text{n}}$ using the node representations as

$$\mathbf{o}^{\text{n}} = \psi\big(\sum_v \sigma\big(f^{\text{n}}(\mathbf{n}_v, \mathbf{h}_v)\big) \odot \psi\big(g^{\text{n}}(\mathbf{n}_v, \mathbf{h}_v)\big)\big) \tag{9}$$

where, f^{n} and g^{n} are two neural networks, ψ is the hyperbolic tangent function, and $\sigma\big(f(\mathbf{n}_v, \mathbf{h}_v)\big)$ is an attention weight vector that decides which nodes are related to the current graph-level representation. Similarly, we obtain a graph-level representation $\mathbf{o}^{\text{e}}$ using edge representations as

$$\mathbf{o}^{\text{e}} = \psi\big(\sum_{u,v} \sigma\big(f^{\text{e}}(\mathbf{e}_{u,v}, \mathbf{h}_{u,v})\big) \odot \psi\big(g^{\text{e}}(\mathbf{e}_{u,v}, \mathbf{h}_{u,v})\big)\big) \tag{10}$$

These two representations provide a summary of the current graph for the Q-network.

Node-type PMF update. To compute a new node-type PMF for node v, we use a GRU and a softmax layer

$$\mathbf{n}'_v = \texttt{softmax}\big(\texttt{GRU}([\mathbf{h}_v, \mathbf{o}^{\text{n}}, \mathbf{o}^{\text{e}}], \mathbf{n}_v)\big) \tag{11}$$

where, the softmax function guarantees that sum of the probability of all node-types for node v, is 1.

Edge-type matrix update. We update the edge-type probability vectors as follows

$$\mathbf{r}_{u,v} = \sigma\big(\mathbf{W}^{(1)} \psi\big(\mathbf{W}^{(2)} \mathbf{x} + \mathbf{b}^{(1)}\big) + \mathbf{b}^{(2)}\big) \tag{12}$$

$$\mathbf{z}_{u,v} = \sigma\big(\mathbf{W}^{(3)} \psi\big(\mathbf{W}^{(4)} \mathbf{x} + \mathbf{b}^{(3)}\big) + \mathbf{b}^{(4)}\big) \tag{13}$$

$$\mathbf{e}'_{u,v} = \mathbf{e}_{u,v} \odot (1 - \mathbf{r}_{u,v}) + (1 - \mathbf{e}_{u,v}) \odot \mathbf{z}_{u,v} \tag{14}$$

where $\mathbf{x} = [\mathbf{h}_{u,v}, \mathbf{n}_v, \mathbf{h}_v, \mathbf{n}_u, \mathbf{h}_u]$, and the weight matrices and biases are trainable parameters. This update grantees that the updated probabilities are between 0 and 1.

Parameter update. We update the parameters of the proposed model after updating the node-type PMFs and the edge-type matrices. For these updates, we minimize the following loss functions

$$\mathcal{L}^{\text{n}} = -\sum_v \mathbf{n}^\star_v \cdot \log(\mathbf{n}'_v) + (1 - \mathbf{n}^\star_v) \cdot \log(1 - \mathbf{n}'_v) \tag{15}$$

$$\mathcal{L}^{\text{e}} = -\sum_{u,v} \mathbf{e}^\star_{u,v} \cdot \log(\mathbf{e}'_{u,v}) + (1 - \mathbf{e}^\star_{u,v}) \cdot \log(1 - \mathbf{e}'_{u,v}) \tag{16}$$

where, $\mathbf{n}^\star_v$ and $\mathbf{e}^\star_{u,v}$ denote the ground-truth for PMF of node v, and edge-type probabilities of edge $e = (u, v)$. That is, $\mathbf{n}^\star_v(m) = 1$ if there is a ground-truth bounding-box u of node-type m such that $\texttt{IoU}(u, v) \geq 0.5$, otherwise $\mathbf{n}^\star_v(m) = 0$. Similarly, $\mathbf{e}^\star_{u,v}(k) = 1$ if there is a ground-truth edge $e = (u', v')$ of type k such that $\texttt{IoU}(\texttt{U}(u, v), \texttt{U}(u', v')) \geq 0.5$, otherwise $\mathbf{e}^\star_{u,v}(k) = 0$.

3 Training Details and Optimization

The discount factor γ is set to 0.85 and ϵ is annealed from 1 to 0.05 during the first 50 epochs, and is fixed after epoch 50. To prevent overfitting, dropout with probability 0.5 and early stopping are applied. D is set to 512. For each image, we use up to 10 captions with a confidence score greater than 1.0. During training, all parameters are tuned except for the weights of the CNN and caption generation component to avoid overfitting. We use default values for all hyper-parameters of the object detector API. Our model takes around two days to train on two NVIDIA Titan X GPUs. We use a region-grounded captioning model from `https://github.com/jcjohnson/densecap` to extract a set of descriptions for the input image, e.g. *a cloudy sky*, *woman holding umbrella*. Each caption c has a bounding-box $\texttt{B}(c)$ and a confidence score. We map the one-hot representation of each word to a semantic space of dimensionality D via a $D \times n$ word embedding matrix, where n is the size of a dictionary which is created using all training captions. To initialize the word embeddings, we apply skip-gram training to all captions, and concatenate the skip-gram vectors with the pretrained GloVe vectors. The last hidden state of a GRU is used to encode a caption. We reset the GRU after presenting each caption.

4 Constructing a Complete PGN

Given an image dataset with ground-truth annotations, we first train an object detector. For this purpose, we use the Tensorflow Object Detection API. We use `faster_rcnn_nas` trained on MS-COCO dataset as the pretrained model. Given an image, the output of the object detector is a set of object bounding-boxes with their objectness scores, classification scores, bounding-box coordinates, and feature vectors.

Each bounding-box v is specified by its coordinates $\texttt{B}(v) = (v_x, v_y, v_{x'}, v_{y'})$, a 512-d feature vector $\mathbf{h}^\circ_v$, and a confidence score $p(v)$, where (v_x, v_y) and $(v_{x'}, v_{y'})$ are the top-left and bottom-right corners of the bounding-box. We use $N = 256$ bounding-boxes per image with the highest objectness scores. We also extract a feature vector for each edge $e = (u, v)$ denoted by $\mathbf{h}^\circ_{u,v}$ from the union of bounding-boxes u and v.

Similar to [11], we obtain a node-type PMF $\mathbf{n}^\circ_v$ for each candidate bounding-box v as follows. We first sort the bounding-boxes based on their confidence score, and apply a bidirectional LSTM as

$$\mathbf{f}_k = \texttt{BiLSTM}([\mathbf{h}^\circ_k, W^{(1)}\hat{\mathbf{n}}^\circ_k], \mathbf{f}_{k-1}) \tag{17}$$

where, $\mathbf{f}_0 = \mathbf{0}$, $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times M}$ is a trainable matrix, and $\hat{\mathbf{n}}^\circ_k \in \mathbb{R}^M$ is the node-type PMF for candidate bounding-box v which is obtained base on the classification scores of bounding-box v. Then, we use an LSTM to decode a node-type distribution $\mathbf{n}^\circ_k$ as

$$\hat{\mathbf{f}}_k = \texttt{LSTM}([\mathbf{f}_k, \mathbf{n}^\circ_{k-1}], \hat{\mathbf{f}}_{k-1}) \tag{18}$$

where $\hat{\mathbf{f}}_0 = \mathbf{0}$, $\mathbf{n}^\circ_k = \texttt{softmax}(\mathbf{W}^{(2)}\hat{\mathbf{f}}_k)$, and $\mathbf{W}^{(2)} \in \mathbb{R}^{M \times d'}$ is a trainable matrix.

Similarly, to get a contextualized representation for edges, we use another bidirectional LSTM as

$$\mathbf{f}^\dagger_k = \texttt{BiLSTM}([\mathbf{f}_k, \mathbf{W}^{(3)}\mathbf{n}_k], \mathbf{f}^\dagger_{k-1}) \tag{19}$$

where, $\mathbf{f}^\dagger_0 = \mathbf{0}$, and $\mathbf{W}^{(3)} \in \mathbb{R}^{d'' \times M}$ is a trainable matrix. Then, we extract a contextualized feature vector from edge (u, v) as $\mathbf{e}^\dagger_{u,v} = \mathbf{W}^{(4)}\mathbf{f}^\dagger_u \odot \mathbf{h}^\circ_{u,v} + \mathbf{W}^{(5)}\mathbf{f}^\dagger_v \odot \mathbf{h}^\circ_{u,v}$, where, $\mathbf{W}^{(4)}, \mathbf{W}^{(5)} \in \mathbb{R}^{d^\circ \times d'^\circ}$. Next, we use a sigmoid layer to decode an edge-type distribution $\mathbf{e}^\circ_{u,v}$ as

$$\mathbf{e}^\circ_{u,v} = \sigma(\mathbf{W}^{(6)}\mathbf{e}^\dagger_{u,v} + \mathbf{b}^{(1)}) \tag{20}$$

where, $\mathbf{W}^{(6)} \in \mathbb{R}^{K \times d^\circ}$, and $\mathbf{b}^{(1)}$ is a trainable bias specific to the decoded object classes of u and v. The d, d', d'', d°, and d'° are set to 512. We train the model by minimizing the sum of the cross entropy for edge-types and cross entropy for node-types. To prevent vanishing gradient problems, highway connections [12] are added to all LSTMs. The output of this section is an initial node-type distribution $\mathbf{n}^\circ_v$ for each node v, and an initial edge-type distribution $\mathbf{e}^\circ_{u,v}$ for each edge $e = (u, v)$.

5 Deep Generative Probabilistic Graph Neural Networks Algorithm

Algorithm 1, provides computational steps in a Deep Generative Probabilistic Graph Neural Network.

Algorithm 1: Deep Generative Graph Neural Networks for Scene Graph Generation

Input : A set of image captions $\mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$
Input : A set of N bounding-boxes
Input : A global image feature vector $\mathbf{g}$
Output Updates the parameters of the models
:

for $u \leftarrow 1$ **to** N **do**
  $\mathbf{d}(u) = 0$
end
Let s be an empty graph
for $i \leftarrow 1$ **to** N **do**
  $\mathcal{A} \leftarrow \{v : \mathbf{d}(v) = 0\} \cup \{\texttt{STOP}\}$
  Generate a random number $z \in (0, 1)$
  if $z < \epsilon$ **then**
    Select a random node a from $\mathcal{A}$
  else
    $a \leftarrow \arg\max_{v \in \mathcal{A}} Q\big(\phi(s + v); \boldsymbol{\theta}\big)$
  end
  if a is not STOP **then**
  Compute reward r, $s' \leftarrow s + a$, $\mathbf{d}(a) \leftarrow 1$
  else break
  $\phi' \leftarrow [\mathbf{g}, \mathbf{p}, \mathbf{d}, \mathbf{h}_1, \ldots, \mathbf{h}_N, \mathbf{n}_1, \ldots, \mathbf{n}_N, \mathbf{o}^{\mathrm{n}}, \mathbf{o}^{\mathrm{e}}]$
  Store transition $(\phi, s, a, r, \phi', s')$
  Sample minibatch of transitions $(\hat{\phi}, \hat{s}, \hat{a}, \hat{r}, \hat{\phi}', \hat{s}')$
  Compute target Q-value $\hat{y}$ ▷ *Eq. 2*
  Update the parameters of the Q-network ▷ *Eq. 3*
  $s \leftarrow s'$, Let s be $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
  for $u \leftarrow 1$ **to** $|\mathcal{V}|$ **do**
    Update $\mathbf{h}_v$ ▷ *Eq. 5*
  end
  for $e = (u, v) \in \mathcal{V} \times \mathcal{V}$ **do**
    Update $\mathbf{h}_{u,v}$ ▷ *Eq. 6*
  end
  for $j \leftarrow 1$ **to** $|\mathcal{C}|$ **do**
    Obtain $\mathbf{c}$ by encoding caption c_j
    if $\texttt{IoU}(\texttt{B}(c_j), \texttt{B}(a)) \geq 0.5$ **then**
      update $\mathbf{h}_a$
    end
    for $u \leftarrow 1$ **to** $|\mathcal{V}|$ **do**
      if $\texttt{IoU}(\texttt{U}(u, a), \texttt{B}(c_j)) \geq 0.5$ **then**
        update $\mathbf{h}_{u,a}$ and $\mathbf{h}_{a,u}$ ▷ *Eq. 7, 8*
      end
    end
  end
  Obtain graph-level representations $\mathbf{o}^{\mathrm{n}}$, $\mathbf{o}^{\mathrm{e}}$ ▷ *Eq. 9*
  for $u \leftarrow 1$ **to** $|\mathcal{V}|$ **do**
    Update $\mathbf{n}_v$ ▷ *Eq. 11*
  end
  for $e = (u, v) \in \mathcal{V} \times \mathcal{V}$ **do**
    Update $\mathbf{e}_{u,v}$ ▷ *Eq. 14*
  end
  Update parameters of the PGN model ▷ *Eq. 15, 16*
end