Dynamic Gated Graph Neural Networks for Scene Graph Generation

Mahmoud Khademi and Oliver Schulte

Simon Fraser University, Burnaby, BC, Canada
{mkhademi,oschulte}@sfu.ca

Abstract. We describe a new deep generative architecture, called Dynamic Gated Graph Neural Networks (D-GGNN), for extracting a scene graph for an image, given a set of bounding-box proposals. A scene graph is a visually-grounded digraph for an image, where the nodes represent the objects and the edges show the relationships between them. Unlike the recently proposed Gated Graph Neural Networks (GGNN), the D-GGNN can be applied to an input image when only partial relationship information, or none at all, is known. In each training episode, the D-GGNN sequentially builds a candidate scene graph for a given training input image and labels additional nodes and edges of the graph. The scene graph is built using a deep reinforcement learning framework: states are partial graphs, encoded using a GGNN, actions choose labels for node and edges, and rewards measure the match between the ground-truth annotations in the data and the labels assigned at a point in the search. Our experimental results outperform the state-of-the-art results for scene graph generation task on the Visual Genome dataset.

Keywords: Gated Graph Neural Networks · Scene Graph Generation.

1 Introduction: Scene Graph Generation

Visual scene understanding is one of the most important goals in computer vision. Over the last decade, there has been great progress in related tasks such as image classification [14,10,29], image segmentation [19], object detection [24], and image caption generation [12,33,31,35]. However, understanding a scene is not just recognizing the individual objects in the scene. The *relationships* between objects also play an important role in visual understating of a scene.

To capture objects and their relationships in a given scene, previous work proposed to build a structured representation called scene graph [13,20,15]. A scene graph for an image is a visually-grounded labeled digraph, where the nodes represent the objects and the edges show the relationships between them (see Figure 1). The visual information represented by a scene graph is useful in applications such as visual question answering [30] and fine-grained recognition [36]. This paper presents a new neural architecture that generates a scene graph for a given image.

An effective scene graph generation model needs to consider visual contextual information as well as domain priors. For example, if we know that a node has

an incoming edge from another node which represents a `man` and the type of the edge is `riding`, then the type of the node is likely `horse` or `bicycle`. Clues from the relational context can be represented by leveraging models such as the recently proposed Gated Graph Neural Networks (GGNN) [16], which can learn a latent feature vector (embedding, representation) for each node from graph-structured inputs. However, to apply the GGNN model, the structure of the input digraph and the type of each edge must be known, whereas the structure and edge-types must be *inferred* in the scene graph generation task. In this work, we propose a new deep generative architecture, called Dynamic Gated Graph Neural Networks (D-GGNN), to perform this inference. Unlike GGNN, the D-GGNN can be applied to an input image to build a scene graph, without assuming that the structure of the input digraph is known.

In each training episode, the D-GGNN constructs a candidate graph structure for a given training input image by sequentially adding new nodes and edges. The D-GGNN builds the graph in a *deep reinforcement learning (RL) framework*. In each training step, the graph built so far is the current state. To encode the current graph in a state feature vector, the D-GGNN leverages the power of a GGNN to exploit the relational context information of the input image, and combines the GGNN with an attention mechanism. Given the current graph, and a current object, D-GGNN selects two actions: i) a new neighbor and its type ii) the type of the new edge from the current object to the new object. The reward for each action is a function of how well the predicted types measure the ground-truth labels. At the test time, the D-GGNN builds a graph for a given input by sequentially selecting the best actions with the greatest expected accuracies (Q-values).

In summary, the contributions of this paper are as follows: i) We propose a new deep generative architecture that uses reinforcement learning to generate from unstructured inputs a heterogeneous graph, which represents the input information with multiple types of nodes and edges. Unlike the recently proposed Gated Graph Neural Networks (GGNN), the D-GGNN can be applied to an input without requiring that the structure of the scene graph is known in advance. ii) We apply the D-GGNN to the scene graph generation task. The D-GGNN can exploit domain priors and the relational context of objects in an input image to generate more accurate scene graphs than previous work. iii) Our model scales to predict thousands of predicates and object classes. The experiments show that our model significantly outperforms the state-of-the-art models for scene graph generation task on the Visual Genome dataset.

2 Related Work

Scene Graph Generation. In [28], the authors introduced a rule-based and a classifier based method for scene graph generation from a natural language scene description. [11] proposed a model based on a conditional random field that reasons about various groundings of potential scene graphs for an image. Recently, [20] proposed a model to detect a relatively large set of relationships

using language priors from semantic word embeddings. In [32], the authors developed a model for scene graph generation which uses a recurrent neural network and learns to improve its predictions iteratively through message passing across the scene graph. After a few steps the learned features are classified. However, their model suffers from a class imbalance since for many pairs of objects, there is no edge between them.

More recently, [15] proposed a neural network model, called Multi-level Scene Description Network (MSDN), to address the object detection, region captioning, and scene graph generation tasks jointly. MSDN aligns object, phrase, and caption regions with a dynamic graph using spatial and semantic links. Then, it applies a feature refining schema to send messages across features of the phrase, objects, and caption nodes via the graph.

Most closely related to our work, [18] proposed a model for visual relationship and attribute detection based on reinforcement learning. In their method, as a state feature representation, they used the embedding of the last two relationships and attributes that were searched during the graph construction. This fixed-length representation will lead to limited representational power, since the resulting representation depends on the order of the actions, that is, the order that the algorithm selects node and edges. For example, this method may generate different representations using different order of actions for the same graph. In contrast, our state feature representation is based on a Gated Graph Neural Network (GGNN) which is tailored towards graph-structured input data and extracts features from the entire graph.

Graph Neural Networks. Several graph neural network models have been proposed for feature learning from graph-structured inputs [27,9,16,7,5]. [26] and [2] summarized recent work on graph neural networks in depth. Graph neural networks have been used for various range of tasks that need rich relational structure such as visual scene understanding [25] and learning to represent programs [1]. Recently, a few work proposed generative models of graphs [3,34,17,4]. To the best of our knowledge, our work is the first work that uses reinforcement learning to build a generative graph neural network architecture.

3 Background: Gated Graph Neural Networks

The state feature vectors in our RL framework encode an entire candidate graph. The first step in building the graph encoding is to encode node information by feature vectors (embedding, representations) such that each node feature vector takes into account contextual information from the neighbor nodes. For example, in scene graph prediction task, if we know that a node has an incoming edge from another node which represents a `man` and the type of the edge is `riding`, then the type of the node is likely `horse` or `bicycle`. There are various approaches for finding node embeddings which can be applied within our RL framework. In this work, we utilize the Gated Graph Neural Network (GGNN) [16], which is a recent state-of-the-art approach.

Formally, a GGNN takes as input a heterogeneous directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each directed edge $e = (v, u) \in \mathcal{E}$ has an edge-type $\texttt{type}(e) \in \{1, \ldots, K\}$, where K is the number of edge-types (classes, labels). The given graph may also contain node-types $\texttt{type}(v) \in \{1, \ldots, M\}$ for each node v, where M is the number of node-types (classes, labels). Given the graph structure and the edge types, a GGNN iteratively computes a new node embedding $\mathbf{h}_v^{(t)} \in \mathbb{R}^d$, $t = 1, \ldots, T$, for each node v via a propagation model, where d is the embedding size.

For each node v, a node representation must take into account the information from every linked neighbor of v. Let E_k be the adjacency matrix for edge-type k. That is, $E_k(u, v) = 1$, if there is an edge with type k from node u to node v, otherwise $E_k(u, v) = 0$. The matrix E_k determines how nodes in the graph communicate with each other via edges of type k. We write $\mathbf{x}_v \in \mathbb{R}^M$ for the one-hot representation of node v's type.

The recurrence of the propagation for computing node embeddings $\mathbf{h}_v^{(t)} \in \mathbb{R}^d$ is initialized with the padded one-hot representation of node v's type, i.e. $\mathbf{h}_v^{(0)} = (\mathbf{x}_v, \mathbf{0}) \in \mathbb{R}^d$. The update equations are

$$\mathbf{a}_v^{(t)} = \sum_{k=1}^{K} W_k H^{(t-1)} E_{k:v} \tag{1}$$

$$\mathbf{h}_v^{(t)} = \texttt{GRU}\big(\mathbf{a}_v^{(t)}, \mathbf{h}_v^{(t-1)}\big). \tag{2}$$

The $H^{(t)} \in \mathbb{R}^{d \times |\mathcal{V}|}$ is a matrix with node representations $\mathbf{h}_v^{(t)}$, $v = 1, \ldots, |\mathcal{V}|$, as its column vectors, $W_k \in \mathbb{R}^{d \times d}$ is a weight matrix for edges of type k that we learn, and $E_{k:v}$ is the v'th column of E_k. The term $H^{(t-1)} E_{k:v}$ represents the sum of latent feature representations for all nodes u that are linked to node v by an edge of type k. Thus, $\mathbf{a}_v^{(t)}$ combines activations from incoming edges of all types for node v, aggregating *messages* from all nodes that have an edge to v. The GGNN first computes the node activations $\mathbf{a}_v^{(t)}$ for each node v. Then, a Gated Recurrent Unit (GRU) is used to update node representations for each node by incorporating information from the previous time-step.

The computation defined by the above equations are repeated for a fixed number of time steps T. The choice of T depends on the task. For each node the state vector from the last time step is used as the node representation. In many real-world applications, the edges and edge-types are not given as part of the input, which limits the applicability of the GGNN model. In this paper, we extend GGNN to infer edges and edge-types.

4 New Reinforcement Learning Architecture for Graph Structure Generation

Given an input image and a set of candidate bounding-boxes $\mathcal{P}$, our goal is to find a labelled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \subseteq \mathcal{P}$, and assign the edge-types and node-types to it jointly such that $\mathcal{G}$ represents contextual information of the given input. The algorithm that we describe here can be applied for extracting

relational information for a variety of input data sources. However, our presentation focuses on scene graph generation, the target application of this paper.

4.1 Initial Information Extraction

We extract initial information to constrain the reinforcement learning graph construction.

Global Type Information. We extract the following information from the training set to model type constraints in the target domain.

1. A set of M node types.
2. For each ordered pair of node-types i and j, a set of possible edge-types $\texttt{e-types}(i, j)$. For example, we may find that $\texttt{e-types}(\texttt{man}, \texttt{horse}) = \{\texttt{riding}, \texttt{next to}, \texttt{on}, \texttt{has}\}$.

Image Node and Type Candidates. We extract the following information from each training image.

1. A set $\mathcal{P}$ of *candidate nodes*.
2. For each candidate node $v \in \mathcal{P}$:
   (a) A confidence score $s(v)$ that measures how likely v is to be a node in the scene graph.
   (b) A set of candidate node-types $\texttt{n-types}(v) \subseteq \{1, \ldots, M\}$.
   (c) A feature vector $\hat{\mathbf{x}}_v$.
   (d) A *vicinity* set $\texttt{vic}(v)$ is given such that $u \notin \texttt{vic}(v)$ implies that $e = (v, u)$ is not an edge in the scene graph.

This information is extracted as follows. Objects (nodes) are represented by bounding-boxes. Each bounding-box v is specified by a confidence score $s(v)$ and its coordinates $(v_x, v_y, v_{x'}, v_{y'})$, where (v_x, v_y) and $(v_{x'}, v_{y'})$ are the top left and bottom right corners of the bounding-box, respectively. Given an image dataset with ground truth annotations, we train an *object detector*, using the Tensorflow Object Detection API from `https://github.com/tensorflow/models/tree/master/research/object_detection`. We use `faster_rcnn_nas` trained on MS-COCO dataset as the pretrained model. Also, we use default values for all hyper-parameters of the object detector API.

For each input image, the object detector outputs a set of object bounding-boxes with their objectness scores, classification scores, and bounding-box coordinates. We used 100 bounding-box per image with the highest objectness scores as the candidate nodes $\mathcal{P}$, and their objectness scores as the confidence scores. For each bounding box, the classification scores rank the possible node-types. The set of candidate object categories $\texttt{n-types}(v)$ comprises the highest-scoring types. For example, $\texttt{n-types}(v)$ can be $\{\texttt{man}, \texttt{boy}, \texttt{skier}\}$.

To extract a feature vector $\hat{\mathbf{x}}_v$ for each bounding-box $v \in \mathcal{P}$, we first feed the image to the 152-layer ResNet [10], pretrained on ImageNet [6], and obtain 1024 feature maps of size 14×14 from layer `res4b35x`. Then, we apply a Region

of Interest pooling layer [8], based on the coordinates of the bounding-box, to get 1024 feature maps of size 7×7. Next, we use two fully-connected layers with ReLU activation function to get a 512-dimensional feature vector. Finally the set $\texttt{vic}(v)$ is a subset of bounding-boxes in $\mathcal{P}$ which are spatially close to bounding-box v ($u_w = u'_x - u_x$ and $u_h = u'_y - u_y$):

$$\texttt{vic}(v) = \{u : u \neq v, |u_x - v_x| < 0.5(u_w + v_w), |u_y - v_y| < 0.5(u_h + v_h)\}$$

4.2 Dynamic Gated Graph Neural Networks

Figure 1 shows the architecture of a D-GGNN for scene graph generation task. The algorithm simultaneously builds the graph and assigns node-types and edge-types using Deep Q-Learning [21,22]. The full algorithm is presented in Algorithm 1. The Q-function maps a state-action pair to an expected cumulative reward value. Actions and states are defined as follows.

Actions. The node with the highest confidence score is the starting node v, and the type with the highest classification score its type l. At each time step t, given a current subject node $v \in \mathcal{P}$ with node type l, we expand the scene graph by adding a new object node and a new edge, and selecting the type of the new node and the type of the new edge. These choices represent two actions.

1. Select a node u_t and a node-type a_t from the pairs

$$\mathcal{A} = \{(a, u) : u \in \texttt{vic}(v) - \texttt{prv}(v), a \in \texttt{n-types}(u)\} \cup \{\texttt{STOP}\} \quad (3)$$

   where $\texttt{prv}(v)$ denotes the set of previously selected nodes for neighbors of v, and STOP is a special action that indicates the end of searching for neighbors of node v. We first select the node type. If there are multiple nodes in $\mathcal{A}$ with the same type as the selected type, we randomly select one of them.
2. Select an edge-type b_t from

$$\mathcal{B} = \texttt{e-types}(l_t, a_t). \quad (4)$$

States. The current RL *state* is a (partial) scene graph denoted by s_t. The new state (graph) is obtained as $s_{t+1} = s_t + u_t + e_t$, where $s_t + u_t$ is obtained by adding node u_t with type a_t to graph s_t, and $s_t + u_t + e_t$ is obtained by adding edge $e_t = (v, u_t)$ with type b_t to graph $s_t + u_t$. A GGNN computes a state representation for state s_t. For each node v' of s_t, we initialize the node representation as $\mathbf{h}^{(0)}_{v'} = W(\mathbf{x}_{v'}, \hat{\mathbf{x}}_{v'})$, where $W \in \mathbb{R}^{d \times (512+M)}$ is a trainable matrix; so instead of padding the one-hot node type vector $\mathbf{x}_{v'}$ with 0, we concatenate it with the ResNet feature vector $\hat{\mathbf{x}}_{v'}$. Then, the state is represented by a vector $\texttt{GGNN}(s_t)$ computed as follows

$$\texttt{GGNN}(s_t) = \tanh\Big(\sum_{v'} \sigma\big(f(\mathbf{x}_{v'}, \mathbf{h}^{(T)}_{v'})\big) \odot \tanh\big(g(\mathbf{x}_{v'}, \mathbf{h}^{(T)}_{v'})\big)\Big) \quad (5)$$

where, f and g are two neural networks, σ is the sigmoid function. The vector $\sigma\big(f(\mathbf{x}_{v'}, \mathbf{h}^{(T)}_{v'})\big)$ is a soft attention mechanism that decides which nodes are relevant to generate the vector representation for state s_t. Intuitively, $\sigma\big(f(\mathbf{x}_{v'}, \mathbf{h}^{(T)}_{v'})\big)$

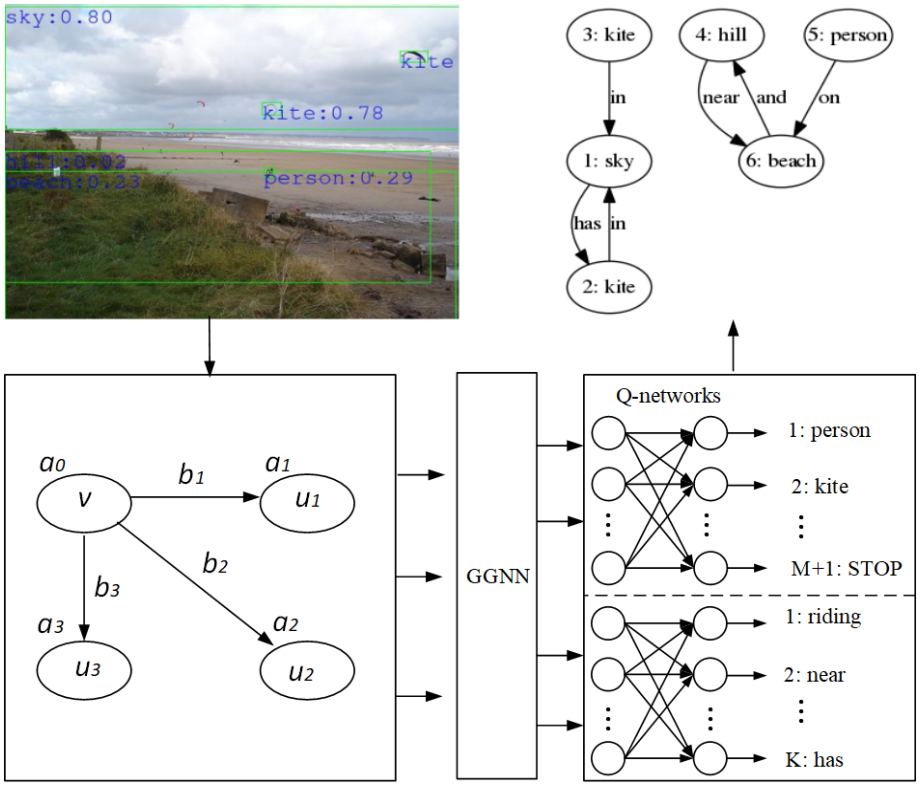

Fig. 1. Dynamic Gated Graph Neural Networks for scene graph generation task. Given an image and its candidate object bounding-boxes, we simultaneously build the graph and assign node-types and edge-types to the nodes and edges in a Deep Q-Learning framework. We use two separate Q-networks to select the actions, i.e. node-types and edge-types. We use two fully-connected layers with ReLU activation function to implement each Q-network. The input to the Q-networks is the concatenation of the GGNN representation of the current graph and a global feature vector from the image. The search for the scene graph continues with the next node in a breadth-first search order.

represents the degree of importance of node v' to represent the current state, i.e. the contribution of node v' for selecting the next node-types and edge-types.

Rewards. Given the current object bounding-box v and the new object bounding-box u, the reward functions for taking actions a (non-`STOP`) and b at state s are defined as follows (`IoU` stands for Intersection over Union):

1. $r(a, u, s) = +1$ if there exists a ground truth bounding-box h with object category a such that $\texttt{IoU}(u, h) \geq 0.5$, otherwise $r(a, u, s) = -1$
2. $r'(b, v, u, s) = +1$ if there exists two ground truth bounding-boxes h and h' such that $\texttt{IoU}(v, h) \geq 0.5$, $\texttt{IoU}(u, h') \geq 0.5$ and the type of edge $e = (v, u)$ is b, otherwise $r'(b, v, u, s) = -1$.

Temporal-Difference Training of Q-networks. We use two separate Q-networks to select the actions: node-type Q-network and edge-type Q-network. We use two fully-connected layers with ReLU activation function to implement each Q-network. The input to each Q-network is $\phi_t = \phi(s_t) = \big(\texttt{GGNN}(s_t), \mathbf{x}\big)$, where $\mathbf{x}$ is a *global image feature vector* extracted from the last layer of the 152-layer ResNet [10]. The global feature vector adds contextual information of the image to the current state information.

After each 5000 steps, node-type Q-network trainable parameters $\theta^{(t)}$ and edge-type Q-network trainable parameters $\theta'^{(t)}$ are copied to $\hat{\theta}^{(t)}$ and $\hat{\theta}'^{(t)}$ which are used to compute the target Q-values for the node-type and edge-type Q-networks, respectively. This helps stabilize the optimization.

To reduce correlation between samples and keep experiences from the past episodes, we use an experience replay technique [21,22]. To update the parameters of the Q-networks, we choose a random minibatch from the replay memory. Let $(\phi_j, s_j, a_j, b_j, r_j, r'_j, \phi_{j+1}, s_{j+1})$ be a random transition sample, and v_{j+1} the subject node in state s_{j+1}. The target Q-values for both networks are obtained as follows:

$$y_j = r_j + \gamma \max_{(a,u)\in\mathcal{A}_{j+1}} Q\big(\phi(s_{j+1}+u); \hat{\theta}^{(t)}\big) \tag{6}$$

$$y'_j = r'_j + \gamma \max_{b\in\mathcal{B}_{j+1}} Q\big(\phi(s_{j+1}+u^{\dagger}_{j+1}+e^{\dagger}_{j+1}); \hat{\theta}'^{(t)}\big) \tag{7}$$

where, $\mathcal{A}_{j+1}$ and $\mathcal{B}_{j+1}$ are actions that can be taken in state s_{j+1}, $u^{\dagger}_{j+1}$ is the node that maximizes 6, and $e^{\dagger}_{j+1} = (v_{j+1}, u^{\dagger}_{j+1})$.

Finally, the parameters of the model are updated as follows:

$$\theta^{(t+1)} = \theta^{(t)} + \alpha\Big(y_j - Q\big(\phi(s_j+u^{\dagger}_j); \theta^{(t)}\big)\Big)\nabla_{\theta} Q\big(\phi(s_j+u^{\dagger}_j); \theta^{(t)}\big) \tag{8}$$

$$\theta'^{(t+1)} = \theta'^{(t)} + \alpha\Big(y'_j - Q\big(\phi(s_j+u^{\dagger}_j+e^{\dagger}_j); \theta'^{(t)}\big)\Big)\nabla_{\theta'} Q\big(\phi(s_j+u^{\dagger}_j+e^{\dagger}_j); \theta'^{(t)}\big) \tag{9}$$

Search Strategy. The search for the scene graph continues with the next node in a breadth-first search order. The algorithm continues until all nodes that are reachable from the starting node have been visited. If some unvisited nodes are remained, we repeat the search for the next component, until the highest confidence score of the unvisited nodes is less than a threshold β, or a maximum number of steps n is reached. This allows the search to generate disconnected graphs. To train the RL 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.

4.3 Optimization and Implementation Details

We used RMSProp with minibatch size of 100. The learning rate α was 0.00001. The number of iterations for the GGNN is set to $T = 2$. The maximum number

Algorithm 1: Dynamic Gated Graph Neural Networks: Deep Q-learning

Hyperparameters are described in the text
Input: A set of bounding-boxes $\mathcal{P}$ of an image with their feature vectors
Input: A global feature vector $\mathbf{x}$ extracted from the image
Result: Updates the parameters of the model (the weight matrices of the GGNN for each edge-type and parameters of both Q-networks)

$\mathcal{C} \leftarrow \mathcal{P}$ ▷ *Mark all nodes as unvisited ($\mathcal{C}$ is the set of all unvisited nodes)*
Let s_0 be an empty graph, $t \leftarrow 0$ ▷ *Initialize the state (graph)*
while $\mathcal{C}$ *is not empty* **do**
  From $\mathcal{C}$ select node v with the highest confidence score and its node-type l
  if $s(v) < \beta$ *or* $t > n$ **then** break
  Add node v to s_t, and compute $\phi(s_t) = \big(\texttt{GGNN}(s_t), \mathbf{x}\big)$ ▷ *Equation 5*
  $q = \text{Queue}(), q.\text{enqueue}((v, l))$ ▷ *Make an empty queue and add (v, l) to it*
  while q *is not empty* **do**
    $(v, l) = q.\text{dequeue}()$, $\texttt{prv}(v) \leftarrow \emptyset$ ▷ *Remove an element from the queue*
    repeat
      Generate a random number $z \in (0, 1)$ with uniform distribution
      if $z < \epsilon$ **then**
        $\mathcal{A} \leftarrow \{(a, u) : u \in \texttt{vic}(v) - \texttt{prv}(v), a \in \texttt{n-types}(u)\} \cup \{\texttt{STOP}\}$
        Select a random node-type a_t and its node u_t from $\mathcal{A}$
        if a_t *is not* `STOP` **then**
          Select a random edge-type b_t from $\mathcal{B} = \texttt{e-type}(l, a_t)$
        end
      else
        Select $(a_t, u_t) = \arg\max_{(a,u)\in\mathcal{A}} Q\big(\phi(s_t + u); \theta\big)$
        if a_t *is not* `STOP` **then**
          Select $b_t = \arg\max_{b\in\mathcal{B}} Q\big(\phi(s_t + u_t + e_t); \theta'\big)$
        end
      end
      if a_t *is not* `STOP` **then**
        Compute rewards r_t and r'_t
        $s_{t+1} \leftarrow s_t + u_t + e_t$
      else
        $s_{t+1} \leftarrow s_t$
      end
      $\phi_{t+1} \leftarrow \phi(s_{t+1}) = \big(\texttt{GGNN}(s_{t+1}), \mathbf{x}\big)$ ▷ *Equation 5*
      Store transition $(\phi_t, s_t, a_t, b_t, r_t, r'_t, \phi_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
      Sample minibatch of transitions $(\phi_j, s_j, a_j, b_j, r_j, r'_j, \phi_{j+1}, s_{j+1})$ from replay memory $\mathcal{D}$
      Compute target Q-values y_j and y'_j ▷ *Equations 6 and 7*
      Update the parameters of the model ▷ *Equations 8 and 9*
      if u_t *is in* $\mathcal{C}$ **then**
        $q.\text{enqueue}((u_t, a_t))$ ▷ *Add (u_t, a_t) to the queue*
      end
      $\texttt{prv}(v) = \texttt{prv}(v) \cup \{u_t\}$, $t \leftarrow t + 1$ ▷ *Add node u_t to the set $prv(v)$*
    until a_t *is* `STOP`
    $\mathcal{C} \leftarrow \mathcal{C} - \big(\{v\} \cup \texttt{vic}(v)\big)$ ▷ *Mark node v and its vicinity as visited*
  end
end

of steps to construct a graph for an image is set to $n = 300$. 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. The embedding size d is set to 512.

To avoid overfitting, we used a low-rank bilinear method [23] to reduce the rank of the weight matrix for each edge type. This technique effectively reduces the number of trainable parameters. We train our model for 100 epochs. Our model takes around two weeks to train on two NVIDIA Titan X GPUs.

5 Experiments

We introduce the datasets, baseline models and evaluation metrics that we use in our experiments. Then, the experimental results are presented and discussed.

5.1 Data, Metrics, and Baseline Models

Datasets. The Visual Genome (VG) dataset [13] contains 108,077 images. We used Visual Genome version 1.4 release. This release contains cleaner object annotations [32]. Annotations provide subject-predicate-object triples. A triplet means that there is an edge between the subject and the object and the type of the edge is indicated by the predictate. We experiment with two variations of the VG dataset.

Following [32], we use the most frequent 150 object categories and 50 predicates for scene graph prediction task. We call this variation `VG1.4-a`. 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, respectively.

Following [18], we use the most frequent 1750 object categories and 347 predicates for scene graph prediction task. We call this variation `VG1.4-b`. Following [18], we used 5000 images for validation, and 5000 for testing. This variation of the data allows large scale evaluation of our model.

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 objectness confidence scores and the Q-values of the selected predicates. Following [32], we evaluate our model on `VG1.4-a` based on three tasks as follows:

1. 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.
2. 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.
3. 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

Intersection over Union (IoU) overlap with a bounding-box annotation with the same category.

Following [18], we evaluate our model on `VG1.4-b` based on two tasks as follows:

1. Relationship phrase detection (REL-PHRASE-DET): to predict a phrase (subject-predicate-object), such that the detected box of the entire relationship has at list 0.5 IoU overlap with a bounding-box annotation.
2. Relationship detection (REL-DET): to predict a phrase (subject-predicate-object), such that detected boxes of the subject and object have at least 0.5 IoU overlap with the corresponding bounding-box annotations.

Baseline Models. We compare our model with several baseline models including the state-of-the-art models [18,32]. Faster R-CNN uses R-CNN to detect object proposals, while CNN+TRPN trains a separate region proposal network (RPN) on `VG1.4-b`.

[20] uses language priors from semantic word embeddings, while [32] 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.

VRL [18] detects visual relationship and attributes based on a reinforcement learning framework. To extract a state feature, they used the embedding of the last two relationships and attributes that were searched during the graph construction.

5.2 Results and Discussion

Tables 1 and 2 report our experimental results on `VG1.4-a` and `VG1.4-b` datasets, respectively. CNN+RPN, Faster R-CNN, and CNN+TRPN train independent detectors for object and predicate classes. As a result, they cannot exploit relational information in the given image. D-GGNN outperforms the state-of-the art models (VRL [18] and [32]) for scene graph generation.

Both D-GGNN and the message passing approach [32] leverage the power of RNNs to learn a feature vector for each bounding-box. However, the message passing suffers from imbalanced classification problem (often there is no edge between many pairs of objects).

Both D-GGNN and VRL use deep Q-learning to generate the scene graph. However, the representational power of VRL is limited, since it represents a state by the embedding of the last two relationships and attributes that were searched during the graph construction. In contrast, our state feature representation is based on a GGNN which exploits contextual clues from the entire graph to more effectively represent objects and their relationships.

Figure 2 and 3 illustrate some scene graphs generated by our model. The D-GGNN predicts a rich semantic representation of the given image by recognizing objects, their locations, and relationships between them. For example, D-GGNN can correctly detect spatial relationships ("trees behind fence", "horse near water", "building beside bus"), parts of objects ("bus has tire", "woman has leg"), and interactions ("man riding motorcycles", "man wearing hat").

6 Conclusions

We have presented a new model for generating a heterogeneous graph from unstructured inputs using reinforcement learning. Our model builds on the recently proposed GGNN model for computing latent node representations that combines relational and feature information. Unlike the recently proposed GGNN, the D-GGNN can be applied to an input when the structure of the input digraph is not known in advance. The target application in this paper was the scene graph generation task. The experiments on Visual Genome dataset show that D-GGNN significantly outperforms the state-of-the-art models for scene graph generation task.

Table 1. Experimental Results of the predicate classification, scene graph classification, and scene graph generation tasks on the `VG1.4-a` dataset. We compare with visual relationship detection with language priors [20], and scene graph generation by iterative message passing [32]

	PRED-CLS		SG-CLS		SG-GEN	
Model	R@50	R@100	R@50	R@100	R@50	R@100
[20]	27.88	35.04	11.79	14.11	00.32	00.47
[32]	44.75	53.08	21.72	24.38	03.44	04.24
D-GGNN	**46.85**	**55.63**	**23.80**	**26.78**	**06.36**	**07.54**

Table 2. Experimental Results of the relationship phrase detection and relationship detection tasks on the `VG1.4-b` dataset.

	REL-PHRASE-DET		REL-DET	
Model	R@100	R@50	R@100	R@50
CNN+RPN [29]	01.39	01.34	01.22	01.18
Faster R-CNN [24]	02.25	02.19	-	-
CNN+TRPN [24]	02.52	02.44	02.37	02.23
[20]	10.23	09.55	07.96	06.01
VRL [18]	16.09	14.36	13.34	12.57
D-GGNN	**18.21**	**15.78**	**14.85**	**14.22**

Acknowledgements

This research was supported by a Discovery Grant to the senior author from the Natural Sciences and Engineering Council of Canada. The Titan X GPUs used for this research were donated by the NVIDIA Corporation.

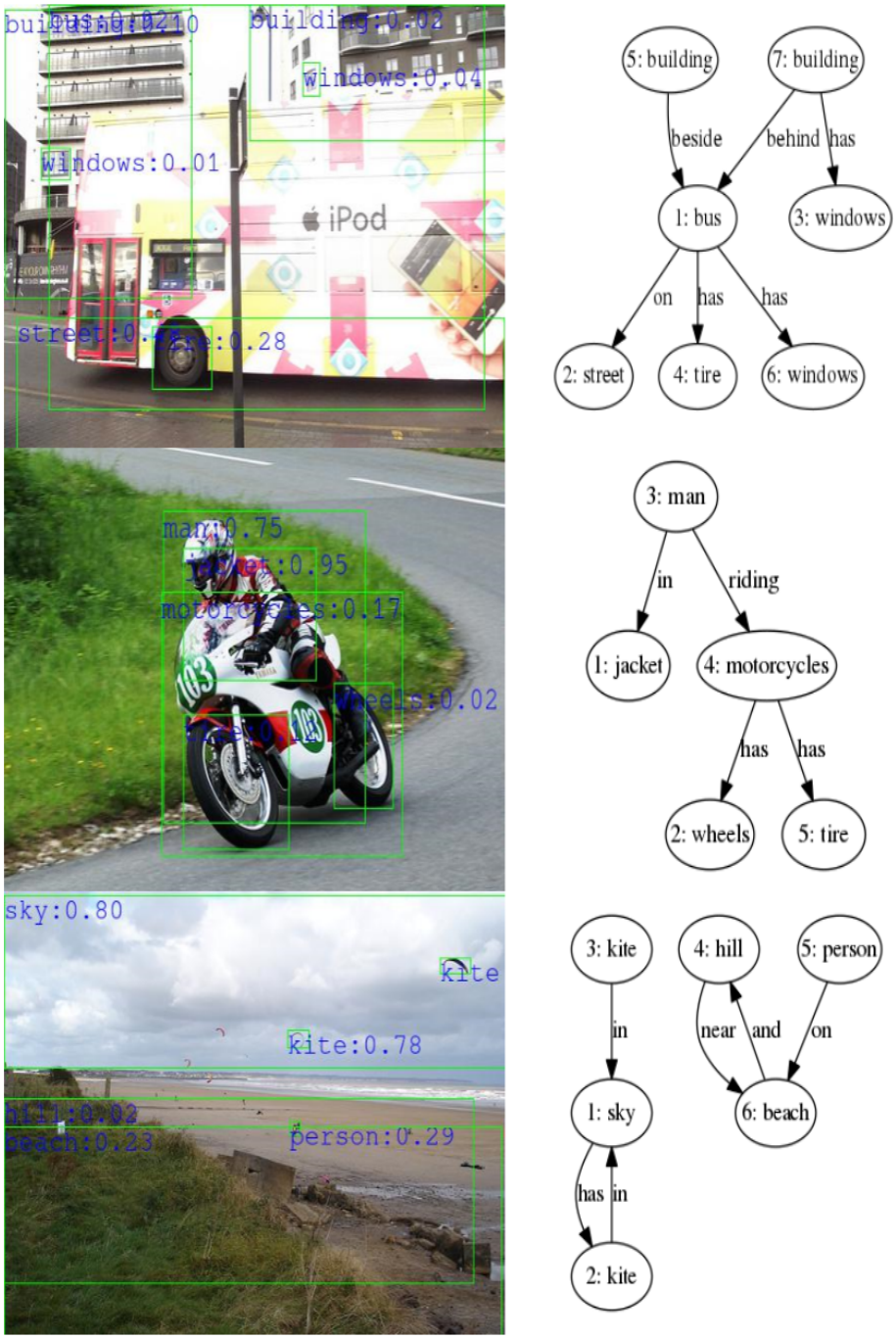

Fig. 2. Some scene graphs generated by our model.

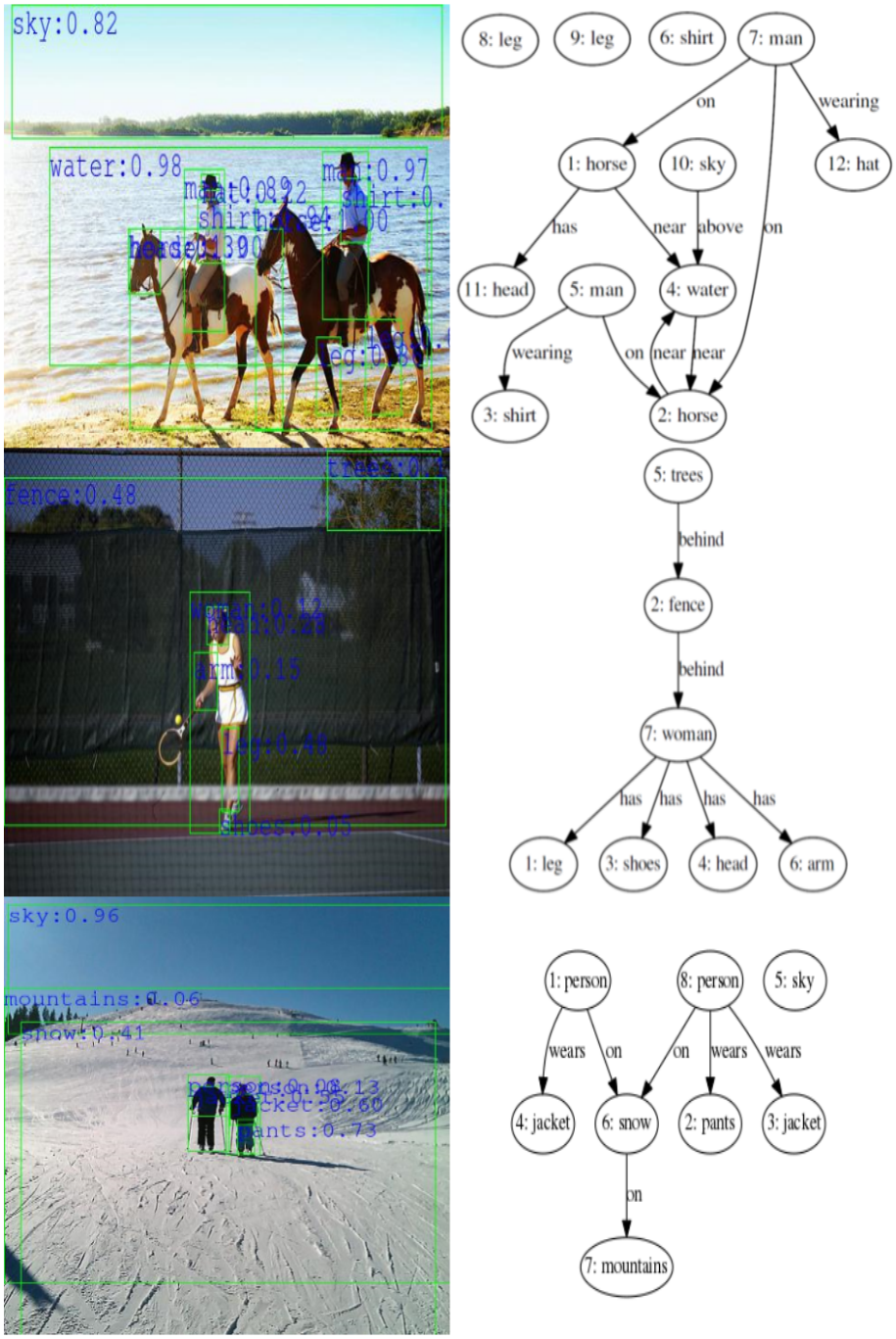

Fig. 3. Some scene graphs generated by our model.

References

1. Allamanis, M., Brockschmidt, M., Khademi, M.: Learning to represent programs with graphs. arXiv preprint arXiv:1711.00740 (2017)
2. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al.: Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261 (2018)
3. Bojchevski, A., Shchur, O., Zügner, D., Günnemann, S.: Netgan: Generating graphs via random walks. arXiv preprint arXiv:1803.00816 (2018)
4. De Cao, N., Kipf, T.: Molgan: An implicit generative model for small molecular graphs. arXiv preprint arXiv:1805.11973 (2018)
5. Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Advances in Neural Information Processing Systems. pp. 3844–3852 (2016)
6. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. pp. 248–255. IEEE (2009)
7. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. arXiv preprint arXiv:1704.01212 (2017)
8. Girshick, R.: Fast r-cnn. arXiv preprint arXiv:1504.08083 (2015)
9. Gori, M., Monfardini, G., Scarselli, F.: A new model for learning in graph domains. In: Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on. vol. 2, pp. 729–734. IEEE (2005)
10. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)
11. Johnson, J., Krishna, R., Stark, M., Li, L.J., Shamma, D., Bernstein, M., Fei-Fei, L.: Image retrieval using scene graphs. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3668–3678 (2015)
12. Karpathy, A., Fei-Fei, L.: Deep visual-semantic alignments for generating image descriptions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3128–3137 (2015)
13. 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. arXiv preprint arXiv:1602.07332 (2016)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
15. Li, Y., Ouyang, W., Zhou, B., Wang, K., Wang, X.: Scene graph generation from objects, phrases and region captions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1261–1270 (2017)
16. Li, Y., Tarlow, D., Brockschmidt, M., Zemel, R.: Gated graph sequence neural networks. arXiv preprint arXiv:1511.05493 (2015)
17. Li, Y., Vinyals, O., Dyer, C., Pascanu, R., Battaglia, P.: Learning deep generative models of graphs. arXiv preprint arXiv:1803.03324 (2018)
18. Liang, X., Lee, L., Xing, E.P.: Deep variation-structured reinforcement learning for visual relationship and attribute detection. In: Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on. pp. 4408–4417. IEEE (2017)

19. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)
20. Lu, C., Krishna, R., Bernstein, M., Fei-Fei, L.: Visual relationship detection with language priors. In: European Conference on Computer Vision. pp. 852–869. Springer (2016)
21. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
22. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
23. Pirsiavash, H., Ramanan, D., Fowlkes, C.C.: Bilinear classifiers for visual recognition. In: Advances in neural information processing systems. pp. 1482–1490 (2009)
24. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
25. Santoro, A., Raposo, D., Barrett, D.G., Malinowski, M., Pascanu, R., Battaglia, P., Lillicrap, T.: A simple neural network module for relational reasoning. In: Advances in neural information processing systems. pp. 4967–4976 (2017)
26. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: Computational capabilities of graph neural networks. IEEE Transactions on Neural Networks **20**(1), 81–102 (2009)
27. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2009)
28. Schuster, S., Krishna, R., Chang, A., Fei-Fei, L., Manning, C.D.: Generating semantically precise scene graphs from textual descriptions for improved image retrieval. In: Proceedings of the fourth workshop on vision and language. pp. 70–80 (2015)
29. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
30. Teney, D., Liu, L., van den Hengel, A.: Graph-structured representations for visual question answering. CoRR, abs/1609.05600 **3** (2016)
31. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. CoRR **abs/1411.4555** (2014), `http://arxiv.org/abs/1411.4555`
32. Xu, D., Zhu, Y., Choy, C.B., Fei-Fei, L.: Scene graph generation by iterative message passing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. vol. 2 (2017)
33. Xu, K., Ba, J., Kiros, R., Courville, A., Salakhutdinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. arXiv preprint arXiv:1502.03044 (2015)
34. You, J., Ying, R., Ren, X., Hamilton, W., Leskovec, J.: Graphrnn: Generating realistic graphs with deep auto-regressive models. In: International Conference on Machine Learning. pp. 5694–5703 (2018)
35. You, Q., Jin, H., Wang, Z., Fang, C., Luo, J.: Image captioning with semantic attention. CoRR **abs/1603.03925** (2016), `http://arxiv.org/abs/1603.03925`
36. Zhu, Y., Fathi, A., Fei-Fei, L.: Reasoning about object affordances in a knowledge base representation. In: European conference on computer vision. pp. 408–424. Springer (2014)