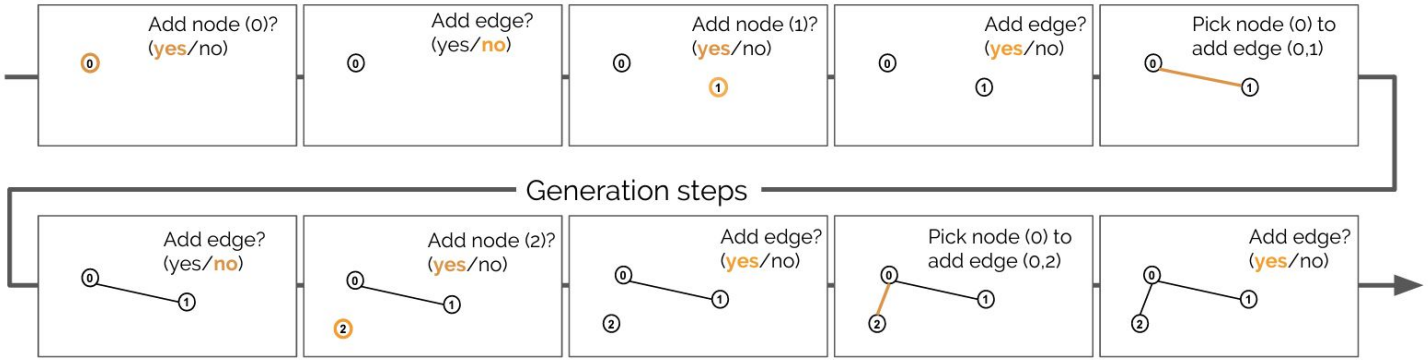


Learning Deep Generative Models of Graphs

Misunderstanding of probability may be the greatest of all impediments to scientific literacy.

-- Stephen Jay Gould

Jay Urbain, PhD - 2/5/2023

References

Learning Deep Generative Models of Graphs

https://arxiv.org/pdf/1803.03324.pdf

Seminal paper from DeepMind

Learning Deep Generative Models of Graphs

Yujia Li [1] Oriol Vinyals [1] Chris Dyer [1] Razvan Pascanu [1] Peter Battaglia [1]

Abstract

Graphs are fundamental data structures which concisely capture the relational structure in many important real-world domains, such as knowledge graphs, physical and social interactions, language, and chemistry. Here we introduce a powerful new approach for learning generative models over graphs, which can capture both their structure and attributes. Our approach uses graph neural networks to express probabilistic dependencies among a graph's nodes and edges, and can, in principle, learn distributions over any arbitrary graph. In a series of experiments our results show that once trained, our models can generate good quality samples of both synthetic graphs as well as real molecular graphs, both unconditionally and conditioned on data. Compared to baselines that do not use graph-structured representations, our models often perform far better. We also explore key challenges of learning generative models of graphs, such as how to handle symmetries and ordering of elements during the graph generation process, and offer possible solutions. Our work is the first and most general approach for learning generative models over arbitrary graphs, and opens new directions for moving away from restrictions of vector- and sequence-like knowledge representations, toward more expressive and flexible relational data structures.

1. Introduction

Graphs are natural representations of information in many problem domains. For example, relations between entities in knowledge graphs and social networks are well captured by graphs, and they are also good for modeling the physical world, e.g. molecular structure and the interactions between objects in physical systems. Thus, the ability to capture the distribution of a particular family of graphs has many applications. For instance, sampling from the graph model can lead to the discovery of new configurations that share same global properties as is, for example, required in drug discovery (Gómez-Bombarelli et al., 2016). Obtaining graph-structured semantic representations for natural language sentences (Kuhlmann & Oepen, 2016) requires the ability to model (conditional) distributions on graphs. Distributions on graphs can also provide priors for Bayesian structure learning of graphical models (Margaritis, 2003).

Probabilistic models of graphs have been studied extensively from at least two perspectives. One approach, based on random graph models, robustly assign probabilities to large classes of graphs (Erdős & Rényi, 1960; Barabási & Albert, 1999). These make strong independence assumptions and are designed to capture only certain graph properties, such as degree distribution and diameter. While these have proven effective at modeling domains such as social networks, they struggle with more richly structured domains where small structural differences can be functionally significant, such as in chemistry or representing meaning in natural language.

A more expressive—but also more brittle—approach makes use of graph grammars, which generalize mechanisms from formal language theory to model non-sequential structures (Rozenberg, 1997). Graph grammars are systems of rewrite rules that incrementally derive an output graph via a sequence of transformations of intermediate graphs.While symbolic graph grammars can be made stochastic or otherwise weighted using standard techniques (Droste & Gastin, 2007), from a learnability standpoint, two problems remain. First, inducing grammars from a set of unannotated graphs is nontrivial since reasoning about the structure building operations that might have been used to build a graph is algorithmically hard (Lautemann, 1988; Aguiñaga et al., 2016, for example). Second, as with linear output grammars, graph grammars make a hard distinction between what is in the language and what is excluded, making such models problematic for applications where it is inappropriate to assign 0 probability to certain graphs.

This paper introduces a new, expressive model of graphs that makes no structural assumptions and also avoids the brittleness of grammar-based techniques.[1] Our model gen-

[1]DeepMind, London, United Kingdom. Correspondence to: Yujia Li <yujiali@google.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[1]An analogy to language modeling before the advent of RNN

Motivation

Important natural graph application space:

- Knowledge graphs
- Social interaction nets
- Physical world
- Molecules

Capturing the distribution of a particular family of graphs is important:

- Drug discovery: discover new structures after sampling from decoder model
- Priors for bayesian structure learning
- Semantic graph representations for natural language sentences

Approaches

Old approaches:

- Based on random graph models: independence assumptions
- Grammar based

New generative approach:

- Generative model makes no structural assumptions (independence)
- Avoids brittleness of grammar-based techniques.

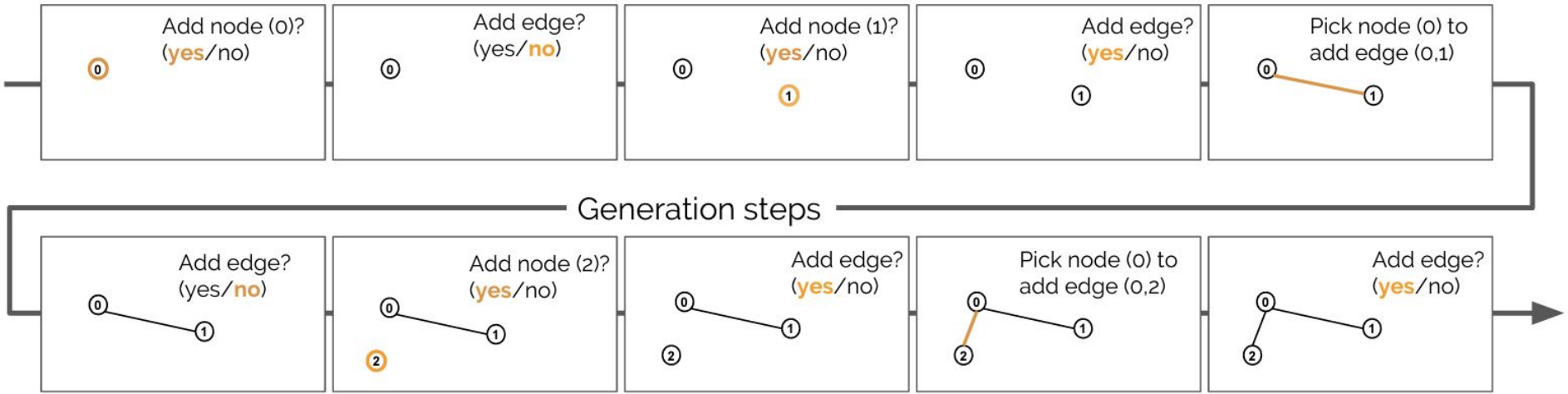
Method

Sequential Generation process

- generate one node at a time
- connect node to current partial graph by creating edges one by one

For each iteration:

1. Sample whether to add a new node of a particular type or terminate
2. If a node type is chosen, we add that type of node
3. Check if any further edges are needed to connect the new node to the existing graph
4. If yes, select a node in the graph and add an edge connecting the new node to the selected node
5. Go back to step 3
6. Repeat until the model decides not to add another edge
7. Go to step (1) to add subsequent nodes



Graph generation process

Sequence of structure building actions:

1) add a new node or not (with probabilities provided by an $f_{addnode}$ module),

2) add a new edge or not (probabilities provided by $f_{addedge}$), and

3) pick one node to connect to the new node (probabilities provided by f_{nodes}).

Learning Graph Generative Models

We can use a number of different generative models to model it treat the sequences as sentences in natural language: use LSTM (GraphRNN).

This paper used graph nets to model this sequential decision process instead

- $f_{addnode}$ (softmax), $f_{addedge}$ (sigmoid) and f_{nodes} (softmax) are GNNs

DGMG: Optimization objective

Similar to language modeling, DGMG trains the model with *behavior cloning*, or *teacher forcing*. Assume for each graph there exists a sequence of *oracle actions* $a_1, \cdots, a_T$ that generates it. What the model does is to follow these actions, compute the joint probabilities of such action sequences, and maximize them.

By chain rule, the probability of taking $a_1, \cdots, a_T$ is:

$$p(a_1, \cdots, a_T) = p(a_1)p(a_2|a_1) \cdots p(a_T|a_1, \cdots, a_{T-1})$$

The optimization objective is then simply the typical MLE loss:

$$-\log p(a_1, \cdots, a_T) = -\sum_{t=1}^{T} \log p(a_t|a_1, \cdots, a_{t-1})$$

Encoding a dynamic graph

All the actions generating a graph are sampled from probability distributions. In order to do that, you project the structured data, namely the graph, onto an Euclidean space. The challenge is that such process, called *embedding*, needs to be repeated as the graphs mutate.

Graph embedding

Let $G = (V, E)$ be an arbitrary graph. Each node v has an embedding vector $\mathbf{h}_v \in \mathbb{R}^n$. Similarly, the graph has an embedding vector $\mathbf{h}_G \in \mathbb{R}^k$. Typically, $k > n$ since a graph contains more information than an individual node.

The graph embedding is a weighted sum of node embeddings under a linear transformation:

$$\mathbf{h}_G = \sum_{v \in V} \text{Sigmoid}(g_m(\mathbf{h}_v)) f_m(\mathbf{h}_v)$$

The first term, $\text{Sigmoid}(g_m(\mathbf{h}_v))$, computes a gating function and can be thought of as how much the overall graph embedding attends on each node. The second term $f_m : \mathbb{R}^n \to \mathbb{R}^k$ maps the node embeddings to the space of graph embeddings.

Update node embeddings via graph propagation

The mechanism of updating node embeddings in DGMG is similar to that for graph convolutional networks. For a node v in the graph, its neighbor u sends a message to it with

$$\mathbf{m}_{u\to v} = \mathbf{W}_m \text{concat}([\mathbf{h}_v, \mathbf{h}_u, \mathbf{x}_{u,v}]) + \mathbf{b}_m$$

$\mathbf{x}_{u,v}$ is the embedding of the edge between u and v.

After receiving messages from all its neighbors, v summarizes them with a node activation vector

$$\mathbf{a}_v = \sum_{u:(u,v)\in E} \mathbf{m}_{u\to v}$$

and use this information to update its own feature:

$$\mathbf{h}'_v = \mathbf{GRU}(\mathbf{h}_v, \mathbf{a}_v)$$

Actions

All actions are sampled from distributions parameterized using neural networks and here they are in turn.

Action 1: Add nodes

Given the graph embedding vector $\mathbf{h}_G$, evaluate

$$\text{Sigmoid}(\mathbf{W}_{\text{add node}}\mathbf{h}_G + b_{\text{add node}})$$

.

which is then used to parameterize a Bernoulli distribution for deciding whether to add a new node.

If a new node is to be added, initialize its feature with

$$\mathbf{W}_{\text{init}}\text{concat}([\mathbf{h}_{\text{init}}, \mathbf{h}_G]) + \mathbf{b}_{\text{init}}$$

where $\mathbf{h}_{\text{init}}$ is a learnable embedding module for untyped nodes.

Action 2: Add edges

Given the graph embedding vector $\mathbf{h}_G$ and the node embedding vector $\mathbf{h}_v$ for the latest node v evaluate

$$\text{Sigmoid}(\mathbf{W}_{\text{add edge}}\text{concat}([\mathbf{h}_G, \mathbf{h}_v]) + b_{\text{add edge}})$$

which is then used to parametrize a Bernoulli distribution for deciding whether to add a new edge starting from v

Action 3: Choose a destination

When action 2 returns `True`, choose a destination for the latest node v.

$u \in \{0, \cdots, v-1\}$, the probability of choosing it is given by

$$\frac{\exp(\mathbf{W}_{\text{dest}}\text{concat}([\mathbf{h}_u, \mathbf{h}_v]) + \mathbf{b}_{\text{dest}})}{\sum_{i=0}^{v-1} \exp(\mathbf{W}_{\text{dest}}\text{concat}([\mathbf{h}_i, \mathbf{h}_v]) + \mathbf{b}_{\text{dest}})}$$

```
nx_g.nodes() []
nx_g.nodes() [0, 1]
nx_g.nodes() [0, 1, 2]
nx_g.nodes() [0, 1, 2, 3]
nx_g.nodes() [0, 1, 2, 3, 4]
nx_g.nodes() [0, 1, 2, 3, 4, 5]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
7]
nx_g.nodes() [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 1
7, 18]
```

Generate a cycle on-the-fly during inference