or we consider a small set of heuristic orderings $\pi_1, ..., \pi_n$ and average over these orderings:

$$p_\theta(\mathcal{G}|\mathbf{z}_\mathcal{G}) \approx \sum_{\pi_i \in \{\pi_1,...,\pi_n\}} \prod_{(u,v)\in\mathcal{V}} \tilde{\mathbf{A}}^{\pi_i}[u,v]\mathbf{A}[u,v]+(1-\tilde{\mathbf{A}}^{\pi_i}[u,v])(1-\mathbf{A}[u,v]).$$

These heuristic orderings do not solve the graph matching problem, but they seem to work well in practice. Liao et al. [2019a] provides a detailed discussion and comparison of these heuristic ordering approaches, as well as an interpretation of this strategy as a variational approximation.

Limitations

As with the node-level VAE approach, the basic graph-level framework has serious limitations. Most prominently, using graph-level latent representations introduces the issue of specifying node orderings, as discussed above. This issue—together with the use of MLP decoders—currently limits the application of the basic graph-level VAE to small graphs with hundreds of nodes or less. However, the graph-level VAE framework can be combined with more effective decoders—including some of the autoregressive methods we discuss in Section 9.3—which can lead to stronger models. We will mention one prominent example of such as approach in Section 9.5, when we highlight the specific task of generating molecule graph structures.

9.2 Adversarial Approaches

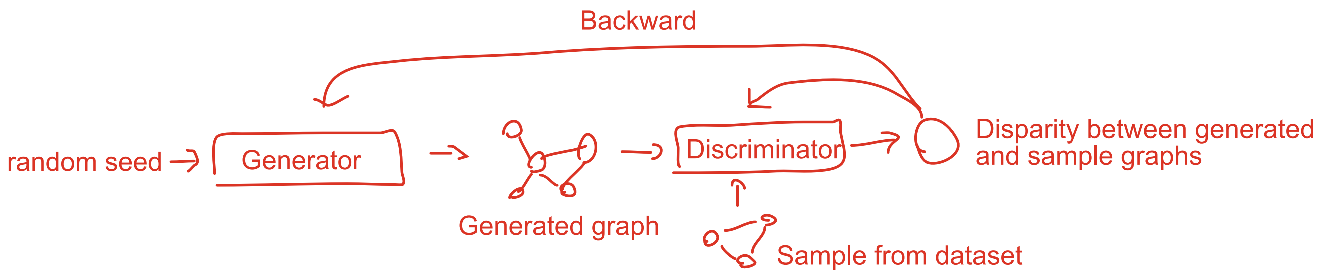
Variational autoencoders (VAEs) are a popular framework for deep generative models—not just for graphs, but for images, text, and a wide-variety of data domains. VAEs have a well-defined probabilistic motivation, and there are many works that leverage and analyze the structure of the latent spaces learned by VAE models. However, VAEs are also known to suffer from serious limitations—such as the tendency for VAEs to produce blurry outputs in the image domain. Many recent state-of-the-art generative models leverage alternative generative frameworks, with generative adversarial networks (GANs) being one of the most popular [Goodfellow et al., 2014].

idea: use another neural network to generate pseudo labels

The basic idea behind a general GAN-based generative models is as follows. First, we define a trainable generator network $g_\theta : \mathbb{R}^d \rightarrow \mathcal{X}$. This generator network is trained to generate realistic (but fake) data samples $\tilde{\mathbf{x}} \in \mathcal{X}$ by taking a random seed $\mathbf{z} \in \mathbb{R}^d$ as input (e.g., a sample from a normal distribution). At the same time, we define a discriminator network $d_\phi : \mathcal{X} \rightarrow [0, 1]$. The goal of the discriminator is to distinguish between real data samples $\mathbf{x} \in \mathcal{X}$ and samples generated by the generator $\tilde{\mathbf{x}} \in \mathcal{X}$. Here, we will assume that discriminator outputs the probability that a given input is fake.

To train a GAN, both the generator and discriminator are optimized jointly in an *adversarial game*:

$$\min_\theta \max_\phi \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log(1 - d_\phi(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{\text{seed}}(\mathbf{z})}[\log(d_\phi(g_\theta(\mathbf{z}))], \quad (9.10)$$



Major limitation:
Hard to train

Read Wasserstein GAN
(Arjovsky, 2017)
for more on why GANs
are hard to trian

where $p_{\text{data}}(\mathbf{x})$ denotes the empirical distribution of real data samples (e.g., a uniform sample over a training set) and $p_{\text{seed}}(\mathbf{z})$ denotes the random seed distribution (e.g., a standard multivariate normal distribution). Equation (9.10) represents a minimax optimization problem. The generator is attempting to minimize the discriminatory power of the discriminator, while the discriminator is attempting to maximize its ability to detect fake samples. The optimization of the GAN minimax objective—as well as more recent variations—is challenging, but there is a wealth of literature emerging on this subject [Brock et al., 2018, Heusel et al., 2017, Mescheder et al., 2018].

A basic GAN approach to graph generation

In the context of graph generation, a GAN-based approach was first employed in concurrent work by Bojchevski et al. [2018] and De Cao and Kipf [2018]. The basic approach proposed by De Cao and Kipf [2018]—which we focus on here—is similar to the graph-level VAE discussed in the previous section. For instance, for the generator, we can employ a simple multi-layer perceptron (MLP) to generate a matrix of edge probabilities given a seed vector $\mathbf{z}$:

$$\tilde{\mathbf{A}} = \sigma\left(\text{MLP}(\mathbf{z})\right), \tag{9.11}$$

Given this matrix of edge probabilities, we can then generate a discrete adjacency matrix $\hat{\mathbf{A}} \in \mathbb{Z}^{|\mathcal{V}|\times|\mathcal{V}|}$ by sampling independent Bernoulli variables for each edge, with probabilities given by the entries of $\tilde{\mathbf{A}}$; i.e., $\hat{\mathbf{A}}[u,v] \sim \text{Bernoulli}(\tilde{\mathbf{A}}[u,v])$. For the discriminator, we can employ any GNN-based graph classification model. The generator model and the discriminator model can then be trained according to Equation (9.10) using standard tools for GAN optimization.

Benefits and limitations of the GAN approach

As with the VAE approaches, the GAN framework for graph generation can be extended in various ways. More powerful generator models can be employed—for instance, leveraging the autoregressive techniques discussed in the next section—and one can even incorporate node features into the generator and discriminator models [De Cao and Kipf, 2018].

We can use a GNN as
discriminator, thus we don't
need to care about node
ordering

One important benefit of the GAN-based framework is that it removes the complication of specifying a node ordering in the loss computation. As long as the discriminator model is permutation invariant—which is the case for almost every GNN—then the GAN approach does not require any node ordering to be specified. The ordering of the adjacency matrix generated by the generator is immaterial if the discriminator is permutation invariant. However, despite this important benefit, GAN-based approaches to graph generation have so far received less attention and success than their variational counterparts. This is likely due to the difficulties involved in the minimax optimization that GAN-based approaches require, and investigating the limits of GAN-based graph generation is currently an open problem.

9.3 Autoregressive Methods

"autoregressive" refers to a type of model that predicts future values based on past values of the same sequence.

The previous two sections detailed how the ideas of variational autoencoding (VAEs) and generative adversarial networks (GANs) can be applied to graph generation. However, both the basic GAN and VAE-based approaches that we discussed used simple multi-layer perceptrons (MLPs) to generate adjacency matrices. In this section, we will introduce more sophisticated *autoregressive* methods that can decode graph structures from latent representations. The methods that we introduce in this section can be combined with the GAN and VAE frameworks that we introduced previously, but they can also be trained as standalone generative models.

By looking a graph as node and edge sequences, we can address this problem using NLP methods.

9.3.1 Modeling Edge Dependencies

The simple generative models discussed in the previous sections assumed that edges were generated *independently*. From a probabilistic perspective, we defined the likelihood of a graph given a latent representation $\mathbf{z}$ by decomposing the overall likelihood into a set of independent edge likelihoods as follows:

$$P(\mathcal{G}|\mathbf{z}) = \prod_{(u,v)\in\mathcal{V}^2} P(\mathbf{A}[u,v]|\mathbf{z}). \tag{9.12}$$

Autoregressive models are good at modeling graphs that are inherently constructed in some order.

Example: road network, SNS network. These networks are built in a step-by-step fashion (roads are built one by one, and SNS users only follow/unfollow one user at a time)

Assuming independence between edges is convenient, as it simplifies the likelihood model and allows for efficient computations. However, it is a strong and limiting assumption, since real-world graphs exhibit many complex dependencies between edges. For example, the tendency for real-world graphs to have high clustering coefficients is difficult to capture in an edge-independent model. To alleviate this issue—while still maintaining tractability—autoregressive model relax the assumption of edge independence.

Graphs that are not suitable to be modeled by autoregressive models:
molecules,
knowledge graph

Instead, in the autoregressive approach, we assume that edges are generated sequentially and that the likelihood of each edge can be conditioned on the edges that have been previously generated. To make this idea precise, we will use $\mathbf{L}$ to denote the lower-triangular portion of the adjacency matrix $\mathbf{A}$. Assuming we are working with simple graphs, $\mathbf{A}$ and $\mathbf{L}$ contain exactly the same information, but it will be convenient to work with $\mathbf{L}$ in the following equations. We will then use the notation $\mathbf{L}[v_1, :]$ to denote the row of $\mathbf{L}$ corresponding to node v_1, and we will assume that the rows of $\mathbf{L}$ are indexed by nodes $v_1, ..., v_{|\mathcal{V}|}$. Note that due to the lower-triangular nature of $\mathbf{L}$, we will have that $\mathbf{L}[v_i, v_j] = 0, \forall j > i$, meaning that we only need to be concerned with generating the first i entries for any row $\mathbf{L}[v_i, :]$; the rest can simply be padded with zeros. Given this notation, the autoregressive approach amounts to the following decomposition of the overall graph likelihood:

$$P(\mathcal{G}|\mathbf{z}) = \prod_{i=1}^{|\mathcal{V}|} P(\mathbf{L}[v_i, :]|\mathbf{L}[v_1, :], ..., \mathbf{L}[v_{i-1}, :], \mathbf{z}). \tag{9.13}$$

In other words, when we generate row $\mathbf{L}[v_i, :]$, we condition on all the previous generated rows $\mathbf{L}[v_j, :]$ with $j < i$.

9.3.2 Recurrent Models for Graph Generation

We will now discuss two concrete instantiations of the autoregressive generation idea. These two approaches build upon ideas first proposed in Li et al. [2018] and are generally indicative of the strategies that one could employ for this task. In the first model we will review—called GraphRNN [You et al., 2018]—we model autoregressive dependencies using a recurrent neural network (RNN). In the second approach—called graph recurrent attention network (GRAN) [Liao et al., 2019a]—we generate graphs by using a GNN to condition on the adjacency matrix that has been generated so far.

GraphRNN

The first model to employ this autoregressive generation approach was GraphRNN [You et al., 2018]. The basic idea in the GraphRNN approach is to use a hierarchical RNN to model the edge dependencies in Equation (9.13).

The first RNN in the hierarchical model—termed the graph-level RNN—is used to model the state of the graph that has been generated so far. Formally, the graph-level RNN maintains a hidden state $\mathbf{h}_i$, which is updated after generating each row of the adjacency matrix $\mathbf{L}[v_i, :]$:

$$\mathbf{h}_{i+1} = \text{RNN}_{\text{graph}}(\mathbf{h}_i, \mathbf{L}[v_i, L]), \tag{9.14}$$

where we use $\text{RNN}_{\text{graph}}$ to denote a generic RNN state update with $\mathbf{h}_i$ corresponding to the hidden state and $\mathbf{L}[v_i, L]$ to the observation.[2] In You et al. [2018]'s original formulation, a fixed initial hidden state $\mathbf{h}_0 = \mathbf{0}$ is used to initialize the graph-level RNN, but in principle this initial hidden state could also be learned by a graph encoder model or sampled from a latent space in a VAE-style approach.

The second RNN—termed the node-level RNN or RNN_{node}—generates the entries of $\mathbf{L}[v_i, :]$ in an autoregressive manner. RNN_{node} takes the graph-level hidden state $\mathbf{h}_i$ as an initial input and then sequentially generates the binary values of $\mathbf{L}[v_i, ;]$, assuming a conditional Bernoulli distribution for each entry. The overall GraphRNN approach is called hierarchical because the node-level RNN is initialized at each time-step with the current hidden state of the graph-level RNN.

Both the graph-level $\text{RNN}_{\text{graph}}$ and the node-level RNN_{node} can be optimized to maximize the likelihood the training graphs (Equation 9.13) using the *teaching forcing* strategy [Williams and Zipser, 1989], meaning that the ground truth values of $\mathbf{L}$ are always used to update the RNNs during training. To control the size of the generated graphs, the RNNs are also trained to output end-of-sequence tokens, which are used to specify the end of the generation process. Note that—as with the graph-level VAE approaches discussed in Section 9.1—computing the likelihood in Equation (9.13) requires that we assume a particular ordering over the generated nodes.

When generating nodes as eq. 9.13, we implicitly assume a node ordering

[2] You et al. [2018] use GRU-style RNNs but in principle LSTMs or other RNN architecture could be employed.

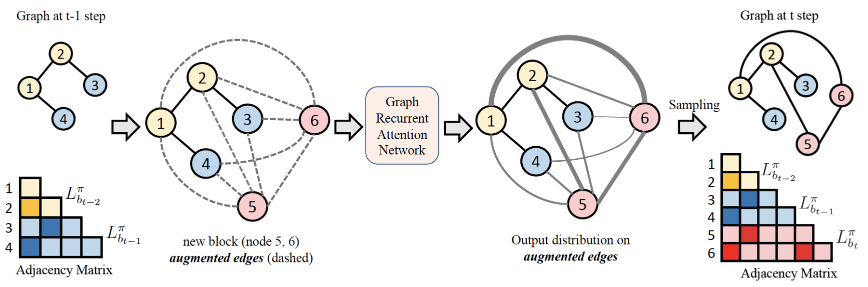


Figure 9.2: Illustration of the GRAN generation approach [Liao et al., 2019a].

After training to maximize the likelihood of the training graphs (Equation 9.13), the GraphRNN model can be used to generate graphs at test time by simply running the hierarchical RNN starting from the fixed, initial hidden state $\mathbf{h}_0$. Since the edge-level RNN involves a stochastic sampling process to generate the discrete edges, the GraphRNN model is able to generate diverse samples of graphs even when a fixed initial embedding is used. However—as mentioned above—the GraphRNN model could, in principle, be used as a decoder or generator within a VAE or GAN framework, respectively.

Graph Recurrent Attention Networks (GRAN)

The key benefit of the GraphRNN approach—discussed above—is that it models dependencies between edges. Using an autoregressive modeling assumption (Equation 9.13), GraphRNN can condition the generation of edges at generation step i based on the state of the graph that has already been generated during generation steps $1, ...i-1$. Conditioning in this way makes it much easier to generate complex motifs and regular graph structures, such as grids. For example, in Figure 9.3, we can see that GraphRNN is more capable of generating grid-like structures, compared to the basic graph-level VAE (Section 9.1). However, the GraphRNN approach still has serious limitations. As we can see in Figure 9.3, the GraphRNN model still generates unrealistic artifacts (e.g., long chains) when trained on samples of grids. Moreover, GraphRNN can be difficult to train and scale to large graphs due to the need to backpropagate through many steps of RNN recurrence.

GRNN GRAN difference:
how the intermeidate state is stored.
GRNN - a vector
GRAN - the current graph itself

To address some of the limitations of the GraphRNN approach, Liao et al. [2019a] proposed the GRAN model. GRAN—which stands for *graph recurrent attention networks*—maintains the autoregressive decomposition of the generation process. However, instead of using RNNs to model the autoregressive generation process, GRAN uses GNNs. The key idea in GRAN is that we can model the conditional distribution of each row of the adjacency matrix by running a GNN on the graph that has been generated so far (Figure 9.2):

$$P(\mathbf{L}[v_i,:]|\mathbf{L}[v_1,:],...,\mathbf{L}[v_{i-1},:],\mathbf{z}) \approx \text{GNN}(\mathbf{L}[v_1:v_{i-1},:],\tilde{\mathbf{X}}). \quad (9.15)$$

This book was wrote before diffusion methods become popular.

Diffusion methods can also be used on graph generation see https://arxiv.org/abs/2302.02591 https://arxiv.org/abs/2401.15617

Here, we use $\mathbf{L}[v_1 : v_{i-1}, :]$ to denote the lower-triangular adjacency matrix of the graph that has been generated up to generation step i. The GNN in Equation (9.15) can be instantiated in many ways, but the crucial requirement is that it generates a vector of edge probabilities $\mathbf{L}[v_i, :]$, from which we can sample discrete edge realizations during generation. For example, Liao et al. [2019a] use a variation of the graph attention network (GAT) model (see Chapter 5) to define this GNN. Finally, since there are no node attributes associated with the generated nodes, the input feature matrix $\tilde{\mathbf{X}}$ to the GNN can simply contain randomly sampled vectors (which are useful to distinguish between nodes).

The GRAN model can be trained in an analogous manner as GraphRNN by maximizing the likelihood of training graphs (Equation 9.13) using teacher forcing. Like the GraphRNN model, we must also specify an ordering over nodes to compute the likelihood on training graphs, and Liao et al. [2019a] provides a detailed discussion on this challenge. Lastly, like the GraphRNN model, we can use GRAN as a generative model after training simply by running the stochastic generation process (e.g., from a fixed initial state), but this model could also be integrated into VAE or GAN-based frameworks.

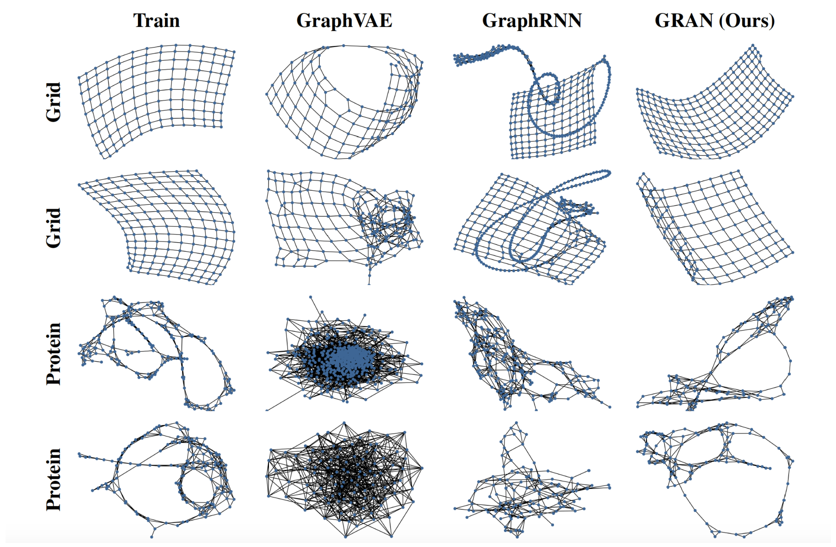
The key benefit of the GRAN model—compared to GraphRNN—is that it does not need to maintain a long and complex history in a graph-level RNN. Instead, the GRAN model explicitly conditions on the already generated graph using a GNN at each generation step. Liao et al. [2019a] also provide a detailed discussion on how the GRAN model can be optimized to facilitate the generation of large graphs with hundreds of thousands of nodes. For example, one key performance improvement is the idea that multiple nodes can be added simultaneously in a single *block*, rather than adding nodes one at a time. This idea is illustrated in Figure 9.2.

9.4 Evaluating Graph Generation

The previous three sections introduced a series of increasingly sophisticated graph generation approaches, based on VAEs, GANs, and autoregressive models. As we introduced these approaches, we hinted at the superiority of some approaches over others. We also provided some examples of generated graphs in Figure 9.3, which hint at the varying capabilities of the different approaches. However, how do we actually quantitatively compare these different models? How can we say that one graph generation approach is better than another? Evaluating generative models is a challenging task, as there is no natural notion of accuracy or error. For example, we could compare reconstruction losses or model likelihoods on held out graphs, but this is complicated by the lack of a uniform likelihood definition across different generation approaches.

Statistics-based evaluation is easy to implement
But often not enough to really tell the graph quality

In the case of general graph generation, the current practice is to analyze different statistics of the generated graphs, and to compare the distribution of statistics for the generated graphs to a test set [Liao et al., 2019a]. Formally, assume we have set of graph statistics $\mathcal{S} = (s_1, s_2, ..., s_n)$, where each of these statistics $s_{i,\mathcal{G}} : \mathbb{R} \to [0, 1]$ is assumed to define a univariate distribution over $\mathbb{R}$



We do not have a universially good criteria for graph generation.

We need:
domain-specific knowledge
or synthesisd datasets.

Unlike image and text generation, which can be easily evaluated by human, telling if a graph is good or not is hard.

To address real world problems with graph generation, we need to find a good criteria before we can do anything.

Figure 9.3: Examples of graphs generated by a basic graph-level VAE (Section 9.1), as well as the GraphRNN and GRAN models. Each row corresponds to a different dataset. The first column shows an example of a real graph from the dataset, while the other columns are randomly selected samples of graphs generated by the corresponding model [Liao et al., 2019a].

for a given graph $\mathcal{G}$. For example, for a given graph $\mathcal{G}$, we can compute the degree distribution, the distribution of clustering coefficients, and the distribution of different motifs or graphlets. Given a particular statistic s_i—computed on both a test graph $s_{i,\mathcal{G}_{\text{test}}}$ and a generated graph $s_{i,\mathcal{G}_{\text{gen}}}$—we can compute the distance between the statistic's distribution on the test graph and generated graph using a distributional measure, such as the total variation distance:

$$d(s_{i,\mathcal{G}_{\text{test}}}, s_{i,\mathcal{G}_{\text{gen}}}) = \sup_{x \in \mathbb{R}} |s_{i,\mathcal{G}_{\text{test}}}(x) - s_{i,\mathcal{G}_{\text{gen}}}(x)|. \tag{9.16}$$

To get measure of performance, we can compute the average pairwise distributional distance between a set of generated graphs and graphs in a test set.

Existing works have used this strategy with graph statistics such as degree distributions, graphlet counts, and spectral features, with distributional distances computed using variants of the total variation score and the first Wasserstein distance [Liao et al., 2019b, You et al., 2018].

9.5 Molecule Generation

All the graph generation approaches we introduced so far are useful for general graph generation. The previous sections did not assume a particular data domain, and our goal was simply to generate realistic graph structures (i.e.,

Molecule generation
The biggest challenge:
Hard to evaluate.

adjacency matrices) based on a given training set of graphs. It is worth noting, however, that many works within the general area of graph generation are focused specifically on the task of *molecule generation*.

The goal of molecule generation is to generate molecular graph structures that are both valid (e.g., chemically stable) and ideally have some desirable properties (e.g., medicinal properties or solubility). Unlike the general graph generation problem, research on molecule generation can benefit substantially from domain-specific knowledge for both model design and evaluation strategies. For example, Jin et al. [2018] propose an advanced variant of the graph-level VAE approach (Section 9.1) that leverages knowledge about known molecular motifs. Given the strong dependence on domain-specific knowledge and the unique challenges of molecule generation compared to general graphs, we will not review these approaches in detail here. Nonetheless, it is important to highlight this domain as one of the fastest growing subareas of graph generation.

Conclusion

This book provides a brief (and necessarily incomplete) tour of graph representation learning. Indeed, even as I am writing, there are new and important works arising in this area, and I expect a proper overview of graph representation learning will never be truly complete for many years to come. My hope is that these chapters provide a sufficient foundation and overview for those who are interested in becoming practitioners of these techniques or those who are seeking to explore new methodological frontiers of this area.

My intent is also for these chapters to provide a snapshot of graph representation learning as it stands in what I believe to be a pivotal moment for this nascent area. Recent years have witnessed the formalization of graph representation learning into a genuine and recognizable sub-field within the machine learning community. Spurred by the increased research attention on this topic, graph neural networks (GNNs) have now become a relatively standard technique; there are now dozens of deep generative models of graphs; and, our theoretical understanding of these techniques is solidifying at a rapid pace. However, with this solidification also comes a risk for stagnation, as certain methodologies become ingrained and the focus of research becomes increasingly narrow.

To this end, I will close this book with a brief discussion of two key areas for future work. These are not certainly not the only important areas for inquiry in this field, but they are two areas that I believe hold promise for pushing the fundamentals of graph representation learning forward.

Latent graph inference

By and large, the techniques introduced in this book assume that a graph structure is given as an input. The challenge of graph representation learning—as I have presented it—is how to embed or represent such a given input graph in an effective way. However, an equally important and complimentary challenge is the task of *inferring* graphs or relational structure from unstructured or (semi-structured) inputs. This task goes by many names, but I will call it *latent graph inference* here. Latent graph inference is a fundamental challenge for graph representation learning, primarily because it could allow us to use GNN-like methods *even when no input graph is given.* From a technical standpoint, this research direction could potentially build upon the graph generation tools introduced in Part III of this book.

Knowledge graph for LLMs
Multimodal sensing...
Reasoning
etc.

Already, there have been promising initial works in this area, such as the Neural Relational Inference (NRI) model proposed by Kipf et al. [2018] and the nearest-neighbor graphs inferred by Wang et al. [2019]. Perhaps the most exciting fact about this research direction is that preliminary findings suggest that latent graph inference might improve model performance *even when we have an input graph.* In my view, building models that can infer latent graph structures beyond the input graph that we are given is a critical direction for pushing forward graph representation learning, which could also open countless new application domains.

Breaking the bottleneck of message passing

Perhaps the single largest topic in this book—in terms of amount of space dedicated—is the neural message passing approach, first introduced in Chapter 5. This message passing formalism—where nodes aggregate messages from neighbors and then update their representations in an iterative fashion—is at the heart of current GNNs and has become the dominant paradigm in graph representation learning.

Whether we need more expressive power than WL-test

Check the specific domain you are working and see what information is needed

Sometiems a tree-structured computation graph is more than needed.

But in other cases, try adding richer features such as ring indicators, degree indicators, etc.

However, the neural message passing paradigm also has serious drawbacks. As we discussed in Chapter 7, the power of message-passing GNNs are inherently bounded by the Weisfeiler-Lehman (WL) isomorphism test. Moreover, we know that these message-passing GNNs are theoretically related to relatively simple convolutional filters, which can be formed by polynomials of the (normalized) adjacency matrix. Empirically, researchers have continually found message-passing GNNs to suffer from the problem of over-smoothing, and this issue of over-smoothing can be viewed as a consequence of the neighborhood aggregation operation, which is at the core of current GNNs. Indeed, at their core message-passing GNNs are inherently limited by the aggregate and update message-passing paradigm. This paradigm induces theoretical connections to the WL isomorphism test as well as to simple graph convolutions, but it also induces bounds on the power of these GNNs based on these theoretical constructs. At a more intuitive level, we can see that the aggregate and update message-passing structure of GNNs inherently induces a tree-structured computation (see, e.g., Figure 5.1). The embedding of each node in a GNN depends on iterative aggregations of neighborhood information, which can be represented as a tree-structured computation graph rooted at that node. Noting that GNNs are restricted to tree-structured computation graph provides yet another view of their limitations, such as their inability to consistently identify cycles and their inability to capture long-range dependencies between the nodes in a graph.

I believe that the core limitations of message-passing GNNs—i.e., being bounded by the WL test, being limited to simple convolutional filters, and being restricted to tree-structured computation graphs—are all, in fact, different facets of a common underlying cause. To push graph representation learning forward, it will be necessary to understand the deeper connections between these theoretical views, and we will need to find new architectures and paradigms that can break these theoretical bottlenecks.

Bibliography

M. Agrawal, M. Zitnik, J. Leskovec, et al. Large-scale analysis of disease pathways in the human interactome. In *PSB*, pages 111–122, 2018.

A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A.J. Smola. Distributed large-scale natural graph factorization. In *WWW*, 2013.

R. Albert and L. Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys*, 74(1):47, 2002.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

L. Babai and L. Kucera. Canonical labelling of graphs in linear average time. In *FOCS*. IEEE, 1979.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

P. Barceló, E. Kostylev, M. Monet, J. Pérez, J. Reutter, and J. Silva. The logical expressiveness of graph neural networks. In *ICLR*, 2020.

P. Battaglia et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NeurIPS*, 2002.

F. Bianchi, D. Grattarola, C. Alippi, and L. Livi. Graph neural networks with convolutional ARMA filters. *arXiv preprint arXiv:1901.01343*, 2019.

A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann. NetGan: Generating graphs via random walks. 2018.

A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, 2013.

K. Borgwardt and H. Kriegel. Shortest-path kernels on graphs. In *ICDM*, 2005.

A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2018.

J. Bruna, W. Zaremba, and Y. Szlam, A.and LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.

C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

S. Cao, W. Lu, and Q. Xu. GraRep: Learning graph representations with global structural information. In *KDD*, 2015.

J. Chen, J. Zhu, and L. Song. Stochastic training of graph convolutional networks with variance reduction. In *ICML*, 2018.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

A. Clauset, C. Shalizi, and M.E.J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, 2009.

H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016.

N. De Cao and T. Kipf. MolGAN: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2018.

C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec. Graph wavelets for structural role similarity in complex networks. In *KDD*, 2018.

J. Elman. Finding structure in time. *Cog. Sci.*, 14(2):179–211, 1990.

P. Erdös and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

H. Gao and S. Ji. Graph u-nets. In *ICML*, 2019.

J. Gilmer, S. Schoenholz, P. Riley, O. Vinyals, and G. Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT press, 2016.

L. Grafakos. *Classical and modern Fourier analysis.* Prentice Hall, 2004.

A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

A. Grover, A. Zweig, and S. Ermon. Graphite: Iterative generative modeling of graphs. In *ICML*, 2019.

W. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 2017a.

W.L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017b.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *NeurIPS*, 2017.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

P. Hoff, A.E. Raftery, and M.S. Handcock. Latent space approaches to social network analysis. *JASA*, 97(460):1090–1098, 2002.

S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bull. Am. Math. Soc*, 43(4):439–561, 2006.

W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. 2019.

Matthew O Jackson. *Social and economic networks*. Princeton university press, 2010.

G. Ji, S. He, L. Xu, K. Liu, and J. Zhao. Knowledge graph embedding via dynamic mapping matrix. In *ACL*, 2015.

W. Jin, R. Barzilay, and T. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *ICML*, 2018.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, 2003.

Y. Katznelson. *An Introduction to Harmonic Analysis*. Cambridge University Press, 2004.

D. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

T. Kipf, E. Fetaya, K. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. In *ICML*, 2018.

T.N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2016a.

T.N. Kipf and M. Welling. Variational graph auto-encoders. In *NeurIPS Workshop on Bayesian Deep Learning*, 2016b.

J. Klicpera, A. Bojchevski, and S. Günnemann. Predict then propagate: Graph neural networks meet personalized PageRank. In *ICLR*, 2019.

N. Kriege, F. Johansson, and C. Morris. A survey on graph kernels. *Appl. Netw. Sci.*, 5(1):1–42, 2020.

J.B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

E. Leicht, P. Holme, and M.E.J Newman. Vertex similarity in networks. *Phys. Rev. E*, 73(2):026120, 2006.

J. Leskovec, A Rajaraman, and J. Ullman. *Mining of Massive Data Sets*. Cambridge University Press, 2020.

R. Levie, F. Monti, X. Bresson, and M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Trans. Signal Process*, 67(1):97–109, 2018.

Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. Gated graph sequence neural networks. In *ICLR*, 2015.

Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia. Learning deep generative models of graphs. In *ICML*, 2018.

Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. In *ICML*, 2019.

R. Liao, Y. Li, Y. Song, S. Wang, W.L. Hamilton, D. Duvenaud, R. Urtasun, and R. Zemel. Efficient graph generation with graph recurrent attention networks. In *NeurIPS*, 2019a.

R. Liao, Z. Zhao, R. Urtasun, and R. Zemel. LanczosNet: Multi-scale deep graph convolutional networks. In *ICLR*, 2019b.

L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A*, 390(6):1150–1170, 2011.

D. Marcheggiani and I. Titov. Encoding sentences with graph convolutional networks for semantic role labeling. In *EMNLP*, 2017.

H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman. Provably powerful graph networks. In *NeurIPS*, 2019.

J. Mason and D. Handscomb. *Chebyshev Polynomials*. Chapman and Hall, 2002.

M. McPherson, L. Smith-Lovin, and J. Cook. Birds of a feather: Homophily in social networks. *Annu. Rev. Sociol*, 27(1):415–444, 2001.

C. Merkwirth and T. Lengauer. Automatic generation of complementary descriptors with molecular graph networks. *J. Chem. Inf. Model*, 45(5):1159–1168, 2005.

L. Mescheder, A. Geiger, and S. Nowozin. Which training methods for GANs do actually converge? In *ICML*, 2018.

C.D. Meyer. *Matrix analysis and applied linear algebra*, volume 71. SIAM, 2000.

C. Morris, M. Ritzert, M. Fey, W.L. Hamilton, J. Lenssen, G. Rattan, and M. Grohe. Weisfeiler and Leman go neural: Higher-order graph neural networks. In *AAAI*, 2019.

R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *ICLR*, 2018.

R. Murphy, B. Srinivasan, V. Rao, and B. Ribeiro. Relational pooling for graph representations. In *ICML*, 2019.

M. Newman. Mathematics of networks. *The New Palgrave Dictionary of Economics*, pages 1–8, 2016.

M. Newman. *Networks*. Oxford University Press, 2018.

D. Nguyen, K. Sirts, L. Qu, and M. Johnson. STranse: A novel embedding model of entities and relationships in knowledge bases. *arXiv preprint arXiv:1606.08140*, 2016.

M. Nickel, V. Tresp, and H. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, 2011.

M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proc. IEEE*, 104(1):11–33, 2016.

A. Oppenheim, R. Schafer, and J. Buck. *Discrete-time signal processing*. Prentice Hall, 1999.

A. Ortega, P. Frossard, J. Kovačević, J. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proc. IEEE*, 106 (5):808–828, 2018.

M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, 2016.

J. Padgett and C. Ansell. Robust action and the rise of the medici, 1400-1434. *Am. J. Sociol.*, 98(6):1259–1319, 1993.

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

S. Pandit, D. Chau, S. Wang, and C. Faloutsos. NetProbe: A fast and scalable system for fraud detection in online auction networks. In *WWW*, 2007.

B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

B. Perozzi, V. Kulkarni, and S. Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. *arXiv preprint arXiv:1605.02115*, 2016.

T. Pham, T. Tran, D. Phung, and S. Venkatesh. Column networks for collective classification. In *AAAI*, 2017.

C.R. Qi, H. Su, K. Mo, and L.J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *KDD*, 2018.

M. Qu, Y. Bengio, and J. Tang. GMNN: Graph markov neural networks. In *ICML*, 2019.

L. Rabiner and B. Gold. *Theory and application of digital signal processing.* Prentice-Hall, 1975.

L.F.R. Ribeiro, P.H.P. Saverese, and D.R. Figueiredo. struc2vec: Learning node representations from structural identity. In *KDD*, 2017.

H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Stat*, pages 400–407, 1951.

D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. Neural Netw. Learn. Syst*, 20(1): 61–80, 2009.

M. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, 2017.

D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. Dill. Learning a SAT solver from single-bit supervision. In *ICLR*, 2019.

N. Shervashidze and K. Borgwardt. Fast subtree kernels on graphs. In *NeurIPS*, 2009.

N. Shervashidze, P. Schweitzer, E. Leeuwen, K. Mehlhorn, and K. Borgwardt. Weisfeiler-lehman graph kernels. *JMLR*, 12:2539–2561, 2011.

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

K. Sinha, S. Sodhani, J. Dong, J. Pineau, and W. Hamilton. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *EMNLP*, 2019.

A. Smola, A. Gretton, L. Song, and B. Schölkopf. A Hilbert space embedding for distributions. In *COLT*, 2007.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

M. Stoer and F. Wagner. A simple min-cut algorithm. *J. ACM*, 44(4):585–591, 1997.

F. Sun, J. Hoffmann, and J. Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *ICLR*, 2020.

Z. Sun, Z. Deng, J. Nie, and J. Tang. RotatE: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2019.

J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: Large-scale information network embedding. In *WWW*, 2015.

K. Teru, E. Denis, and W.L. Hamilton. Inductive relation prediction on knowledge graphs. In *ICML*, 2020.

T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

P. Veličković, W. Fedus, W.L. Hamilton, P. Liò, Y. Bengio, and R.D. Hjelm. Deep graph infomax. In *ICLR*, 2019.

O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2015.

S.V.N. Vishwanathan, N.N. Schraudolph, R. Kondor, and K.M. Borgwardt. Graph kernels. *JMLR*, 11:1201–1242, 2010.

U. Von Luxburg. A tutorial on spectral clustering. *Stat. Comput.*, 17(4):395–416, 2007.

M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1–2):1–305, 2008.

Y. Wang, Y. Sun, Z. Liu, S. Sarma, M. Bronstein, and J. Solomon. Dynamic graph CNN for learning on point clouds. *ACM TOG*, 38(5):1–12, 2019.

Z. Wang, J Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.

Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *Nature*, 393(6684):440–442, 1998.

B. Weisfeiler and A. Lehman. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16, 1968.

R. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280, 1989.

F. Wu, T. Zhang, C. Souza, A.and Fifty, T. Yu, and K. Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

K. Xu, C. Li, Y. Tian, T. Sonobe, K. Kawarabayashi, and S. Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, 2018.

K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *ICLR*, 2019.

B. Yang, W. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*.

Z. Yang, W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. Le. XL-net: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, 2019.

R. Ying, R. He, K. Chen, P. Eksombatchai, W.L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, 2018a.

R. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, 2018b.

J. You, R. Ying, X. Ren, W.L. Hamilton, and J. Leskovec. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *ICML*, 2018.

W. Zachary. An information flow model for conflict and fission in small groups. *J. Anthropol. Res.*, 33(4):452–473, 1977.

M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola. Deep sets. In *NeurIPS*, 2017.

Y. Zhang, X. Chen, Y. Yang, A. Ramamurthy, B. Li, Y. Qi, and L. Song. Can graph neural networks help logic reasoning? In *ICLR*, 2020.

D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *NeurIPS*, 2004.

M. Zitnik, M. Agrawal, and J. Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 2018.