An algorithmic reasoning approach to GNNs

Angela Carraro and Matteo Scorcia

DSSC + IN20 - UniTS

Contents

1 Graph Representation Learning **3**

1.1 The Graph Neural Network Model 3

1.1.1 Neural Message Passing 4

1.1.2 Graph Neural Networks in Practice 5

2 Graph Representation Learning - Video **6**

3 Relational inductive biases, deep learning, and graph networks **7**

3.1 Introduction 7

3.2 Relational inductive biases 7

3.3 Relational inductive biases in standard deep learning building blocks 9

3.4 Graph networks 9

3.4.1 Definition of "graph" 10

3.4.2 Internal structure of a GN block 10

3.4.3 Computational steps within a GN block 10

Index **12**

References **13**

1 Graph Representation Learning

1.1 The Graph Neural Network Model

The first part of this book discussed approaches for learning low-dimensional embeddings of the nodes in a graph. The node embedding approaches we discussed used a shallow embedding approach to generate representations of nodes, where we simply optimized a unique embedding vector for each node. In this chapter, we turn our focus to more complex encoder models. We will introduce the graph neural network (GNN) formalism, which is a general framework for defining deep neural networks on graph data. The key idea is that we want to generate representations of nodes that actually depend on the structure of the graph, as well as any feature information we might have.

The primary challenge in developing complex encoders for graph-structured data is that our usual deep learning toolbox does not apply. For example, convolutional neural networks (CNNs) are well-defined only over grid-structured inputs (e.g., images), while recurrent neural networks (RNNs) are well-defined only over sequences (e.g., text). To define a deep neural network over general graphs, we need to define a new kind of deep learning architecture.

Permutation invariance and equivariance One reasonable idea for defining a deep neural network over graphs would be to simply use the adjacency matrix as input to a deep neural network. For example, to generate an embedding of an entire graph we could simply flatten the adjacency matrix and feed the result to a multi-layer perceptron (MLP):

$$\mathbf{z}_{\mathcal{G}} = \text{MLP}(\mathbf{A}[1] \oplus \mathbf{A}[2] \oplus \ldots \oplus \mathbf{A}[|\mathcal{V}|]); \tag{1}$$

where $\mathbf{A}[i] \in \mathbf{R}^{|\mathcal{V}|}$ denotes a row of the adjacency matrix and we use $\oplus$ to denote vector concatenation.

The issue with this approach is that it *depends on the arbitrary ordering of nodes that we used in the adjacency matrix*. In other words, such a model is not *permutation invariant*, and a key desideratum for designing neural networks over graphs is that they should permutation invariant (or equivariant). In mathematical terms, any function f that takes an adjacency matrix $\mathbf{A}$ as input should ideally satisfy one of the two following properties:

$$f(\mathbf{PAP}^T) = f(\mathbf{A}) \qquad \text{(Permutation Invariance)} \tag{2}$$

$$f(\mathbf{PAP}^T) = \mathbf{P}f(\mathbf{A}) \qquad \text{(Permutation Equivariance)}, \tag{3}$$

where P is a permutation matrix. Permutation invariance means that the function does not depend on the arbitrary ordering of the rows/columns in the adjacency matrix, while permutation equivariance means that the output of f is permuted in an consistent way when we permute the adjacency matrix. (The shallow encoders we discussed in Part I are an example of permutation equivariant functions.) Ensuring invariance or equivariance is a key challenge when

we are learning over graphs, and we will revisit issues surrounding permutation equivariance and invariance often in the ensuing chapters.

1.1.1 Neural Message Passing

The defining feature of a GNN is that it uses a form of neural message passing in which vector messages are exchanged between nodes and updated using neural networks.

During each message-passing iteration in a GNN, a *hidden embedding* $\mathbf{h}_u^{(k)}$ corresponding to each node $u \in \mathcal{V}$ is updated according to information aggregated from u's graph neighborhood $\mathcal{N}(u)$. This message-passing update can be expressed as follows:

hidden embedding

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}; \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \quad (4)$$

$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right), \quad (5)$$

where UPDATE and AGGREGATE are arbitrary differentiable functions (i.e., neural networks) and $\mathbf{m}_{\mathcal{N}(u)} = \text{AGGREGATE}(\{\mathbf{h}_v, \forall v \in \mathcal{N}(u)\})$ is the "message" that is aggregated from u's graph neighborhood $\mathcal{N}(u)$. We use superscripts to distinguish the embeddings and functions at different iterations of message passing. The different iterations of message passing are also sometimes known as the different "layers" of the GNN.

At each iteration k of the GNN, the AGGREGATE function takes as input the set of embeddings of the nodes in u's graph neighborhood $\mathcal{N}(u)$ and generates a message $\mathbf{m}_{\mathcal{N}(u)}^{(k)} = \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})$ based on this aggregated neighborhood information. The update function UPDATE then combines the message $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ with the previous embedding $\mathbf{h}_u^{(k1)}$ of node u to generate the updated embedding $\mathbf{h}_u^{(k)}$. The initial embeddings at k = 0 are set to the input features for all the nodes, i.e., $\mathbf{h}_u^{(0)} = \mathbf{x}_u, \forall u \in \mathcal{V}$. After running K iterations of the GNN message passing, we can use the output of the final layer to define the embeddings for each node, i.e.,

$$\mathbf{z}_u = \mathbf{h}_u^{(K)}, \forall u \in \mathcal{V}. \quad (6)$$

Note that since the AGGREGATE function takes a set as input, GNNs defined in this way are permutation equivariant by design.

In cases where no node features are available, there are still several options. One option is to use node statistics to define features. Another popular approach is to use identity features, where we associate each node with a one-hot indicator feature, which uniquely identifies that node. Note, however, that the using identity features makes the model transductive and incapable of generalizing to unseen nodes.

The basic intuition behind the GNN message-passing framework is straightforward: at each iteration, every node aggregates information from its local

neighborhood, and as these iterations progress each node embedding contains more and more information from further reaches of the graph.

But what kind of "information" do these node embeddings actually encode? Generally, this information comes in two forms. On the one hand there is *structural information* about the graph. For example, after k iterations of GNN message passing, the embedding $\mathbf{h}_u^{(k)}$ of node u might encode information about the degrees of all the nodes in u's k-hop neighborhood. In addition to structural information, the other key kind of information captured by GNN node embedding is *feature-based*. After k iterations of GNN message passing, the embeddings for each node also encode information about all the features in their k-hop neighborhood. This local feature-aggregation behaviour of GNNs is analogous to the behavior of the convolutional kernels in convolutional neural networks (CNNs). However, whereas CNNs aggregate feature information from spatially-defined patches in an image, GNNs aggregate information based on local graph neighborhoods.

structural information

feature-based

1.1.2 Graph Neural Networks in Practice

In Chapter 5, we introduced a number of graph neural network (GNN) architectures. However, we did not discuss how these architectures are optimized and what kinds of loss functions and regularization are generally used. In this chapter, we will turn our attention to some of these practical aspects of GNNs. We will discuss some representative applications and how GNNs are generally optimized in practice, including a discussion of unsupervised pre-training methods that can be particularly effective. We will also introduce common techniques used to regularize and improve the efficiency of GNNs.

In the vast majority of current applications, GNNs are used for one of three tasks: node classification, graph classification, or relation prediction. As discussed in Chapter 1, these tasks reflect a large number of real-world applications, such as predicting whether a user is a bot in a social network (node classification), property prediction based on molecular graph structures (graph classification), and content recommendation in online platforms (relation prediction). In this section, we briefly describe how these tasks translate into concrete loss functions for GNNs, and we also discuss how GNNs can be pre-trained in an unsupervised manner to improve performance on these downstream tasks.

2 Graph Representation Learning - Video

Graphs are a general and universal language for describing and modelling complex systems/data.

In a graph setting, we are not considering a set of independent points, but really the whole object that we're trying to actually do learning upon is bound up in the interconnections or the relationships between these points. So rather than a set of individual data points we're considering the relationships between them.

Even trying to tell whether or not two graphs are the same is NP-indeterminate (it's believed not to be solved in polynomial time), since there is no standard or canonical way to order the nodes in the adjacency matrices (in a graph there is no up and down like in an image!).

3 Relational inductive biases, deep learning, and graph networks

3.1 Introduction

A key signature of human intelligence is the ability to make "infinite use of finite means", in which a small set of elements (such as words) can be productively composed in limitless ways (such as into new sentences). This reflects the principle of *combinatorial generalization*, that is, constructing new inferences, predictions, and behaviors from known building blocks. Here we explore how to improve modern AI's capacity for combinatorial generalization by biasing learning towards structured representations and computations, and in particular, systems that operate on graphs. The question of how to build artificial systems which exhibit combinatorial generalization has been at the heart of AI since its origins, and was central to many structured approaches.

combinatorial generalization

Recently, a class of models has arisen at the intersection of deep learning and structured approaches, which focuses on approaches for reasoning about explicitly structured data, in particular graphs. What these approaches all have in common is a capacity for performing computation over discrete entities and the relations between them. What sets them apart from classical approaches is how the representations and structure of the entities and relations — and the corresponding computations — can be learned, relieving the burden of needing to specify them in advance. Crucially, these methods carry strong *relational inductive biases*, in the form of specific architectural assumptions, which guide these approaches towards learning about entities and relations, which we, joining many others, suggest are an essential ingredient for human-like intelligence.

3.2 Relational inductive biases

We define *structure* as the product of composing a set of known building blocks. "Structured representations" capture this composition (i.e., the arrangement of the elements) and "structured computations" operate over the elements and their composition as a whole. *Relational reasoning*, then, involves manipulating structured representations of *entities* and *relations*, using *rules* for how they can be composed. We use these terms to capture notions from cognitive science, theoretical computer science, and AI, as follows:

relational reasoning

- An *entity* is an element with attributes, such as a physical object with a size and mass.

entity

- A *relation* is a property between entities. Relations between two objects might include same size as, heavier than, and distance from. Relations can have attributes as well. The relation more than X times heavier than takes an attribute, X, which determines the relative weight threshold for the relation to be true vs. false. Relations can also be sensitive to the

relation

global context. For a stone and a feather, the relation falls with greater acceleration than depends on whether the context is in air vs. in a vacuum. Here we focus on pairwise relations between entities.

- A *rule* is a function (like a non-binary logical predicate) that maps entities and relations to other entities and relations, such as a scale comparison like is entity X large? and is entity X heavier than entity Y?. Here we consider rules which take one or two arguments (unary and binary), and return a unary property value.

rule

An *inductive bias* allows a learning algorithm to prioritize one solution (or interpretation) over another, independent of the observed data. In a Bayesian model, inductive biases are typically expressed through the choice and parameterization of the prior distribution. In other contexts, an inductive bias might be a regularization term added to avoid overfitting, or it might be encoded in the architecture of the algorithm itself. Inductive biases often trade flexibility for improved sample complexity and can be understood in terms of the bias-variance tradeoff. Ideally, inductive biases both improve the search for solutions without substantially diminishing performance, as well as help find solutions which generalize in a desirable way; however, mismatched inductive biases can also lead to suboptimal performance by introducing constraints that are too strong.

inductive bias

Many approaches in machine learning and AI which have a capacity for relational reasoning use a *relational inductive bias*. While not a precise, formal definition, we use this term to refer generally to inductive biases which impose constraints on relationships and interactions among entities in a learning process.

relational inductive bias

To explore the relational inductive biases expressed within various deep learning methods, we must identify several key ingredients: what are the *entities*, what are the *relations*, and what are the *rules* for composing entities and relations, and computing their implications? In deep learning, the entities and relations are typically expressed as distributed representations, and the rules as neural network function approximators; however, the precise forms of the entities, relations, and rules vary between architectures. To understand these differences between architectures, we can further ask how each supports relational reasoning by probing:

- The arguments to the rule functions (e.g., which entities and relations are provided as input).
- How the rule function is reused, or shared, across the computational graph (e.g., across different entities and relations, across different time or processing steps, etc.).
- How the architecture defines interactions versus isolation among representations (e.g., by applying rules to draw conclusions about related entities, versus processing them separately).

3.3 Relational inductive biases in standard deep learning building blocks

...

3.4 Graph networks

graph neural networks

Models in the *graph neural network* family have been explored in a diverse range of problem domains, across supervised, semi-supervised, unsupervised, and reinforcement learning settings. They have been effective at tasks thought to have rich relational structure, such as

- visual scene understanding tasks and few-shot learning
- learn the dynamics of physical systems and multi-agent systems
- reason about knowledge graphs
- predict the chemical properties of molecules
- predict traffic on roads
- classify and segment images and videos and 3D meshes and point clouds
- classify regions in images
- perform semi-supervised text classification
- machine translation
- combinatorial optimization
- boolean satisfiability

The works cited above are by no means an exhaustive list, but provide a representative cross-section of the breadth of domains for which graph neural networks have proven useful.

graph networks (GN)

We now present our *graph networks (GN)* framework, which defines a class of functions for relational reasoning over graph-structured representations. Note, we avoided using the term "neural" in the "graph network" label to reflect that they can be implemented with functions other than neural networks, though here our focus is on neural network implementations.

The main unit of computation in the GN framework is the GN block, a "graph-to-graph" module which takes a graph as input, performs computations over the structure, and returns a graph as output. As described in Box 3, entities are represented by the graph's nodes, relations by the edges, and system-level properties by global attributes.

3.4.1 Definition of "graph"

Here we use "graph" to mean a directed, attributed multi-graph with a global attribute. In our terminology, a *node* is denoted as $\mathbf{v}_i$, an *edge* as $\mathbf{e}_k$, and the *global attributes* as $\mathbf{u}$. We also use s_k and r_k to indicate the indices of the sender and receiver nodes (see below), respectively, for edge k. To be more precise, we define these terms as:

node, edge, global attribute

Directed:	one-way edges, from a "sender" node to a "receiver" node.
Attribute:	properties that can be encoded as a vector, set, or even another graph.
Attributed:	edges and vertices have attributes associated with them.
Global attribute:	a graph-level attribute.
Multi-graph:	there can be more than one edge between vertices, including self-edges.

Within our GN framework, a *graph* is defined as a 3-tuple $G = (\mathbf{u}; V; E)$. The $\mathbf{u}$ is a *global attribute*; the $V = \{\mathbf{v}_i\}_{i=1:N^v}$ is the *set of nodes* (of cardinality N^v), where each $\mathbf{v}_i$ is a node's attribute. The $E = \{(\mathbf{e}_k; r_k; s_k)\}_{k=1:N^e}$ is the *set of edges* (of cardinality N^e), where each $\mathbf{e}_k$ is the edge's attribute, r_k is the index of the receiver node, and s_k is the index of the sender node.

graph

3.4.2 Internal structure of a GN block

A GN block contains three "update" functions, ϕ, and three "aggregation" functions, ρ,

$$\begin{aligned} \mathbf{e}'_k &= \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right) & \bar{\mathbf{e}}'_i &= \rho^{e\to v}\left(E'_i\right) \\ \mathbf{v}'_i &= \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right) & \bar{\mathbf{e}}' &= \rho^{e\to u}\left(E'\right) \\ \mathbf{u}' &= \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right) & \bar{\mathbf{v}}' &= \rho^{v\to u}\left(V'\right) \end{aligned} \tag{7}$$

where $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$, $V' = \{\mathbf{v}'_i\}_{i=1:N^v}$, and $E' = \bigcup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$.

The ϕ^e is mapped across all edges to compute per-edge updates, the ϕ^v is mapped across all nodes to compute per-node updates, and the ϕ^u is applied once as the global update. The ρ functions each take a set as input, and reduce it to a single element which represents the aggregated information. Crucially, the ρ functions must be invariant to permutations of their inputs, and should take variable numbers of arguments (e.g., elementwise summation, mean, maximum, etc.).

3.4.3 Computational steps within a GN block

When a graph, G, is provided as input to a GN block, the computations proceed from the edge, to the node, to the global level. Algorithm 1 shows the following steps of computation:

Algorithm 1 Steps of computation in a full GN block.

function GRAPHNETWORK(E, V, $\mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right)$ ▷ 1. Compute updated edge attributes
  end for
  for $i \in \{1 \dots N^n\}$ **do**
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}\left(E'_i\right)$ ▷ 2. Aggregate edge attributes per node
    $\mathbf{v}'_i \leftarrow \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right)$ ▷ 3. Compute updated node attributes
  end for
  let $V' = \{\mathbf{v}'\}_{i=1:N^v}$
  let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}\left(E'\right)$ ▷ 4. Aggregate edge attributes globally
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}\left(V'\right)$ ▷ 5. Aggregate node attributes globally
  $\mathbf{u}' \leftarrow \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right)$ ▷ 6. Compute updated global attribute
  return $(E', V', \mathbf{u}')$
end function

1. ϕ^e is applied per edge, with arguments $(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$, and returns $\mathbf{e}'_k$. The set of resulting per-edge outputs for each node, i, is, $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\ k=1:N^e}$. And $E' = \bigcup_i E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$ is the set of all per-edge outputs.
2. $\rho^{e\to v}$ is applied to E'_i, and aggregates the edge updates for edges that project to vertex i, into $\bar{\mathbf{e}}'_i$, which will be used in the next step's node update.
3. ϕ^v is applied to each node i, to compute an updated node attribute, $\mathbf{v}'_i$. The set of resulting per-node outputs is, $V' = \{\mathbf{v}'_i\}_{i=1:N^v}$.
4. $\rho^{e\to u}$ is applied to E', and aggregates all edge updates, into $\bar{\mathbf{e}}'$, which will then be used in the next step's global update.
5. $\rho^{v\to u}$ is applied to V', and aggregates all node updates, into $\bar{\mathbf{v}}'$, which will then be used in the next step's global update.
6. ϕ^u is applied once per graph, and computes an update for the global attribute, $\mathbf{u}'$.

Index

combinatorial generalization, 7

edge, 10
entity, 7

feature-based, 5

global attribute, 10
graph, 10
graph networks (GN), 9
graph neural network, 9

hidden embedding, 4

inductive bias, 8

node, 10

relation, 7
relational inductive bias, 7, 8
relational reasoning, 7
rule, 8

set of edges, 10
set of nodes, 10
structural information, 5
structure, 7

References

[1] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H. F., Ballard, A. J., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K. R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks. *CoRR abs/1806.01261* (2018).

[2] Hamilton, W. L. Graph representation learning.

[3] Hamilton, W. L. *Graph Representation Learning*, vol. 14. Morgan and Claypool, 2020.

[4] Hamilton, W. L., Ying, R., and Leskovec, J. Representation learning on graphs: Methods and applications. *CoRR abs/1709.05584* (2017).

[5] Veličković, P. Graph representation learning for algorithmic reasoning.

[6] Veličković, P. Theoretical foundations of graph neural networks.

[7] Yan, Y., Swersky, K., Koutra, D., Ranganathan, P., and Hashemi, M. Neural execution engines: Learning to execute subroutines. *CoRR abs/2006.08084* (2020).