Relational inductive biases, deep learning, and graph networks[*]

Peter Battaglia et al.

2018

1 What

- The authors explore how we can combine relational inductive biases and DL.
- They introduce *graph network* – a building block which generalises the current approaches of neural networks operating on graphs.
- They discuss the future direction of the work on marrying DL and relational approach.

2 Why

We've seen a lot of progress in the research, but there is still a huge gap between humans and our best models. The authors claim that this is due to the lack of *combinatorial generalisation*, i.e. constructing new inferences, predictions, and behaviours from known building blocks.

Humans are good at representing structure and reasoning about relations. We are also good at finding hierarchies and thinking about things as something composite of simpler blocks.

Some time ago people thought the relational approach was the Thing. Recently we have seen a flood of DL research from the folks who do not care about this old Thing. There is a lot of critiques recently which shows that smart terminators are still too far away.

What if we take both of these two worlds and get an amazing generalisation power of the relational algebra along with very powerful function approximators and all the recent advances in machine learning research.

[*]Notes by Vitaly Kurin `https://yobibyte.github.io/`. Thanks to most of the authors for the paper sources (I copypaste long formulas and figures sometimes). And thanks to you all for feedback. Stay tuned!

3 How

3.1 Relational inductive biases

Let's start with a bit of terminology.

- *Structured X* means that X is composed of some known building blocks.
- An *entity* is an element with attributes (e.g. size)
- A *relation* is a property between entities (e.g. 'bigger than')
- A *rule* is a function that maps entities and relations to other entities and relations.

Relational inductive bias (RIB) is not strictly defined, but implies imposing additional constraints on relations and interactions among entities during learning.

Inductive biases, though not relational, are already out there: network architecture, dropout, regularisation etc. The relational inductive biases are also kinda out there:

- For MLPs RIBs are quite weak, with units being entities and relations being 'all-to-all'.
- Grid elements are entities in CNNs, the relations are local, the network cares about locality and tries to be spatial translation invariant.
- For RNNs the timesteps are the entities, the relations are sequential and we care about time translation invariance.

One cool thing about graph networks is that the relations and entities are much more general. There are fewer assumptions here, actually, edges might mean anything. Graph networks are node/edge permutations invariant.

The profit from RIB here comes not from the presence of something, but rather from the absence. Why having connections everywhere as in MLP when there is only a couple of connections that really matter?

3.2 Graph networks

Graph neural networks have been out there for a while, and authors give a lot of references to reinforce this point. However, they want to unify existing approaches and present the *graph network* block.

Here, by a graph, we assume a directed, attributed multi-graph with a global attribute. A graph is a tuple $G = (\mathbf{u}, V, E)$, where $V = \{\mathbf{v}_i\}_{i=1:N^v}$ is a set of nodes, $E = \{\mathbf{e}_k, r_k, s_k\}_{k=1:N^e}$ is the set of edges, and u is a global attribute, $\mathbf{e}_k$ is edge attributes, $\mathbf{v}_i$ is node attributes. The graph is directed, so, the edge direction matters with s_k being the sender node, and r_k being the receiever node.

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

$$\begin{aligned} \mathbf{e}'_k &= \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right) & \bar{\mathbf{e}}'_i &= \rho^{e\to v}\left(E'_i\right) \\ \mathbf{v}'_i &= \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right) & \bar{\mathbf{e}}' &= \rho^{e\to u}\left(E'\right) \\ \mathbf{u}' &= \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right) & \bar{\mathbf{v}}' &= \rho^{v\to u}\left(V'\right) \end{aligned} \quad (1)$$

where $E'_i = \left\{(\mathbf{e}'_k, r_k, s_k)\right\}_{r_k=i,\ k=1:N^e}$, $V' = \left\{\mathbf{v}'_i\right\}_{i=1:N^v}$, and $E' = \bigcup_i E'_i = \left\{(\mathbf{e}'_k, r_k, s_k)\right\}_{k=1:N^e}$. Aggregation functions must be input permutations invariant and take any amount of input points.

What does it all mean? Everybody cares about the global graph attribute, even the global graph attribute. Apart from that, every edge cares about its attributes and attributes of the nodes it connects. A node, apart from the global attribute, cares about the incoming edges and its attributes.

The update process works as follows:

- We update the edge attributes for each of the edges.
- Then for each of the nodes we aggregate incoming edge attributes and update node attributes.
- Then we aggregate all the edges attributes, do the same with nodes and update the global attribute.

The authors point out that it's not necessarily to do the updates in the order above. It can be, for instance, reversed.

So, why is all of this might help? First of all, an edge is a very general thing. It just tells that the two nodes are somehow related, there are no restrictions whatsoever. Second, entities are invariant to permutations and we do not need to learn all combinations possible. Finally, per-edge/per-node functions are reused across all edges/nodes. The authors claim that this makes them automatically support some form of combinatorial generalisation. It is not clear to me, actually.

3.3 Design principles for graph network architectures

Graph networks are flexible in terms of attributes representations we might want to choose: tensors, sets, graphs etc. They will most likely depend on your problem. Regarding the output, we can use edges, nodes or the global attributes based on what we want to achieve.

Having all these possibilities means that we have to spend more time on our model architectures when using GNs. Not like now when if it's an image, use CNNs, when it's a sequence, use LSTM [1]. At the same time, it looks like the architecture choice is more principled in this case since there is not 'just two more layers' or 'add residual connection and use relu6 here'.

[1]Or two, that's always better.

Regarding the question of how the input data is to be represented as a graph, there are two possible options: the input contains the relational structure or the relations must be inferred.

Apart from all above, we have to decide on the inner-block architecture: what are edge/node/globals functions? The authors show how Graph Networks unify some of the existing approaches, e.g. Message-Parsing neural network, Non-Local neural network and some others. Finally, the authors show different ways of combining graph network blocks: composition, encode-process-decode, recurrent GN.

4 Discussion

In the final section, the authors show the empirical evidence of combinatorial generalisation in different variants of models operating on graphs. They do not forget about the limitations, e.g. there is no guarantee that GNs can solve some classes of problems, such as discriminating between certain non-isomorphic graphs.

Some important tools as recursion, control flow and conditional iteration are not easy to represent on graphs. The authors point out that programs and other 'computer-like' approaches might be more powerful for the notions above.

In the end, the paper shows some of the open questions in the domain and some directions for tackling these issues:

- Where is the graph, Lebowski? Who provides us with it? DL is cool because it can work with raw inputs. Who does the raw $\rightarrow$ graph conversion here?
- How to adaptively change the graph during computation?
- How can interpretability profit from GNs?

5 Comments

- Disclaimer: before I've read this paper I had a very vague idea of what this kind of models is. If I say, something is super novel but it's not, it is mostly due to my ignorance, but not the authors' evil intent.
- Having said above, however, I do not like 'graph network' term. I think I got an idea, they try to unify current approaches and make something default. But naming them 'graph networks' feels like 'nothing similar was there before'. Maybe that's just me. They also say they removed 'neural' from the term to show that they can be implemented via functions different from NNs. Why did they leave 'network' then? Or is is because a graph is a network?

- I don't really like the term *combinatorial generalisation*. It looks like something really strictly defined, but in reality, it is not. Or I just missed something about it. It looks like by *combinatorial generalisation* the authors mean something like 'disentangled representations' on the reasoning level (also a vague thing x_X). But it's not completely clear.

- The paper has a huge reference section and that makes my inner Juergen happy.

- The paper has a 'Limitations' section, no joke! That's so cool! And, actually, 'Open Questions' section can be considered as 'Current Limitations' as well.

- I didn't get the point about input representation when we have no idea about the relations and might do all-to-all connections. Can't we then achieve this with MLP and lose all the fancy features of graph networks?

- There is no code right now, but it should appear soon [2].

- Interesting, that, as the authors put it, one takeaway from the paper is less about graphs and more about blending powerful DL methods with structured representations.

[2]`https://twitter.com/OriolVinyalsML/status/1006655938797932547`