

# Relational inductive biases, deep learning, and graph networks

---

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, Razvan Pascanu.

November 15, 2018

# Table of contents

1. Relational inductive biases
2. Graph networks
3. Conclusions

## Relational inductive biases

---

# Relational reasoning

We define *structure* as the product of composing a set of known building blocks. Relational reasoning, then, involves manipulating structured representations of *entities* and *relations*, using *rules* for how they can be composed.

- An *entity* is an element with attributes, such as a physical object with a size and mass.
- A *relation* is a property between entities.
  - Relations can have attributes.
  - Relations can also be sensitive to the global context.
- A *rule* is a function that maps entities and relations to other entities and relations.

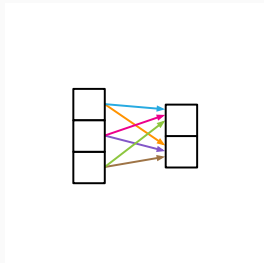
# Inductive biases

Learning involves searching a space of solutions for one expected to provide a better explanation of the data or to achieve higher rewards. But in many cases, there are multiple solutions which are equally good. An *inductive bias* allows a learning algorithm to prioritize one solution (or interpretation) over another, independent of the observed data.

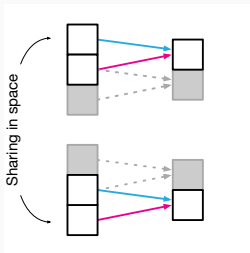
Ideally, inductive biases both improve the search for solutions without substantially diminishing performance.

# Computations over sets and graphs

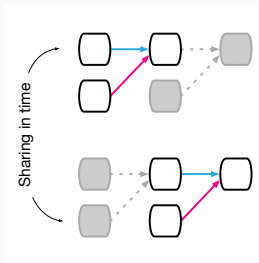
While the standard deep learning toolkit contains methods with various forms of relational inductive biases



(a) Fully connected



(b) Convolutional

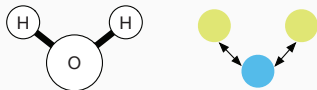


(c) Recurrent

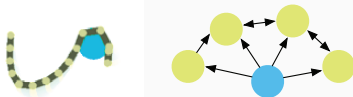
there is no “default” deep learning component which operates on arbitrary relational structure. Moreover, entities in the world do not have a natural order; rather, orderings can be defined by the properties of their relations.

# Different graph representations

(a) Molecule



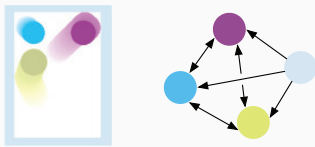
(b) Mass-Spring System



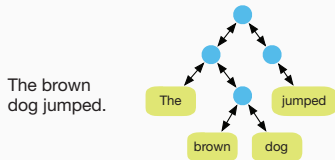
(c)  $n$ -body System



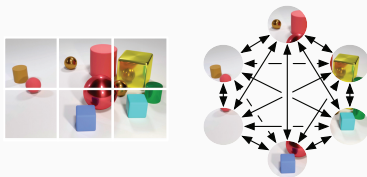
(d) Rigid Body System



(e) Sentence and Parse Tree



(f) Image and Fully-Connected Scene Graph



Sets are a natural representation for systems which are described by entities whose order is undefined or irrelevant. Consider the task of predicting the center of mass of a solar system comprised of  $n$  planets, whose attributes (e.g., mass, position, velocity, etc.) are denoted by  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ . Here the order in which we consider the planets does not matter. However, if we use a MLP for this task, having learned the prediction for a particular input  $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$  would not necessarily transfer to making a prediction for the same inputs under a different ordering  $(\mathbf{x}_n, \mathbf{x}_1, \dots, \mathbf{x}_2)$ . Since there are  $n!$  such possible permutations, in the worst case, the MLP could consider each ordering as fundamentally different.



# Graph networks

---

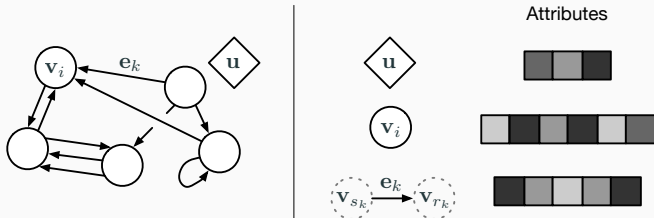
# Graph networks

Graphs are representations that supports arbitrary relational structure, and afford a strong relational inductive bias beyond that which convolutional and recurrent layers can provide.

Neural networks that operate on graphs, named “graph neural networks”, have been used in:

- Visual scene understanding.
- Multi-agent systems.
- Dynamics of physical systems.
- Reason about knowledge graphs.
- To predict traffic on roads.
- To classify and segment images and videos.
- To perform semi-supervised text classification.
- Machine translation

# Definition of Graph



The main unit of computation in the *graph networks* 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. Here we use “graph” to mean a directed, attributed multi-graph with a global attribute.

# Definition of graph

To be more precise, we define these terms as:

**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.

# Formal definition

Within a 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.

# Internal structure of a GN block

A GN block contains three “update” functions,  $\phi$ , and three “aggregation” functions,  $\rho$ ,

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

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

# Internal structure of a GN block

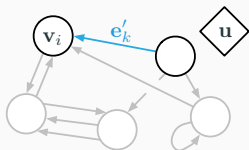
**Algorithm 1** Steps of computation in a full GN block.

---

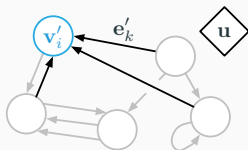
```
function GRAPHNETWORK( $E, V, \mathbf{u}$ )  
  for  $k \in \{1 \dots N^e\}$  do  
     $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$  ▷ 1. Compute updated edge attributes  
  end for  
  for  $i \in \{1 \dots N^n\}$  do  
    let  $E'_i = \{(\mathbf{e}'_k,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

---

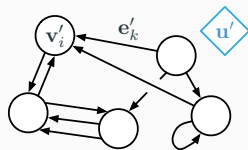
# Internal structure of a GN block



(a) Edge update



(b) Node update



(c) Global update



# Graph network architecture

The  $\phi$ 's can be implemented using neural networks (denoted  $\text{NN}_e$ ,  $\text{NN}_v$ , and  $\text{NN}_u$  below). Their  $\rho$  can be implemented using elementwise summation, but averages and max/min could also be used ( $[\mathbf{x}, \mathbf{y}, \mathbf{z}]$  indicates vector/tensor concatenation).

$$\phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) := f^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) = \text{NN}_e([\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}])$$

$$\phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) := f^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) = \text{NN}_v([\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}])$$

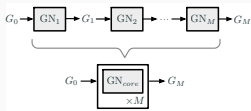
$$\phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) := f^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) = \text{NN}_u([\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}])$$

$$\rho^{e \rightarrow v}(E'_i) := \sum_{\{k: r_k=i\}} \mathbf{e}'_k$$

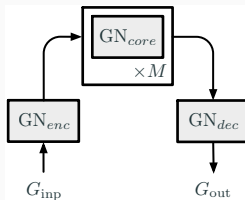
$$\rho^{v \rightarrow u}(V') := \sum_i \mathbf{v}'_i$$

$$\rho^{e \rightarrow u}(E') := \sum_k \mathbf{e}'_k$$

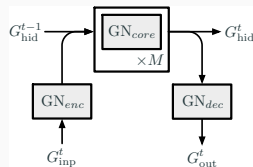
# Composable multi-block architectures



(a) Composition of GN blocks



(b) Encode-process-decode



(c) Recurrent GN architecture

# Conclusions

---

# Summary

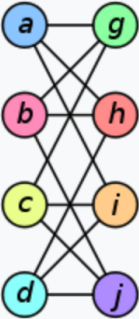
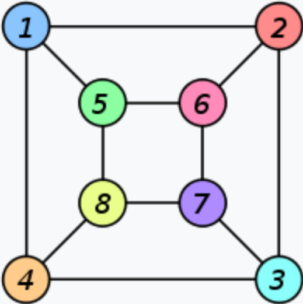
Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

# Relational inductive biases in graph networks

The GN framework imposes several strong relational inductive biases when used as components in a learning process:

- Graphs can express arbitrary relationships among entities, which means the GN's input determines how representations interact and are isolated, rather than those choices being determined by the fixed architecture.
- Graphs represent entities and their relations as sets, which are invariant to permutations.
- a GN's per-edge and per-node functions are reused across all edges and nodes, respectively. This means GNs automatically support a form of combinatorial generalization.

# Problem: Graph isomorphism problem

Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Thank you!