Deep Neural Networks (Głębokie Sieci Neuronowe)

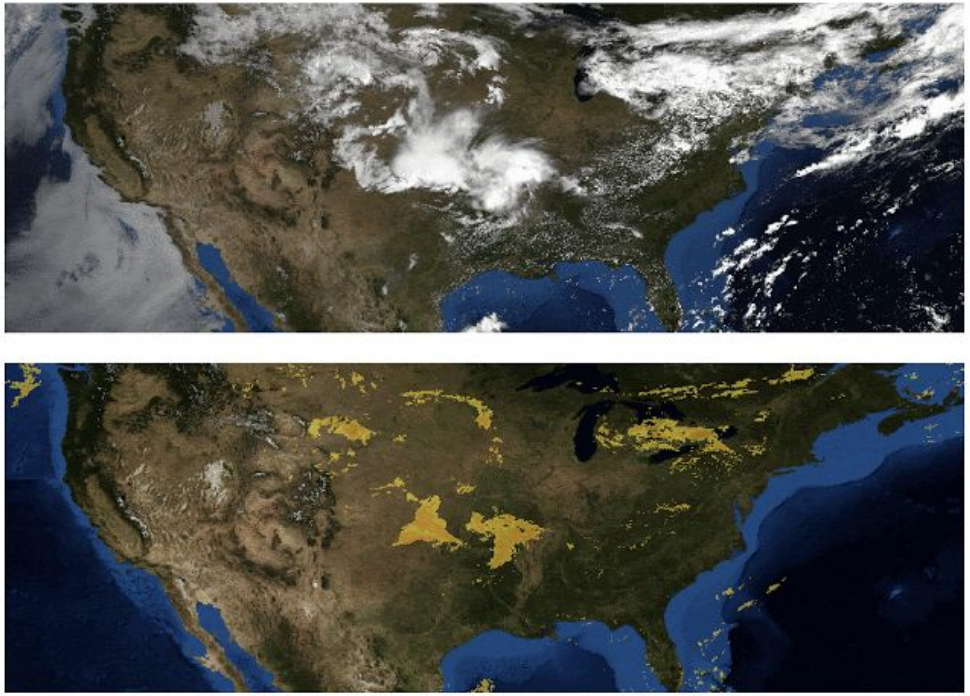
Module 5: Graph Neural Networks

Krzysztof Krawiec

Wydział Informatyki i Telekomunikacji
Politechnika Poznańska
2019/2020
http://www.cs.put.poznan.pl/kkrawiec/

First: some news: the power of good old UNet

https://ai.googleblog.com/2020/01/using-machine-learning-to-nowcast.html?m=1



Outline

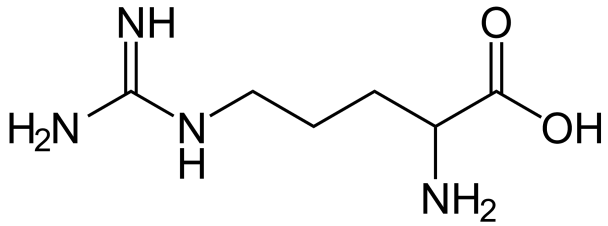
1. Introduction
2. Message-passing GNNs: A use case
3. Covariant graph networks
4. Attempts of unification

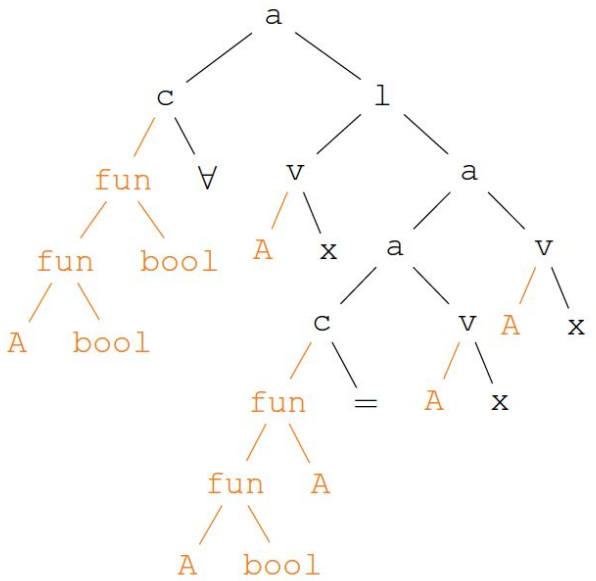
Recall: Sequence processing will be intensely discussed in the Linguistic Engineering course.

Introduction

Motivations for processing graphs

- A range of application areas involve non-tensor data: computer vision, chemistry, network science, programs, …
- Graphs are natural for modeling many natural phenomena: relationships, flows, dependencies, ...





Challenges in processing graphs with NNs

The primary challenge: variable-size structure.
Compared to sequence learning:

- Sequences are 1D structures; trees and graphs aren't.
- More arbitrarity in traversing the structure:
  - Which node to start with? Which to select next? Or should we parse all nodes in parallel?
- No 'natural ordering' of constituents of the structure.
  - Note that for sequences such ordering is explicit.

Therefore:

- Most approaches assume that graph structure does not change during processing.
- Rather than that, they process iteratively the transient states generated for and associated with individual nodes and edges (~ message passing).

Types of Graph NNs (GNNs)

- Message-passing
- Graph-2-graph transducers
- Generic models (e.g. Battaglia et al.)

Message-passing GNNs: A use case

Graph Representations for Higher-Order Logic and Theorem Proving
Aditya Paliwal, Sarah Loos, Markus Rabe, Kshitij Bansal, Christian Szegedy
https://arxiv.org/abs/1905.10006

Application area

- The goal: To improve the effectiveness of theorem provers.
- Key question/challenge: How to navigate in the vast space of premises (and corresponding paths of proof) during theorem proving?

The benchmark:

- HOList (Bansal et al. 2019): stateless theorem proving API https://sites.google.com/view/holist/home
- More than 20k mathematical theorems and their proofs (human-constructed).
- The HOList proof assistant allows ML algorithms to interact with the HOL Light interactive theorem prover.

The hol-light database

```
let fib = define
 `fib n = if n = 0 \/ n = 1 then 1 else fib(n - 1) + fib(n - 2)`;;

let fib2 = define
 `(fib2 0 = 1) /\
  (fib2 1 = 1) /\
  (fib2 (n + 2) = fib2(n) + fib2(n + 1))`;;

let halve = define `halve (2 * n) = n`;;

let unknown = define `unknown n = unknown(n + 1)`;;

define
 `!n. collatz(n) = if n <= 1 then n
                   else if EVEN(n) then collatz(n DIV 2)
                   else collatz(3 * n + 1)`;;
```

```
ARITH_RULE
 `(a * x + b * y + a * y) EXP 3 + (b * x) EXP 3 +
  (a * x + b * y + b * x) EXP 3 + (a * y) EXP 3 =
  (a * x + a * y + b * x) EXP 3 + (b * y) EXP 3 +
  (a * y + b * y + b * x) EXP 3 + (a * x) EXP 3`;;
```

https://github.com/brain-research/hol-light/blob/master/Tutorial/Recursive_definitions.ml

Data representation

S-expressions, e.g.: f(x) is represented as

(a (v (fun A B) f) (v A x)))

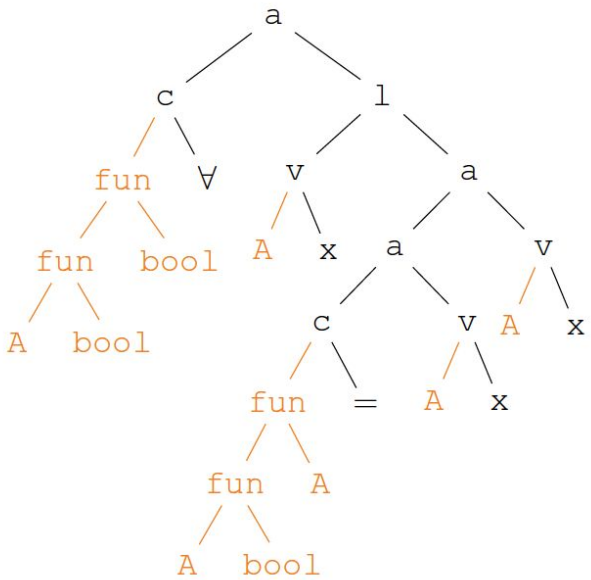
- a: function application
- v: is-a-variable; (v A a) states “a is a variable of type A”
- A, B: input and output types

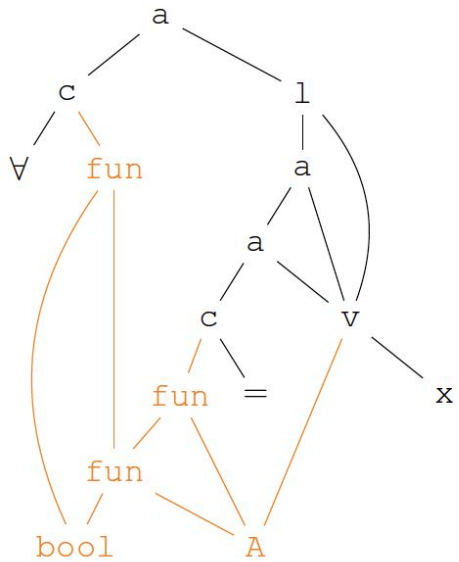
Important: The authors consider also variants with bottom-up edges (each child points to its parents).

Motivation for graphs: TreeNNs focus on each node capturing only the properties of the subgraph rooted in that node.

ASTs: Tree vs. graph

Representation of the axiom: $\forall x : x = x.$





Graph representations are more compact by avoiding subtree duplication (which then requires potentially costly testing for equivalence).

Scope of experiments

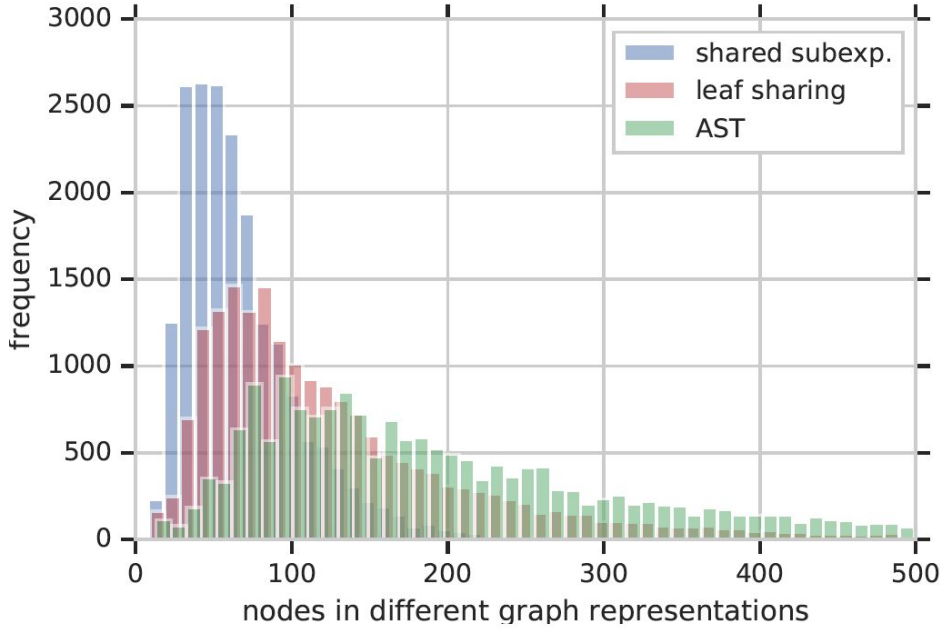
Input graphs can be transformed before feeding into a GNN.

Transformations considered by the authors:

- Sharing nodes of syntactically equal subexpressions (*subexpression sharing*).
- Sharing equal leaves of the AST (*leaf sharing*).
- Replacing all variable names by x (*variable blinding*).
- Removing all edges from nodes to their parents and only keeping those to their children (*top down*).
- Removing all edges from nodes to their children and only keeping those to their parents (*bottom up*).
- Adding random edges (*random*).

Impact of leaf and subexpression sharing

Statistics of the HOList database:



Core component: Message-passing GNN

Graph neural networks (GNNs) compute embeddings for nodes in a graph via consecutive rounds (also called hops) of end-to-end differentiable message passing.

- Iterative aggregation (*t*) of information ‘received’ by each node from its neighbors.
- The number of iterations (hops) *T* determines the effective size of the neighborhood (‘receptive field’).
- Individual mappings realized with multi-layer perceptrons (MLP).
- The resulting graph has exactly the same topology as the input one.
    - Extra steps required to produce output for classification, regression, etc.

Next slides describe the steps of a single iteration of message passing.

Graph definition

$$G = (V, E, l_V, l_E)$$

where:

- *V* is a set of nodes,
- *E* is a set of directed edges,
- l_V maps nodes to a fixed vocabulary of tokens,
- l_E maps each edge e to a single scalar indicating if the edge is to (or from) the first or the second child*, encoded as 0 and 1 (recall: we consider directed graphs, in general)

*child = node in this context

Definition of l_E can be extended for graphs with labels on edges.

Propagation: Step 1

Separate embedding of each node *v* and edge *e* individually:

$$\boldsymbol{h}_v^1 = \mathrm{MLP}_V(\boldsymbol{x}_{l_V(v)})$$

$$\boldsymbol{h}_e = \mathrm{MLP}_E(l_E(e))$$

where:

- x_t: trainable feature vector associated with t-th token (token/label embedding)
- l_E: binary variable indicating the direction of the edge

This paper: $|h_v|=128$

Propagation: Step 2

Generation of messages: for each edge (u,v) aggregates

- the embeddings for u, v,
- and the edge that connects them,

separately for the parent (incoming) nodes (first formula) and the child (outgoing) nodes (second formula):

$$\boldsymbol{s}_{u,v}^{t} = \mathrm{MLP}_{\mathrm{edge}}^{t}([\boldsymbol{h}_{u}^{t-1}, \boldsymbol{h}_{v}^{t-1}, \boldsymbol{h}_{e}])$$

$$\hat{\boldsymbol{s}}_{u,v}^{t} = \hat{\mathrm{MLP}}_{\mathrm{edge}}^{t}([\boldsymbol{h}_{u}^{t-1}, \boldsymbol{h}_{v}^{t-1}, \boldsymbol{h}_{e}])$$

Propagation: Step 3

Aggregation of messages for each node separately, based on averaged messages:

$$\boldsymbol{h}_v^t = \boldsymbol{h}_v^{t-1} + \mathrm{MLP}_{\mathrm{aggr}}\left(\left[\boldsymbol{h}_v^{t-1}, \sum \frac{\boldsymbol{s}_{u,v}^t}{p(v)}, \sum \frac{\hat{\boldsymbol{s}}_{u,v}^t}{c(v)}\right]\right)$$

where:

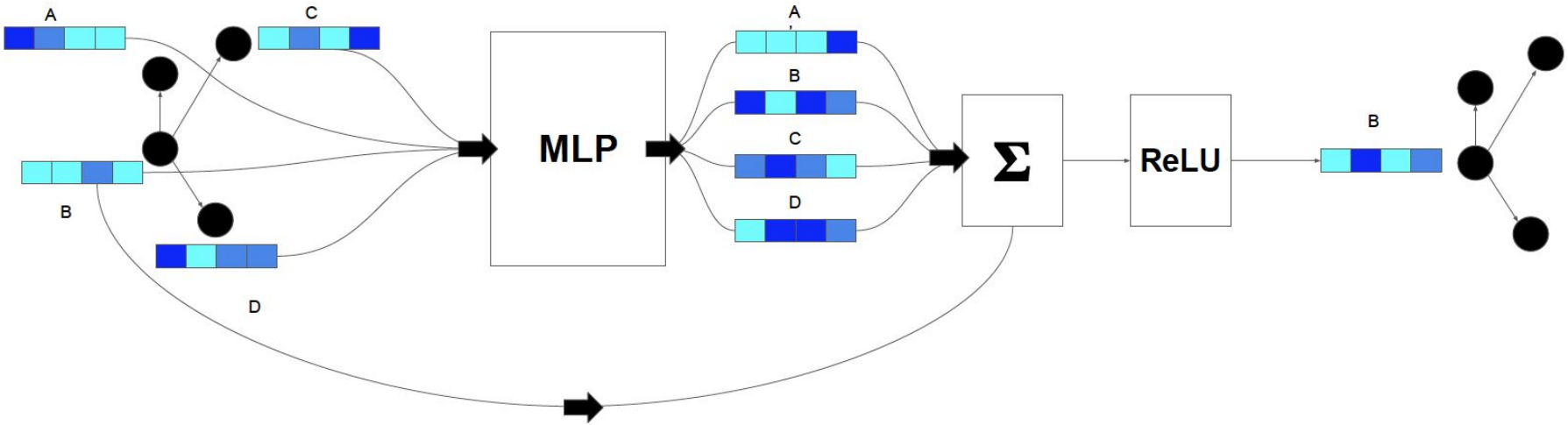
- p(v): the number of parents of v
- c(v): the number of children of v

Comments:

- Just five MLPs in total.
- The whole formalism becomes even simpler for undirected graphs.
- Notice the incremental character of the formula.

Message passing: Example (node B)

Single (*t*-th) step of processing:



- MLPs applied independently to each node and edge.
- What does the lowermost arrow remind you of?

Comments

Model's perception of the actual graph structure is quite limited. Recall the averaging in the aggregation step:

$$\boldsymbol{h}_v^t = \boldsymbol{h}_v^{t-1} + \text{MLP}_{\text{aggr}}\left(\left[\boldsymbol{h}_v^{t-1}, \sum \frac{\boldsymbol{s}_{u,v}^t}{p(v)}, \sum \frac{\hat{\boldsymbol{s}}_{u,v}^t}{c(v)}\right]\right)$$

Limitations:

- The aggregation network has no access to the number of adjacent nodes.
- Different sets of messages will be mapped to the some output.

Question:

- What type of architecture is this message passing GNN?
- How to set T? What is its relationship with graph size?

Overall architecture for proof search

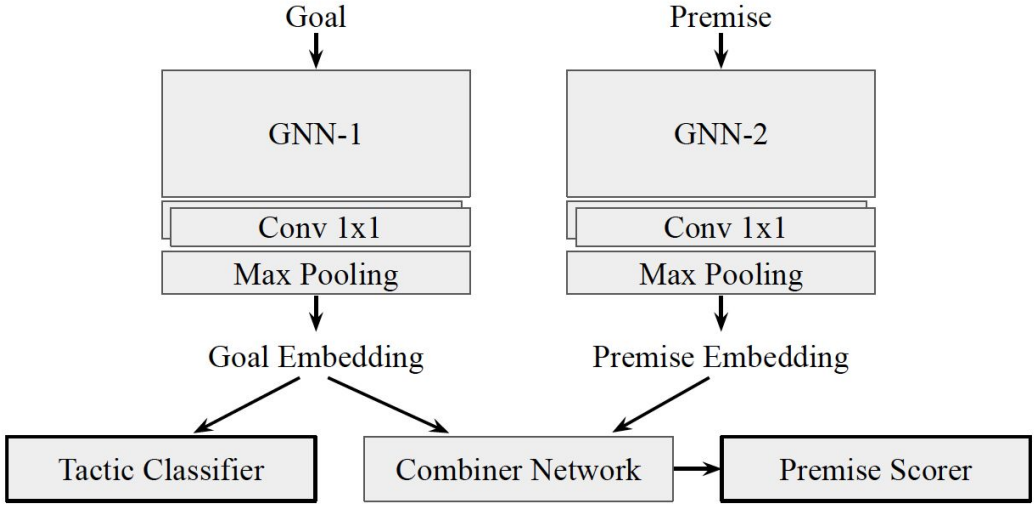
The approach uses two message-passing GNNs (same architecture, distinct weights) for:

1. Scoring the 41 predefined tactics used by the prover (Tactic Classifier)
2. Scoring the premises to follow in the next step of proof (Premise Scorer)

- The input to TC is only the current goal (an intermediate state in proof search)
- PS takes into account both the current goal and a candidate premise

- Node embedding used in both GNNs is relatively small (128), so the authors expand them afterwards to 1024 using 1x1 convolutions.
- Cross-entropy for permise scoring.

Overall architecture



- GNNs operate as graph embeddings
  - Not to be confused with node and edge embeddings used by GNNs internally!
- Interesting combiner network: concatenates both inputs and their elementwise multiplication, and then applies 3 FC layers.
- GNN-2 applied to premises off-line (?), so only the combiner network has to be applied to all goal-premise combinations for a given (sub)goal

Comments

- Learning mode: Imitation learning: the system learns to mimic the behavior of a human prover.
  - Notice the difference with supervised learning (?)
  - A plausible alternative: reinforcement learning
- Training set derived from 10,200 top-level theorems.
  - ~375,000 proof steps (subgoals)
- Test set: 3,225 theorems.
- Each positive examples is a triple fetched from a human proof:
  - the (sub)goal to be proven (at that step),
  - the tactic applied to it
  - a list of theorem parameters passed to the tactic (or special token for no parameter).
- Negative examples: random pairing of goals with theorems, for theorems that have been used at least once in the training set.

Loss function

Weighted combination of:

- cross-entropy loss of the tactic classifier,
- cross-entropy loss of pairwise premise scorer, and
- AUCROC loss (Burges et al. 2005; Eban et al. 2017):

$$AUCROC_b = \sum_i \sum_j loss(logit_i - logit_j)$$

$$loss(l) = \ln(1 + e^{-l})$$

where *i* ranges over the positive premises in batch *b*, and *j* ranges over the negatives in *b*.

Relation to ranking: an attempt to maximize the spread between the positives and negatives.

Recall: Scope of experiments

Input graphs can be transformed before feeding into a GNN.

Transformations considered by the authors:

- Sharing nodes of syntactically equal subexpressions (*subexpression sharing*).
- Sharing equal leaves of the AST (*leaf sharing*).
- Replacing all variable names by x (*variable blinding*).
- Removing all edges from nodes to their parents and only keeping those to their children (*top down*).
- Removing all edges from nodes to their children and only keeping those to their parents (*bottom up*).
- Adding random edges (*random*).

Some results

The metric: Percentage of proofs *closed* on the validation set: running the prover once over the validation set using the parameterized tactics predicted by the checkpoint to guide proof search.

Table 1: Our best model, compared against previous state of the art and bag of words model as baselines.

Network Architecture	% Proofs Closed (Validation Set)
Baseline: S-expression as a string	
WaveNet (Bansal et al. 2019)	32.65%
Baseline: Bag of words	
Max pooling only	37.98%
Subexpression sharing	
12-hop GNN	**49.95%**

Representation	**0 Hops**	**2 Hops**	**4 Hops**	**8 Hops**	**12 Hops**
Abstract syntax trees (AST)	40.18%	43.84%	44.58%	46.66%*	45.67%*
Leaf sharing	41.76%	33.89%	29.24%	29.51%	30.51%
Leaf sharing + variable blinding	31.78%	32.18%	32.80%	30.04%	31.00%
Subexpression sharing	40.86%	42.94%	46.94%	47.22%	**49.95%**
Subexpression sharing + variable blinding	31.75%	34.44%	35.96%	34.07%	37.36%
Subexpression sharing + random	41.24%	43.68%	43.84%	42.63%	42.94%
Subexpression sharing + top down	40.55%	43.59%	45.51%	48.24%	48.40%
Subexpression sharing + bottom up	39.72%	40.58%	41.16%	41.86%	40.99%

Conclusions: More hops helps. Subexpression sharing is essential.

Covariant graph networks

Predicting molecular properties with covariant compositional networks
Truong Son Hy, Shubhendu Trivedi, Horace Pan, Brandon M. Anderson, and Risi Kondor
https://doi.org/10.1063/1.5024797

Covariant Compositional Networks For Learning Graphs
Risi Kondor, Hy Truong Son, Horace Pan, Brandon Anderson, Shubhendu Trivedi
http://arxiv.org/abs/1801.02144

Summary: Address the question: which types of invariances should we care about when processing graphs?

Motivations

The two major issues that graph learning methods need to grapple with are:

- invariance to permutations,
- capturing structures at multiple different scales.

Claim: *Invariant aggregation functions, of the type popularized by message passing neural networks, are not the most general way to build compositional models for compound objects, such as graphs*.

For instance:

- In CNNs, if the input image is translated, the activations in each layer translate in the same way (barring pooling etc.)
- This quasi-invariance is called *equivariance*.

Quasi-invariance in Comp-nets

Quasi-invariance amounts to asserting that the activation f i at any given node must only depend on Pi = (ej1 , ... , ejk) as a *set* and *not on the internal ordering of the atoms* ej1 , ... , ejk making up the receptive field.

However, this property is potentially problematic since we lose all information about which vertex (in the receptive field) has contributed what to the aggregate information (activation).

The solution is to regard the Pi receptive fields as ordered sets and explicitly establish how f i co-varies with the internal ordering of the receptive fields.

Formal definition of covariance

Definition 2. *Assume that* $\mathcal{N}$ *is the comp-net of a graph* $\mathcal{G}$ *and* $\mathcal{N}'$ *is the comp-net of the same graph but after its vertices have been permuted by some permutation* σ. *Given any* $\mathfrak{n}_i \in \mathcal{N}$ *with receptive field* $\mathcal{P}_i = \left(e_{p_1}, \ldots, e_{p_m}\right)$, *let* $\mathfrak{n}'_j$ *be the corresponding node in* $\mathcal{N}'$ *with receptive field* $\mathcal{P}'_j = \left(e_{q_1}, \ldots, e_{q_m}\right)$. *Assume that* $\pi \in \mathbb{S}_m$ *is the permutation that aligns the orderings of the two receptive fields, i.e., for which* $e_{q_{\pi(a)}} = e_{p_a}$. *We say that the comp-nets are* ***covariant to permutations*** *if for any* π, *there is a corresponding function* R_π *such that* $f'_j = R_\pi(f_i)$.

Implementation

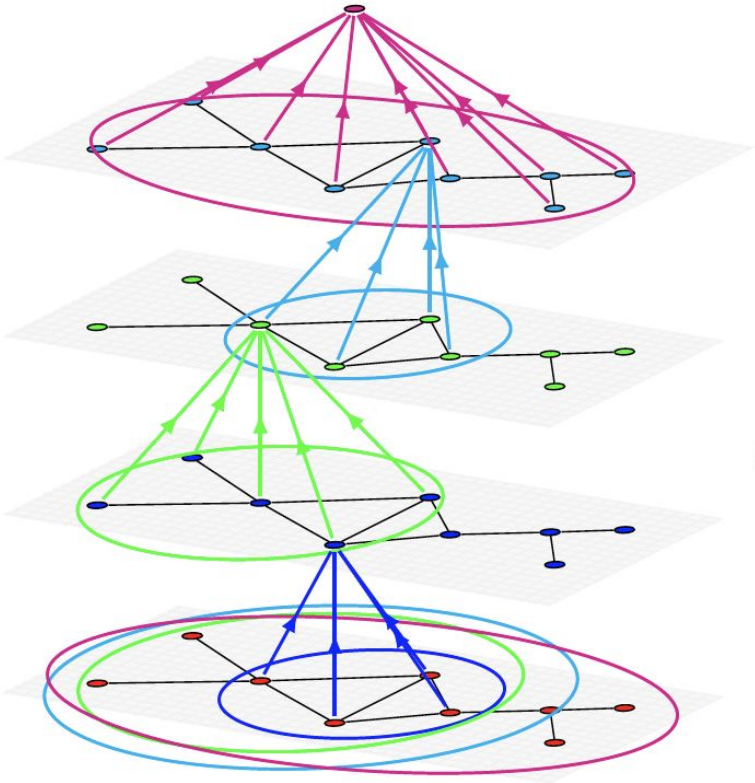
Uses k-order tensors and permutation matrices to implement the covariance.

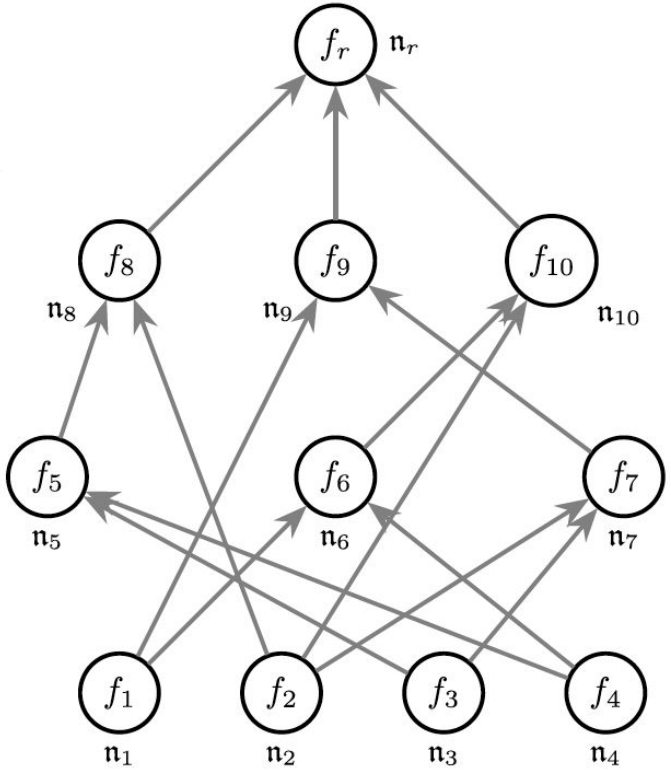
$$[P_\pi]_{i,j} = \begin{cases} 1 & \text{if } \pi(j) = i, \\ 0 & \text{otherwise.} \end{cases}$$

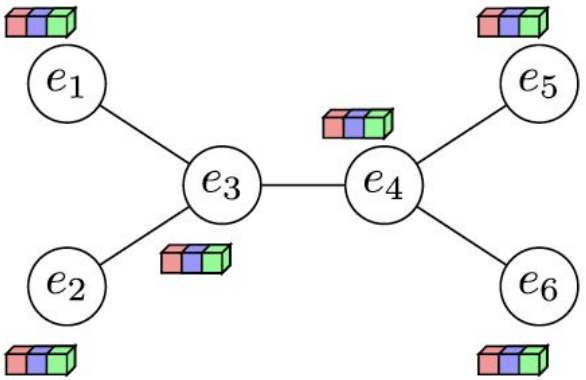
Represents excitations as vectors and `stacks' permutation matrices using Kronecker product:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 1\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 2\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \\ 3\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} & 4\begin{bmatrix} 0 & 5 \\ 6 & 7 \end{bmatrix} \end{bmatrix}$$

Receptive fields

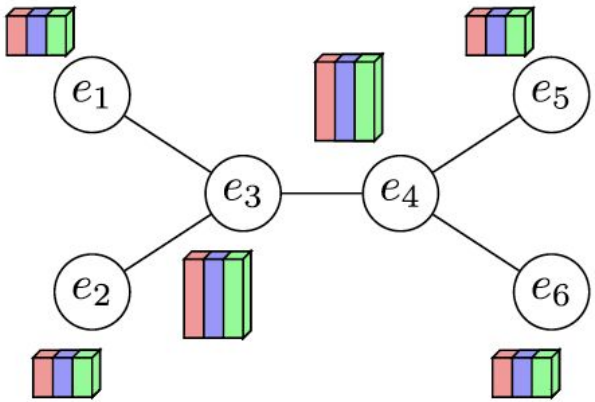






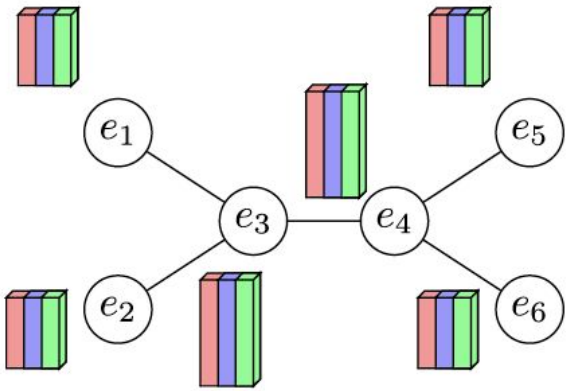
$\ell = 0$

(a)



$\ell = 1$

(b)



$\ell = 2$

(c)

Benchmarks

Classification:

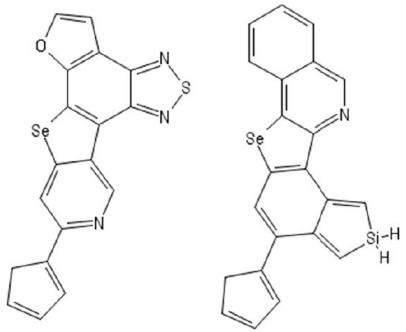
1. MUTAG: 188 mutagenic aromatic and heteroaromatic compounds
2. PTC: 344 chemical compounds that have been tested for positive or negative toxicity in lab rats
3. NCI1 and NCI109: 4110 and 4127 compounds, respectively, each screened for activity against small cell lung cancer and ovarian cancer lines.

Regression:

- Harvard Clean Energy Project (HCEP): 2.3M organic candidate compounds for use in solar cells
    - Inputs: molecular graphs (derived from their SMILES strings),
    - Regression target: power conversion efficiency (PCE).

Other: QM9: 134K organic molecules.

- Target: 13 molecular properties (multiple regression)



Some results

TABLE I. Classification of results on the kernel datasets (accuracy ± standard deviation).

	MUTAG	PTC	NCI1	NCI109
Wesifeiler–Lehman kernel[33]	84.50 ± 2.16	59.97 ± 1.60	84.76 ± 0.32	85.12 ± 0.29
Wesifeiler–Lehman edge kernel[33]	82.94 ± 2.33	60.18 ± 2.19	84.65 ± 0.25	85.32 ± 0.34
Shortest path kernel[29]	85.50 ± 2.50	59.53 ± 1.71	73.61 ± 0.36	73.23 ± 0.26
Graphlets kernel[31]	82.44 ± 1.29	55.88 ± 0.31	62.40 ± 0.27	62.35 ± 0.28
Random walk kernel[28]	80.33 ± 1.35	59.85 ± 0.95	Timed out	Timed out
Multiscale Laplacian graph kernel[62]	87.94 ± 1.61	63.26 ± 1.48	81.75 ± 0.24	81.31 ± 0.22
PSCN($k = 10$)[37]	88.95 ± 4.37	62.29 ± 5.68	76.34 ± 1.68	N/A
Neural graph fingerprints[21]	89.00 ± 7.00	57.85 ± 3.36	62.21 ± 4.72	56.11 ± 4.31
Second order CCN (our method)	91.64 ± 7.24	70.62 ± 7.04	76.27 ± 4.13	75.54 ± 3.36

HCEP

TABLE II. HCEP regression results. Error of predicting power conversion efficiency in units of percent. (Best results indicated in bold.)

	Test MAE	Test RMSE
Lasso	0.867	1.437
Ridge regression	0.854	1.376
Random forest	1.004	1.799
Gradient boosted trees	0.704	1.005
Weisfeiler–Lehman kernel[33]	0.805	1.096
Neural graph fingerprints[21]	0.851	1.177
PSCN ($k = 10$)[37]	0.718	0.973
Second order CCN (our method)	**0.340**	**0.449**

QM9

Properties:

(a) U_0: atomization energy at 0 K (eV),
(b) U: atomization at room temperature (eV),
(c) H: enthalpy of atomization at room temperature (eV),
(d) G: free energy of atomization (eV),
(e) ω_1: highest fundamental vibrational frequency (cm^{-1}),
(f) ZPVE: zero point vibrational energy (eV),
(g) HOMO: highest occupied molecular orbital, energy of the highest occupied electronic state (eV),
(h) LUMO: lowest unoccupied molecular orbital, energy of the lowest unoccupied electronic state (eV),
(i) GAP: difference between HOMO and LUMO (eV),
(j) R^2: electronic spatial extent ($bohr^2$),
(k) μ: norm of the dipole moment (D),
(l) α: norm of the static polarizability ($bohr^3$),
(m) C_v: heat capacity at room temperature (cal/mol/K).

TABLE III. QM9(a) regression results (mean absolute error). Here we have only used the graph as the learning input without any physical features. (Best results indicated in bold.)

	WLGK	NGF	PSCN	CCN 2D
α ($bohr^3$)	3.75	3.51	1.63	**1.30**
C_v [cal/(mol K)]	2.39	1.91	1.09	**0.93**
G (eV)	4.84	4.36	3.13	**2.75**
GAP (eV)	0.92	0.86	0.77	**0.69**
H (eV)	5.45	4.92	3.56	**3.14**
HOMO (eV)	0.38	0.34	0.30	**0.23**
LUMO (eV)	0.89	0.82	0.75	**0.67**
μ (D)	1.03	0.94	0.81	**0.72**
ω_1 (cm^{-1})	192.16	168.14	152.13	**120.10**
R_2 ($bohr^2$)	154.25	137.43	61.70	**53.28**
U (eV)	5.41	4.89	3.54	**3.02**
U_0 (eV)	5.36	4.85	3.50	**2.99**
ZPVE (eV)	0.51	0.45	0.38	**0.35**

Attempts of unification

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
https://arxiv.org/abs/1806.01261

Motivation

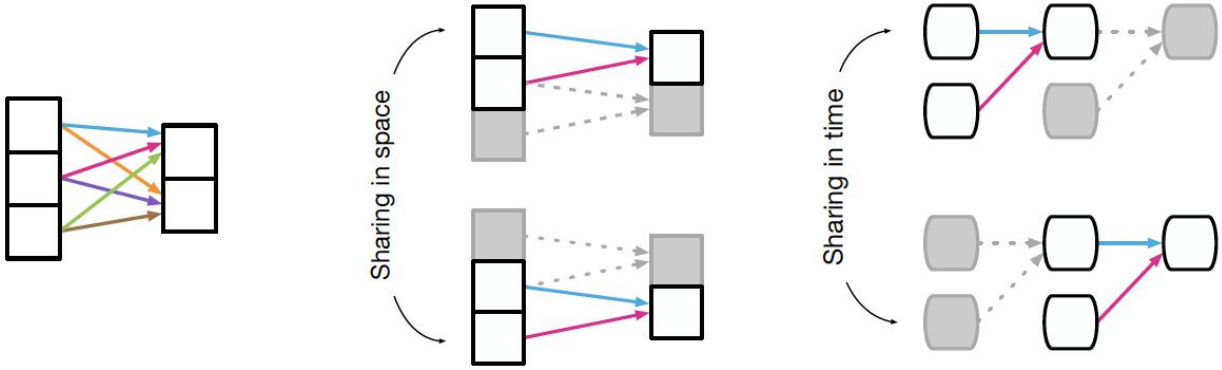
Can we define an overarching formalism for Graph NNs?

What are the types of invariance and relational inductive biases embodied by various NN architectures?

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

Table 1: Various relational inductive biases in standard deep learning components. See also Section 2.

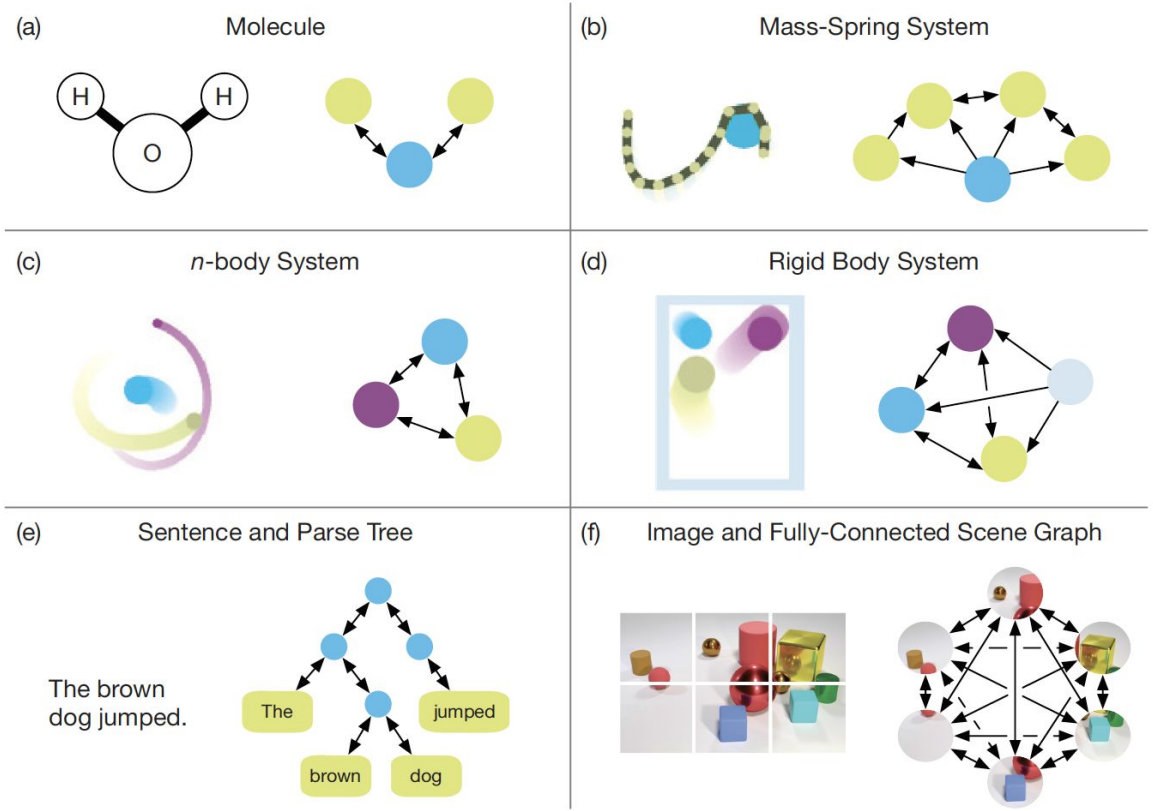
Types of weigth sharing in NNs and DL



(a) Fully connected (b) Convolutional (c) Recurrent

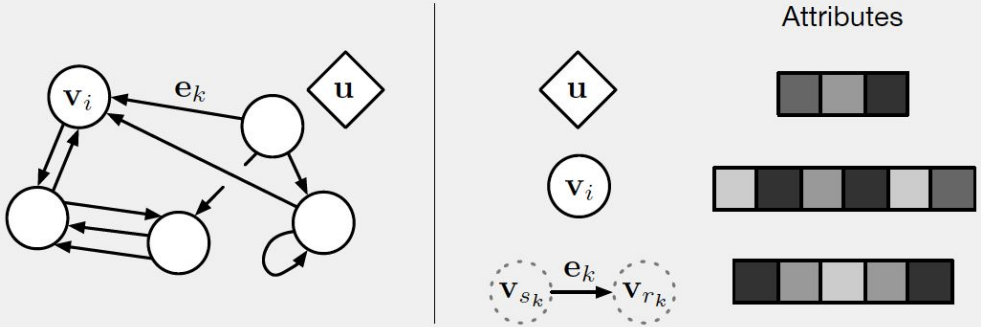
Figure 1: Reuse and sharing in common deep learning building blocks. (a) Fully connected layer, in which all weights are independent, and there is no sharing. (b) Convolutional layer, in which a local kernel function is reused multiple times across the input. Shared weights are indicated by arrows with the same color. (c) Recurrent layer, in which the same function is reused across different processing steps.

Different graph representations



Definition of ‘extended graph’

Box 3: Our 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:

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.

Figure 2 shows a variety of different types of graphs corresponding to real data that we may be interested in modeling, including physical systems, molecules, images, and text.

Signal propagation algorithm

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

    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\rightarrow v}\left(E'_i\right)$ ▷ 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 $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$

  $\bar{\mathbf{e}}' \leftarrow \rho^{e\rightarrow u}\left(E'\right)$ ▷ 4. Aggregate edge attributes globally

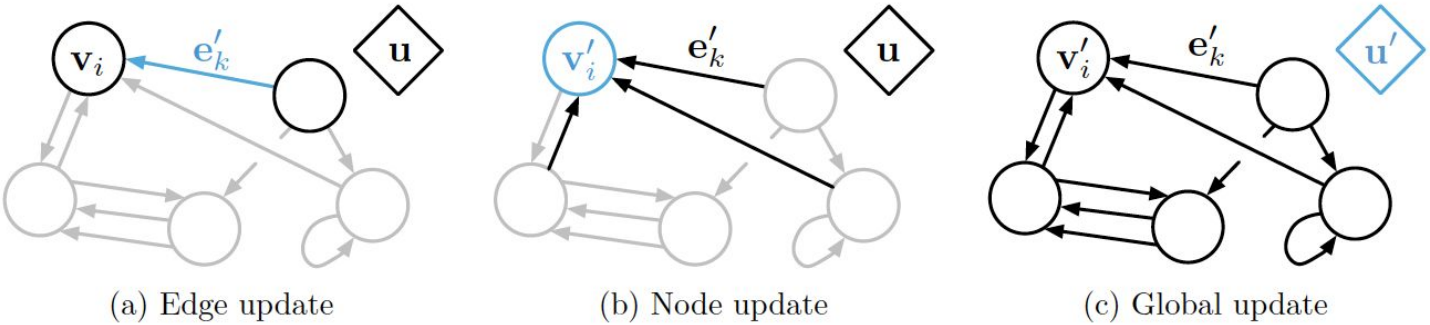
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\rightarrow u}\left(V'\right)$ ▷ 5. Aggregate node attributes globally

  $\mathbf{u}' \leftarrow \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right)$ ▷ 6. Compute updated global attribute

  return $(E', V', \mathbf{u}')$

end function

Updates in a GN block



(a) Edge update (b) Node update (c) Global update

Blue indicates the element that is being updated, and black indicates other elements which are involved in the update.

- Note that the pre-update value of the blue element is also used in the update.

Summary

- A rather conceptual work.
  - No experimental validation.
- Unifying perspective
- The authors how a few major earlier architectures can be realized within the GN framework:
  - Message-passing NN
  - Non-local neural networks (NLNN)
  - Relation Networks
  - Deep Sets
  - PointNet
  - ...