Battaglia et al. 2018: Relational inductive biases, deep learning, and graph networks

Minqi Pan

April 28, 2020

Relational inductive biases, deep learning, and graph networks

- arXiv:1806.01261 (Submitted on 4 Jun 2018, last revised 17 Oct 2018)
- 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
- DeepMind, Google Brain, MIT, University of Edinburgh

Outline

Outline

GNN Usage (1)

- Scene understanding
- Few-shot learning
- Learning the dynamicis of physical systems and multi-agent systems
- Reasoning about knowledge graphs
- Predicting the chemical properties of molecules
- Predicting traffic on roads
- Classifying and segmenting images and videos, 3D meshes and point clouds

GNN Usage (2)

- Classifying regions in images
- Performing semi-supervised text classfication
- Machine translation
- Model-free continous control
- Model-based continuous control
- Model-free reinforcement learning
- More classical approaches to planning

GNN Usage in Traditional CS Problems

- Combinatorial optimization
- Boolean satisfiabiliity
- Program representation and verification
- Modeling cellular automata and Turing machines
- Performing inference in graphical models

Recent Work

- Building generative models of graphs
- Unsupervised learning of graph embeddings

Outline

The GN framework

- The GN framework defines a class of functions for relational reasoning over graph-structured representations
- The GN framework generalizes and extends various GNN, MsgPassingNN and NonLocalNN approaches, and supports constructing complex architectures from simple building blocks
- Term “neural” dropped to reflect that they can be implemented with functions other than NN, though here our focus is on NN implementations

GN Block

- GN Block is a “graph2graph” module which takes a graph as input, performs computations over the structure, and returns a graph as output
- GN Block emphasizes customizability and synthesizing new architectures which express desired relational inductive biases
- Key design principles are:
    - Flexible Representations
    - Configurable within-block structure
    - Composable multi-block architectures

The Definition of “graph”

- “Graph” G is a directed, attributed multi-graph (there can be more than 1 edge between vertices, including self-edges) with a global attribute (graph-level properties that can be encoded as a vector, set, or even another graph):

$$\begin{aligned} G &= (u, V, E) \\ V &= \{v_i\}_{i=1:N^v} \\ E &= \{(e_k, r_k, s_k)\}_{k=1:N^e} \end{aligned}$$

- u: the global attributes
- v_i: a node
- e_k: an edge
- s_k, r_k: the sender, receiver nodes of e_k
- Fig. 2

An example

- Consider predicting the movements a set of rubber balls in an arbitrary gravitational
  eld
- Instead of bouncing against one another, each have one or more springs which connect them to some (or all) of the others
- u: the gravitational field
- V: each ball with attributes for position, velocity and mass
- E: the presence of springs between different balls and their corresponding spring constants

Internal Structure of a GN Block

$$
\begin{aligned}
e'_k &= \phi^e(e_k, v_{r_k}, v_{s_k}, u) \\
E'_i &= \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e} \\
\bar{e}'_i &= \rho^{e\rightarrow v}(E'_i) \\
v'_i &= \phi^v(\bar{e}'_i, v_i, u) \\
E' &= \cup_i E'_i = \{(e'_k, r_k, s_k)\}_{k=1:N^e} \\
\bar{e}' &= \rho^{e\rightarrow u}(E') \\
V' &= \{v'_i\}_{i=1:N^v} \\
\bar{v}' &= \rho^{v\rightarrow u}(V') \\
u' &= \phi^u(\bar{e}', \bar{v}', u)
\end{aligned}
$$

- A GN block contains three “update” functions, ϕ, and three “aggregation” functions, ρ

Internal Structure of a GN Block

$$
\begin{aligned}
e'_k &= \phi^e(e_k, v_{r_k}, v_{s_k}, u) \\
E'_i &= \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e} \\
\bar{e}'_i &= \rho^{e\to v}(E'_i) \\
v'_i &= \phi^v(\bar{e}'_i, v_i, u) \\
E' &= \cup_i E'_i = \{(e'_k, r_k, s_k)\}_{k=1:N^e} \\
\bar{e}' &= \rho^{e\to u}(E') \\
V' &= \{v'_i\}_{i=1:N^v} \\
\bar{v}' &= \rho^{v\to u}(V') \\
u' &= \phi^u(\bar{e}', \bar{v}', u)
\end{aligned}
$$

- ϕ^e: being mapped across all edges to compute per-edge updates

Internal Structure of a GN Block

$$
\begin{aligned}
e'_k &= \phi^e(e_k, v_{r_k}, v_{s_k}, u) \\
E'_i &= \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e} \\
\bar{e}'_i &= \rho^{e\to v}(E'_i) \\
v'_i &= \phi^v(\bar{e}'_i, v_i, u) \\
E' &= \cup_i E'_i = \{(e'_k, r_k, s_k)\}_{k=1:N^e} \\
\bar{e}' &= \rho^{e\to u}(E') \\
V' &= \{v'_i\}_{i=1:N^v} \\
\bar{v}' &= \rho^{v\to u}(V') \\
u' &= \phi^u(\bar{e}', \bar{v}', u)
\end{aligned}
$$

- ϕ^v: being mapped across all nodes to compute per-node updates

Internal Structure of a GN Block

$$
\begin{aligned}
e'_k &= \phi^e(e_k, v_{r_k}, v_{s_k}, u) \\
E'_i &= \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e} \\
\bar{e}'_i &= \rho^{e\rightarrow v}(E'_i) \\
v'_i &= \phi^v(\bar{e}'_i, v_i, u) \\
E' &= \cup_i E'_i = \{(e'_k, r_k, s_k)\}_{k=1:N^e} \\
\bar{e}' &= \rho^{e\rightarrow u}(E') \\
V' &= \{v'_i\}_{i=1:N^v} \\
\bar{v}' &= \rho^{v\rightarrow u}(V') \\
u' &= \phi^u(\bar{e}', \bar{v}', u)
\end{aligned}
$$

- ϕ^u: being applied once as the global update

Internal Structure of a GN Block

$$\begin{aligned}
e'_k &= \phi^e(e_k, v_{r_k}, v_{s_k}, u) \\
E'_i &= \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e} \\
\bar{e}'_i &= \rho^{e\rightarrow v}(E'_i) \\
v'_i &= \phi^v(\bar{e}'_i, v_i, u) \\
E' &= \cup_i E'_i = \{(e'_k, r_k, s_k)\}_{k=1:N^e} \\
\bar{e}' &= \rho^{e\rightarrow u}(E') \\
V' &= \{v'_i\}_{i=1:N^v} \\
\bar{v}' &= \rho^{v\rightarrow u}(V') \\
u' &= \phi^u(\bar{e}', \bar{v}', u)
\end{aligned}$$

- ρ: taking a set as input and reducing it to a single element which represents the aggregated information

Internal Structure of a GN Block

$$
\begin{aligned}
e'_k &= \phi^e(e_k, v_{r_k}, v_{s_k}, u) \\
E'_i &= \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e} \\
\bar{e}'_i &= \rho^{e\rightarrow v}(E'_i) \\
v'_i &= \phi^v(\bar{e}'_i, v_i, u) \\
E' &= \cup_i E'_i = \{(e'_k, r_k, s_k)\}_{k=1:N^e} \\
\bar{e}' &= \rho^{e\rightarrow u}(E') \\
V' &= \{v'_i\}_{i=1:N^v} \\
\bar{v}' &= \rho^{v\rightarrow u}(V') \\
u' &= \phi^u(\bar{e}', \bar{v}', u)
\end{aligned}
$$

- ρ MUST be invariant to permutations of their inputs, and should take variable number of arguments

Internal Structure of a GN Block

$$\begin{aligned}
e'_k &= \phi^e(e_k, v_{r_k}, v_{s_k}, u) \\
E'_i &= \{(e'_k, r_k, s_k)\}_{r_k=i,k=1:N^e} \\
\bar{e}'_i &= \rho^{e\to v}(E'_i) \\
v'_i &= \phi^v(\bar{e}'_i, v_i, u) \\
E' &= \cup_i E'_i = \{(e'_k, r_k, s_k)\}_{k=1:N^e} \\
\bar{e}' &= \rho^{e\to u}(E') \\
V' &= \{v'_i\}_{i=1:N^v} \\
\bar{v}' &= \rho^{v\to u}(V') \\
u' &= \phi^u(\bar{e}', \bar{v}', u)
\end{aligned}$$

- ρ: e.g. element-wise summation, mean, maximum, etc.

Computational Steps within a GN Block

- When a graph G is provided as input to a GN block, the computations...
  - proceed from the edges
  - to the node
  - to the global level
- Figure 3
- Figure 4

Computational Steps within a GN Block (1)

- ϕ^e is applied per edge, with arguments $(e_k, v_{r_k}, v_{s_k}, u)$, and returns e'_k
- E.g. this mighht correspond to the forces or potential energies between two connected balls
- The set of resulting per-edge outputs for each node i is $E_i = \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$
- $E' = \cup_i E'_i$ is the set of all per-edge outputs

Computational Steps within a GN Block (2)

- $\rho^{e \to v}$ is applied to E'_i, and aggregates the edge updates for edges that project to vertex i, into $\bar{e}'_i$, which will be used in th next step's node update
- E.g. summing all the forces or potential energies acting on the i-th ball

Computational Steps within a GN Block (3)

- ϕ^v is applied to each node i, to compute an updated node attribute, v_i'
- E.g. the updated position, velocity, and kinetic energy of each ball
- The set of resulting per-node output is

$$V' = \{v_i'\}_{i=1:N^v}.$$

Computational Steps within a GN Block (4)

- $\phi^{e \to u}$ is applied to E', and aggregates all edge updates, into $\bar{e}'$, which will then be used in the next step's global update
- E.g. $\rho^{e \to u}$ may be compute the summed forces (which should be zero, in this case, due to Newton's 3rd law) and the springs' potential energies

Computational Steps within a GN Block (5)

- $\rho^{v \rightarrow u}$ is applied to V'
- And aggregates all node updates into $\bar{v}'$
- This will then be used in thhe next step's global update
- E.g. compute the total kinetic energy of the system

Computational Steps within a GN Block (6)

- ϕ^u is applied once per graph
- And computes an update for the global attribute u'
- E.g. compute something analogous to the net forces and total energy of the physical system

```
**function** $\textsc{GraphNetwork}(E, V, u)$
    **for** $k \in \{1, \ldots, N^e\}$ **do**
        $e'_k \leftarrow \phi^e(e_k, v_{r_k}, v_{s_k}, u)$
    **end for**
    **for** $i \in \{1, \ldots, N^n\}$ **do**
        Let $E'_i = \{(e'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$
        $\bar{e}'_i \leftarrow \rho^{e \rightarrow v}(E'_i)$
        $v'_i \leftarrow \phi^v(\bar{e}'_i, v_i, u)$
    **end for**
    Let $V' = \{v'\}_{i=1:N^v}$
    Let $E' = \{(e'_k, r_k, s_k)\}_{k=1:N^e}$
    $\bar{e}' \leftarrow \rho^{e \rightarrow u}(E')$
    $\bar{v}' \leftarrow \rho^{v \rightarrow u}(V')$
    $u' \leftarrow \phi^u(\bar{e}', \bar{v}', u)$ **return** $(E', V', u')$
**end function**
```

Outline

Flexible Attribute Representations

- The global, node and edge attributes of a GN block can use arbitrary representational formats
- In deep learning implementations, real-valued vectors and tensors are most common
- Other data structures such as sequences, sets, or even graphs could be also be used

Flexible Attribute Representations

- The requirements of the problem will often determine what representations should be used for the attributes
- E.g. when the input data is an image, the attributes might be represented as tensors of image patches
- When the input data is a text document, the attributes might be sequences of words corresponding to sentences

Flexible Attribute Representations

- For each GN block within a broader architecture
- The edge and node outputs typically correspond to lists of vectors or tensors, one per edge or node
- The global outputs correspond to a single vector or tensor
- This allows a GN's output to be passed to other deep learning building blocks such as MLPs, CNNs, and RNNs

Flexible Attribute Representations

- The output of a GN block can also be tailored to the demands of the task. In particular,
  - An "edge-focused" GN uses the edges as output, for example to make decisions about interactions among entities
  - A "node-focused" GN uses the nodes as ouput, for example to reason about physical systems
  - A "graph-focused" GN uses the globals as output, for example to predict the potential energy of a physical system, the properties of a molecule, or answers to questions about a visual scene
- The nodes, edges and global outputs can also be mixed-and-matched depending on the task
- E.g. use both the ouput edge and global attributes to compute a policy over actions

Flexible Graph Structures

- When defining how the input data will be represented as a graph, there are generally 2 scenarios
- First, the input explicitly specifies the relational structure
- Second, the relational structure must be inferred or assumed
- There are not hard distinctions, but extremes along a continuum

Flexible Graph Structures

- Examples of data with more explicitly specified entities and relations include Knowledge Graphs, Social Networks, Parse Trees, Optimization Problems, Chemical Graphs, Road Networks, and Physical Systems w/ known interactions
- Examples of data where the relational structure is not made explicit, and must be inferred or assumed, include Visual Scenes, Text Corpora, Programming Language Source Code, and Multi-agent Systems

Flexible Graph Structures

- In these types of settings, the data may be formatted as a set of entities without relations, or even just a vector or tensor (e.g., an image)
- If the entities are not specified explicitly, they might be assumed, for instance, by treating each word in a sentence or each local feature vector in a CNN's output feature map, as a node. Or, it might be possible to use a separate learned mechanism to infer entities from a unstructured signal
- If relations are not available, the simplest approach is to instantiate all possible directed edges between entities. This can be prohibitive for large number of entities, however, because the number of possible edges grows quadratically with the number of nodes

Outline

- The structure and functions within a GN block can be configured in different ways
- This offers flexibility in what information is made available as inputs to its functions, as well as how output edge, node, and global updates are produced
- In particular, each ϕ must be implemented with some function f, where f's argument signature determines what information it requires as input
- ϕ can be implemented via neural networks, e.g. MLP (for vector attributes) or CNN (for image features)
- ρ can be implemented using elementwise summation, but averages and max/min could also be used
- The ϕ functions can also use RNNs, which requires an additional hidden state as input and ouput
- Figure 4

Outline

- A key design principle of GN is constructing complex architectures by composing GN blocks
- We defined a GN block as always taking a graph comprised of edge, node, and global elements as input, and returning a graph with the same constituent
- Simply pass through the input elements to the output when those elements are not explicitly updated
- This graph2graph IO interface ensures that the output of one GN block can be passed as input to another, even if their internal configurations are different
- Similar to the tensor-to-tensor interface of the standard deep learning toolkit

- Arbitrary number of GN blocks can be composed
- Figure 6
- The blocks can be unshared (different functions and/or parameters, analogous to layers of a CNN) or shared (reused functions and parameters, analogous to an unrolled RNN)
- Shared configurations are analogous to MsgPassing, where the same local update procedure is applied iteratively to propagate information across the structure
- Figure 7

- If we exclude the global u (which aggregates info from across the nodes and edges), the info that a node has access to after m steps of propagation is determined by the set of nodes and edges that are at most m hops away
- This can be interpreted as breaking down a complex computation into smaller elmentary steps
- The steps can also be used to capture sequentiality in time
- E.g. if each propagation step predicts the physical dynamics over one time step of duration Δt, then the M propogation steps result in a total simulation of $M \cdot \Delta t$

- A common architecture design is what we call the "encode-process-decode" configuration
    - An input graph is transformed into a latent representation G_0 by an encoder
    - A shared core block is applied M times to return G_M
    - Finally an output graph is decoded
- E.g. the encoder might compute the initial forces annd interaction energies between the balls, the core might apply an elementary dynamics update, and the decoder might read out the final positions from the updated graph states

Recurrent GN

- Similar to the encode-process-decode design, recurrent GN-based architectures can be built by maintaining a hidden graph, taking as input an observed graph and returning an output graph on each step
- This type of architecture can be particularly useful for predicting sequences of graphs, such as predicting the trajectory of a dynamical system over time
- The "encoded graph" must have the same structure as the "hidden graph" and they can be easily combined by concatenating their corresponding e_k, v_i, u vectors before being passed to the core

Recurrent GN

- For the output, the hidden graph is copied and decoded
- This design reuses GN blocks in several ways: enc, dec and core blocks are shared across each step, t; and within each step, the core may perform multiple shared sub-steps

- Various other techniques for designing GN-based architectures can be useful
- Graph skip connections would concatenate a GN block's input graph G_m with its output graph G_{m+1} before proceeding to further computations
- Merging and smoothing input and hidden graph information can use LSTM- or GRU- style gating schemes, instead oof simple concatenation
- Or distinct, recurrent GN blocks can be composed before and/or after other GN blocks to improve stability in the representations over multiple propogation steps

Outline

- Similar to CNNs, which are naturally parallelizable (e.g. on GPUs), GNs have a natural parallel structure
- Since the ϕ^e and ϕ^v functions are shared over the edges and nodes respectively
- They can be computed in parallel
- In practice, this means that w.r.t. ϕ^e and ϕ^v, the nodes and edges can be treated like batch dimension in typical mini-batch training regimes
- Moreover, several graphs can be naturally batched together by treating them as disjoint components of a larger graph

- Reusing ϕ^e and ϕ^v also improves GNs' sample efficiency
- Again, analogous to a convolutional kernel, the number of samples which are used to optimize a GN's ϕ^e and ϕ^v functions is the number of edges and nodes, respectively, across all training graphs
- E.g. a scene with 4 balls which are all connected by springs will provide 12 examples of the contact interaction between them