A Gaze into the Internal Logic of Graph Neural Networks, with Logic

Paul Tarau

University of North Texas

AUGUST 2, 2022

ICLP'2022

Overview

- our *objective*: modeling with the help of a logic program the information flows involved in learning properties of new nodes
- the inference process distills information from:
    - the link structure of a graph
    - the information content of its nodes
- we emulate the learning via message passing mechanisms used by Graph Neural Networks (**GNNs**)
- we infer upper bounds on what can be learn from the graph's link structure and the content of its labeled nodes
- as a *practical outcome*, we obtain a logic program, that, when seen as machine learning algorithm, performs close to the state of the art on a *node property prediction* benchmark

commitment to fully *replicable* results: SWI-Prolog code at `https://github.com/ptarau/StanzaGraphs/tree/main/logic`

Node Property Prediction with GNNs

- GNNs learn by using:
  - the graph structure (that can be seen as a binary predicate describing the connections between nodes)
  - properties associated to nodes and edges (ground facts in a database in logic programming parlance)
- GNNs use message passing aggregating information from each node's neighbors
- they generalize Convolutional NNs (which can be seen as specialized graphs made of pixels neighboring other pixels)
- this information propagation mechanism is also similar to the (more general) relational reasoning steps of a logic program
- what we cannot replicate in LP: differentiable backpropagation (but we can use an effective weighted voting mechanism instead)

The Dataset: a Logical View

- our node property prediction task: the `ogbn-arxiv` citation network from the Open Graph Benchmark (OGB) dataset
- its leaderboard is at `https://ogb.stanford.edu/docs/leader_nodeprop/#ogbn-arxiv`
- the original OGB graph:
  1. a node identifier form `0 to 169342`, an embedding of the abstract of the paper associated to the node and (optionally) the actual summary
  2. a marker in the set `tr, va, te`, indicating if the fact belongs to the training, validation or test set
  3. a label indicating one of the 40 arxiv CS classes, from `0` to `39`
- the graph, as seen in Prolog, after preprocessing:
  1. a Prolog term representing a Directed Acyclic Graph (DAG), derived by merging the dependency trees of each sentence of the summary associated to each node
  2. a list of each node's neighbors, containing the node identifiers of the cited papers associated to the node
  3. the Prolog dataset is at: `http://www.cse.unt.edu/~tarau/datasets/arxiv_all.pro.zip`

Node Content: Merging Dependency Trees Into Text DAGs

TEXT: “Logic Programming and Functional Programming are declarative programming paradigms. Evaluation in Logic Programs is eager while for functional programs it may be lazy. As a common feature, unification is used for evaluation in Logic Programming and for type inference in Functional Programming.”

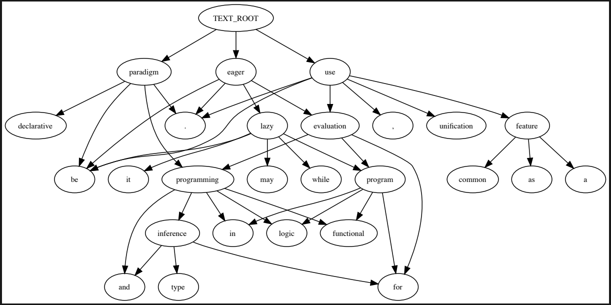

Figure: Plain dependency DAGs of the paragraph.

Building the Distilled Dependency DAG

- we start by reversing their dependency links
- we eliminate possible cycles via a DFS-tree extraction algorithm
- we run Pagerank on the reversed graph to filter out the terms unlikely to be relevant for comparing the content of two texts

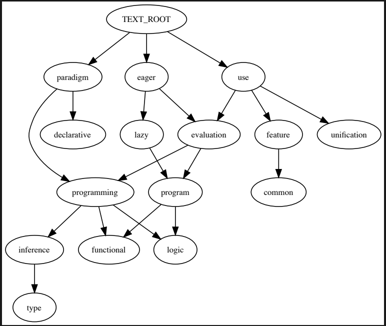

Figure: Distilled dependency DAGs of the paragraph.

The Label Inference Algorithm

- to infer an unknown label associated to a node:
  1. we analyze the labels of its references, when known
  2. when this information is not available, content similarity relations are used to propagate information from potentially any node in the graph
  3. a voting mechanism prioritizes the most relevant and content-wise close neighbors with known labels
- the Prolog code skeleton:

```
inferred_label(YtoGuess, YasGuessed):-
   param(max_neighbor_nodes,MaxNodes),
   param(neighbor_kind,NK), % NK is one of any, diverse, none
   most_freq_class(FreqClass),
   select_diverse_peers(Peers), % set of K peers for each label
   tester_at(N,YtoGuess,MyTextTerm,Neighbors),
(     NK\=none, at_most_n_sols(MaxNodes,YW,
       neighbor_data(NK,Peers,MyTextTerm,Neighbors,YW),YsAndWeights)->true
   ; peer_data(Peers,MyTextTerm,YsAndWeights)->true
   ; YsAndWeights=[FreqClass-1.0] % default values
)    ,vote_for_best_label(YsAndWeights,YasGuessed).
```

The Voting Algorithm

- the predicate `vote_for_best_label/3` implements our weighted voting mechanism
- after summing up the weight of each label of the same kind, it picks the one with the highest total weight using the list predicate `max_member`

```
vote_for_best_label(YsAndWeights,YasGuessed):-
   keygroups(YsAndWeights,YsAndWeightss),
   maplist(sum_up,YsAndWeightss,WeightAndYs),
   max_member(_-YasGuessed,WeightAndYs).

 keygrups(Ps,KXs):-
   keysort(Ps,Ss),
   group_pairs_by_key(Ss,KXs).

sum_up(Y-Ws,Weight-Y):-sumlist(Ws,Weight).
```

- `vote_for_best_label` can be seen as an approximation of the message-passing information propagation mechanism used in GNNs, giving the peers of a node a say on the node's decision to pick a given label

Ground Term Similarity Measures

- using distilled dependency DAGs represented as Prolog ground terms as input, we compute:
- *shared path similarity*: extracts segments of paths in the trees corresponding to two terms, assuming that the order of the arguments is irrelevant and that distinct functor symbols at the same level can be skipped
- *termlet similarity*: summing up sizes of subterms up to a give size, shared by the two terms
- *Jaccard Similarity*: (size of intersection vs. size of union) on set of constant symbols in the DAGs

Evaluating Accuracy

- we run our label assignment algorithm for several similarity relations
- we do not use the embeddings as our objective is to evaluate our purely symbolic algorithms
- we put them in perspective, by comparing them with the state-of-the-art GNNs

similarity measure used	mock	termlet	shared_path	forest	jaccard_edge	jaccard_node
with graph structure	0.6719	0.6639	0.6780	0.6713	0.6624	**0.6860**
content similarity only	0.0586	0.0815	0.1558	0.1476	0.1282	**0.2825**

Table: Performance impact of graph structure vs. similarity based node content

state of the art GNN performance	OGB GNN	Leaderboard GNN
with structure and content information	0.7213	**0.7431**

Table: GNN performance

Graph Structure, Node Content and the Impact of Diversity

- the results in Table 1 show that when we hide the connectivity information of the graph and use exclusively node content similarity, our best result is still only `0.2825`
- when bringing in 40 rather than the default 4 “content peer” nodes the resulting accuracy of **0.4637** shows that *peer diversity* brings a significant increase of the effectiveness of our node content similarity measures
- when using also the graph structure, we can lift accuracy to **0.6930** with a runtime of `1694.20` seconds, but quickly reaching diminishing returns after that
- $\Rightarrow$ that brings us less than 3 points below the OGB GNN result!

The Limits

- How much we can learn from the Graph's Structure?
  - we compute the number of "guessable nodes" for which it is possible, using exclusively the link structure of the graph, to predict their label
  - we check if, for a given node in the test set, there's at least one neighbor in the training set that has the same label as the one we need to correctly predict
  - $\Rightarrow$ upper limit of **0.8441**
- How much we can Learn from Node Content?
  - the theoretical limit is close to 100% with all our similarity relations, as there are enough content-wise similar nodes, that if found, possibly far away in the graph, would transfer the correct label
  - in practice this is not very useful however, as the large number of matches can easily move the vote towards a closer false positive
  - $\Rightarrow$ the most likely practical limit is somewhere close above the **0.4637** value corresponding to a set of 40 * 40 = 1600 peer nodes with evenly distributed labels

Conclusions

- the declarative simplicity of our code shows the effectiveness of a mature Prolog ecosystem as a competitive exploration tool revealing fine points about the internal logic of deep learning tools
- while we have focussed on a concrete dataset (*ogbn-arxiv*) and a specific task (node property prediction) the general methodology, relying on exploring the analogy between information propagation in a GNNs and the equivalent reasoning steps implemented in Prolog is generalizable to other graph-based datasets, as well as link or graph property prediction tasks
- while we have observed that the graph structure dominates, we have also discovered that if possibly far away peer nodes are used instead of the neighbors, diversity brought by ensuring that all labels are fairly represented can significantly improve results
- $\Rightarrow$ an actionable hint on designing GNNs able to reach far away in the graph's link structure, guided by content-similarity measures