Neural execution of Graph Algorithms

ISPR Course @UniPi, A.A. 23/24

Marco Lavorini

Introduction

Neural Graph Algorithm Execution

- Many solutions to well known **graph problems** have been discovered by theoretical computer science.
- Usually these solutions rely on shared **subroutines**, such as enumerate sets of adjacent edges for a node.
- The goal of this work is to demonstrate how **graph neural networks** are able to transfer knowledge by learning several graph algorithm simultaneously.

Model description

Graph Neural Networks

- **Message-Passing Neural Network** with a **maximisation aggregator** due to the majority of algorithms requiring *discrete decision* over neighbourhoods.
- Encode-process-decode paradigm, for each algorithm A:
  - Encoder-network f_A
  - Processor-network P, shared among all algorithms
  - Decoder-network g_A
- A **GNN layer**, capable of exploiting the edge features as P, is then employed.

Neural Graph Algorithm output

Termination Network

- Given a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges, for each algorithm A:
  - The decoder-network calculates the node and algorithm specific outputs.
  - The processor network P makes the decision whether to terminate the algorithm.
- To address this we need an **Algorithm-specific termination network** T_A that, given the current latent node features and the average node embedding, will provide the probability of termination $\tau^{(t)}$ after applying the sigmoid function σ.

Probability of Termination

- The probability of termination is then defined as:

$$\tau^{(t)} = \sigma(T_A(\mathbf{H}^{(t)}, \overline{\mathbf{H}^{(t)}}))$$

- Where $\mathbf{H^{(t)}} = P(f_A(\vec{x}_i^{(t)}, \vec{h}_i^{(t-1)}), \vec{e}_{ij}^{(t)})$ is the output latent feature, obtained by the processor network taking as input
  - The encoded input $f_A(\vec{x}_i^{(t)}, \vec{h}_i^{(t-1)})$, from the node features and previous latent feature, with $i \in V$
  - The edge features, with $e \in E$ and $i, j \in V$
- And $\overline{\mathbf{H}^{(t)}} = \frac{1}{|V|} \sum_{i \in V} \vec{h}_i^{(t)}$ is the average node embedding
- If $\tau^{(t)} < 0.5$ (i.e. not terminal state) all the computation are repeated

Results

- The experiments were performed on different graph structures and problems, the following table reports the results for both *reachability* and *shortest-path predecessor* problems.
- The 'Model' refers to the GNN layer

Model	**Reachability**	**Predecessor**
LSTM (Hochreiter & Schmidhuber, 1997)	81.97% / 82.29%	47.20% / 47.04%
GAT* (Veličković et al., 2018)	93.28% / 99.86%	64.77% / 60.37%
GAT-full* (Vaswani et al., 2017)	78.40% / 77.86%	67.31% / 63.99%
MPNN-mean (Gilmer et al., 2017)	**100.0% / 100.0%**	93.83% / 93.20%
MPNN-sum (Gilmer et al., 2017)	99.66% / **100.0%**	82.46% / 80.49%
MPNN-max (Gilmer et al., 2017)	**100.0% / 100.0%**	**97.13% / 96.84%**

- The two values are for *mean* and *last-step* accuracy respectively
- More results can be found on the original paper

Conclusions

- The results show that MPNN with a **maximisation aggregator** generalises better in the task of solving **different** graph problems.
- Clear signs of positive knowledge transfer in problems with **similar subroutines**
- The observations still hold independently of the graph structure and size with worsening general performance on bigger graphs
- The **Neural Graph Algorithm Execution** task could be extended to intractable graph problems, or specialised for a specific category of problems, such as interdiction or any flow-related problems.