Assignment 4 (Paper 15)

Di Riccio Tommaso

11/06/2024

Introduction to the problem

Given a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges, the problem is to solve a graph algorithm. This problem is addressed using a GNN.

The authors want to show that, since many classical algorithms share related subroutines, learning several algorithms simultaneously and providing a supervision signal there is an increase of the performance and the generalization ability of the GNN.

In particular, the supervision signal is driven by how a known classical algorithm would process such inputs so that the GNN learn how to imitate individual steps of classical graph algorithms instead of learn a mapping from raw inputs to the ground truth solution.

They called this approach **neural graph algorithm execution**.

Model description

The model follow an *encode-process-decode* architecture.

An *encoder network* f_A is defined **for each algorithm A**. It is applied to the current input features and to the latent features at previous layer to produce an encoded inputs.

$$\vec{z}_i^{(t)} = f_A(\vec{x}_i^{(t)}, \vec{h}_i^{(t-1)})$$

The encoded inputs are then processed using a *processor network P*. The processor network **shares its parameters among all algorithms** being learnt. The processor network takes as input the encoded inputs and the edge features, and produces as output latent node features.

$$\mathbf{H}^{(t)} = P(\mathbf{Z}^{(t)}, \mathbf{E}^{(t)})$$

The node and algorithm specific outputs are then calculated by the *decoder network* g_A, defined **for each algorithm A**.

$$\vec{y}_i^{(t)} = g_A(\vec{z}_i^{(t)}, \vec{h}_i^{(t)})$$

Model description

Another network, called *termination network* T_A, is used to decide on when to terminate the algorithm. It takes in input the latent nodes feature and the mean of them. A sigmoid activation is then applied to the output of T_A to get a probability of termination. The algorithm is terminated if this probability is greater than a chosen threshold. Also the T_A network is defined **for each algorithm A**.

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

If the algorithm hasn't terminated the computation is repeated starting from the encoder network to the decoder network, using parts of the $y_i^{(t)}$ as $x_i^{(t+1)}$.

In the experiment executed in the paper linear projections are used as f_A, g_A, T_A, leaving in the *process network P* most of the representation power.

Key catches of the model

In the best performing version of the model, a GNN is used as *process network P*. In particular a message-passing neural networks is the best performing. The message-passing is implemented with the following formula:

$$\vec{h}_i^{(t)} = U\left(\vec{z}_i^{(t)}, \bigoplus_{(j,i)\in E} M\left(\vec{z}_i^{(t)}, \vec{z}_j^{(t)}, \vec{e}_{ij}^{(t)}\right)\right)$$

where M is a NN that takes the encoded inputs of the node *i* and one of its neighbor *j* (from the encoded network f_A) and the edge feature between the two nodes *(i, j)*. The results for each neighbor *j* [or edge *(i, j)*] are then aggregated using a maximization, summation or averaging operator. To produce the latent node feature, the aggregated result is given in input to another NN U together with the encoded input of the node i.

This allow to have all the information needed to execute all the classical graph algorithms and to let the decoder-network (hopefully) give the correct output for each node.

Key catches of the model

Another important point of the learning is the **supervised signal** provided to the network to let it understand how to execute the algorithm step by step. It is algorithm dependent so the formulas are not reported.

Finally, in the appendix is reported a theoretical insight into why learning to imitate multiple algorithms simultaneously may provide benefits to downstream predictive power

$$
\begin{aligned}
I(Y_A; Y_B|X) &= H(Y_A|X) + H(Y_B|X) - H(Y_A, Y_B|X) \\
&= H(Y_A|X) + H(Y_B|X) - (H(Y_A|Y_B, X) + H(Y_B|X)) \\
&= H(Y_A|X) - H(Y_A|Y_B, X)
\end{aligned}
$$

If we have 2 algorithms A and B that share subroutines, then the conditional mutual information of the corresponding random variables to predict is positive, $I(Y_A; Y_B|X) > 0$. Let H be the Shannon entropy, using the formula we can conclude that $H(Y_A|X) > H(Y_A|Y_B, X)$. This tell us that the information-theoretic uncertainty in y_A is lower if we learn both algorithm simultaneously (i.e. the model can perform better).

Key empirical results

Model	Reachability (mean step accuracy / last-step accuracy) *20 nodes*	*50 nodes*	*100 nodes*
LSTM (Hochreiter & Schmidhuber, 1997)	81.97% / 82.29%	88.35% / 91.49%	68.19% / 63.37%
GAT* (Veličković et al., 2018)	93.28% / 99.86%	93.97% / **100.0%**	92.34% / **99.97%**
GAT-full* (Vaswani et al., 2017)	78.40% / 77.86%	85.76% / 91.83%	88.98% / 91.51%
MPNN-mean (Gilmer et al., 2017)	**100.0% / 100.0%**	61.05% / 57.89%	27.17% / 21.40%
MPNN-sum (Gilmer et al., 2017)	99.66% / **100.0%**	94.25% / **100.0%**	94.72% / 98.63%
MPNN-max (Gilmer et al., 2017)	**100.0% / 100.0%**	**100.0% / 100.0%**	**99.92%** / 99.80%

Model	Predecessor (mean step accuracy / last-step accuracy) *20 nodes*	*50 nodes*	*100 nodes*
LSTM (Hochreiter & Schmidhuber, 1997)	47.20% / 47.04%	36.34% / 35.24%	27.59% / 27.31%
GAT* (Veličković et al., 2018)	64.77% / 60.37%	52.20% / 49.71%	47.23% / 44.90%
GAT-full* (Vaswani et al., 2017)	67.31% / 63.99%	50.54% / 48.51%	43.12% / 41.80%
MPNN-mean (Gilmer et al., 2017)	93.83% / 93.20%	58.60% / 58.02%	44.24% / 43.93%
MPNN-sum (Gilmer et al., 2017)	82.46% / 80.49%	54.78% / 52.06%	37.97% / 37.32%
MPNN-max (Gilmer et al., 2017)	**97.13% / 96.84%**	**94.71% / 93.88%**	**90.91% / 88.79%**
MPNN-max (*curriculum*)	95.88% / 95.54%	91.00% / 88.74%	84.18% / 83.16%
MPNN-max (*no-reach*)	82.40% / 78.29%	78.79% / 77.53%	81.04% / 81.06%
MPNN-max (*no-algo*)	78.97% / 95.56%	83.82% / 85.87%	79.77% / 78.84%

The empirical results show the MPNN with the max aggregation exhibits great generalization abilities in both the breadth-first search (reachability of a node from the source) and the Bellman-Ford (predecessor on the shortest path to a node) algorithms. The mean step accuracy is an important metric given the objective of the authors to imitate the algorithms execution.

The comparison with LSTM show that a GNN-like architecture is necessary to gain in performance using graph-structured inputs.

The comparison with *no-reach* (i.e. learning only Bellman-Ford) show that there is a positive knowledge transfer occurring between the reachability and shortest-path tasks.

Finally, the comparison with *no-algo* (trained directly on the final output) show that there is a benefit using the supervised signal to learn how to imitate the algorithm.

Comment on novelties, strong points and weaknesses

The **main novelties** are the imitation of the individual step of the graph algorithms and the knowledge transfer obtained thanks to the proposed model architecture.

The **strong points** are:

- The model is applicable to both sequential and parallel algorithms
- Strong generalization ability across various graph types and sizes
- The authors proposed a way to provide explainability for the internal decision-making process of the GNN

The **weaknesses** are:

- If trained with small graphs (20 nodes) the model still suffer with large graphs (1500 nodes) despite its good generalization abilities. With big graphs its is also possible to have numerical instability due to the max aggregation operator.
- The supervised signal is algorithm dependent and crafting it appropriately can be complex
- For now, the model can’t deal with negative weight cycles