ISPR 2024 – Assignment 4

Neural Execution of Graph Algorithms

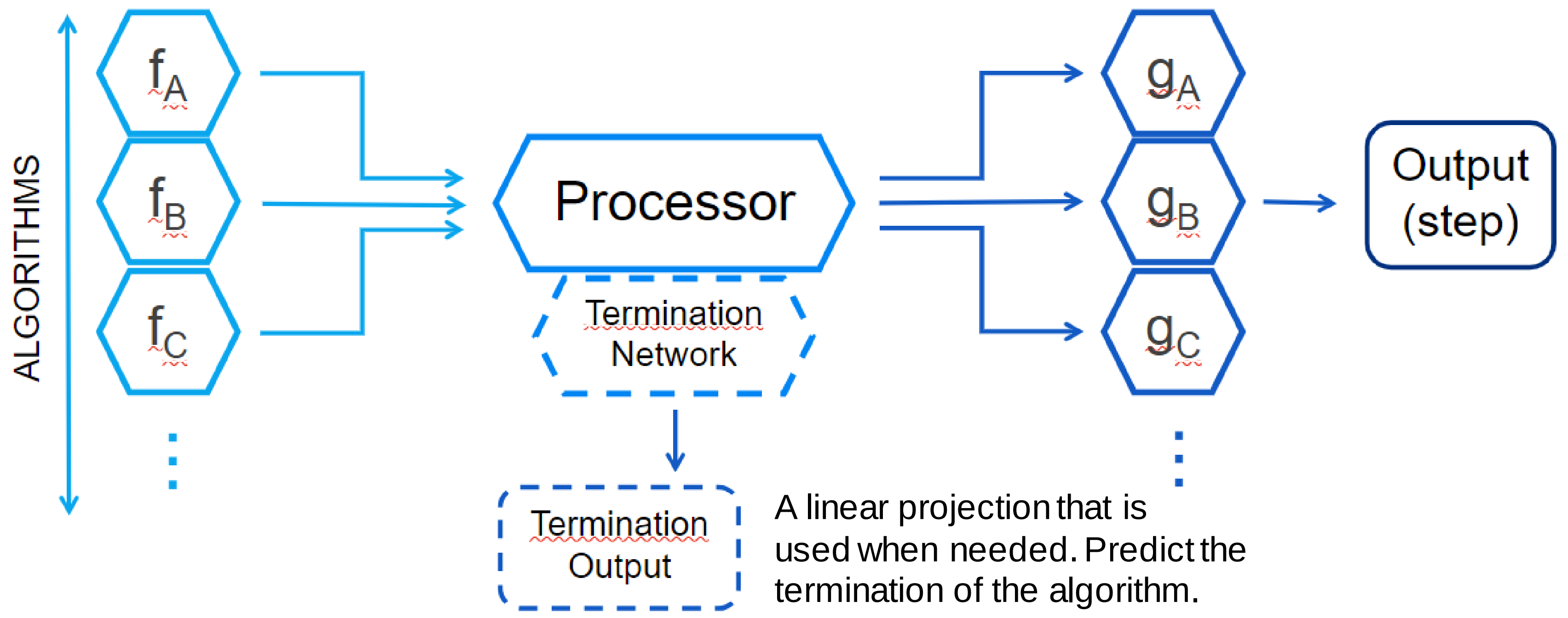
ICLR 2020 - Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, Charles Blundell

Introduction to the Problem

- Many real-world tasks can be formulated as graph problems (e.g., navigation, web search, protein folding), it is therefore important to study the possible applications of neural networks in this field.
- It is possible to apply Graph Neural Networks to solve many of these problems on graphs, training the network to calculate algorithm results from raw input data.
- GNN can also be trained to apply a single step of the algorithm, and in this way arrive at the solution iteratively. Focus on learning in the space of algorithms.
- The authors hypothesise that maximisation-based message passing neural networks are bestsuited for such objectives, and validate this claim empirically.
- The authors chose to implement three well-known algorithms via NN: Breadth-first search, Bellman-Ford algorithm and Prim's algorithm.

Models Description

The main objective of the work is to train a neural network that can generalise with respect to the algorithm to be calculated, and it is trained on multiple algorithms simultaneously. This is made possible by the architecture, which follows the encode-process-decode paradigm.

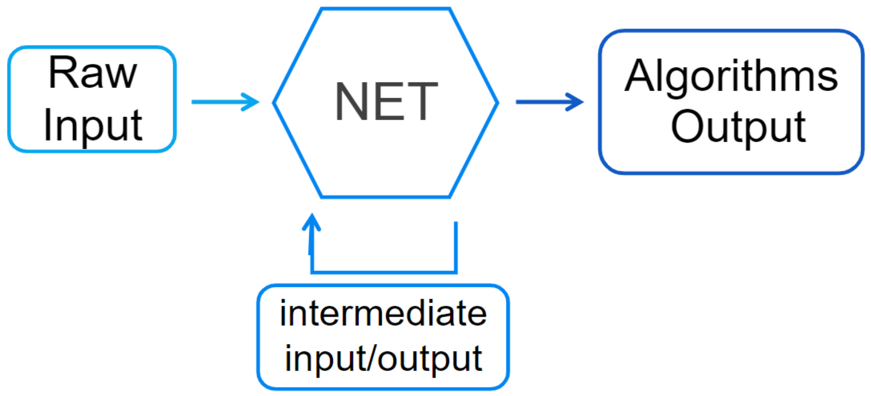


A linear projection that is used when needed. Predict the termination of the algorithm.

For each algorithm learned by the network, it is necessary to train two additional components, encoder and decoder, because only the central processor network is truly shared between all algorithms. Encoders and decoders only represent linear projections of the problem in a new vector space shared by all algorithms.

Models Description

The model calculates only one step of the chosen algorithm (chosen via input). Therefore, many repeated passages through the net is needed to solve one single problem.



The most important module in the whole architecture is the processor network, and the researchers have tested many different sub-networks that fall into two major categories: graph attention networks (GATs) and message-passing neural networks (MPNNs).

Another objective of the research is to study the task of executing algorithms from a 'programmer's perspective': the networks are trained only over a small set of small training examples and tested for their ability to generalise to larger inputs (training on graphs of 20 nodes is tested on graphs of 50 or more).

Key Catch of the Model

GAT

$$\vec{h}_i^{(t)} = \mathrm{ReLU}\left(\sum_{(j,i)\in E} a\left(\vec{z}_i^{(t)}, \vec{z}_j^{(t)}, \vec{e}_{ij}^{(t)}\right) \mathbf{W} \vec{z}_j^{(t)}\right)$$

MPNN

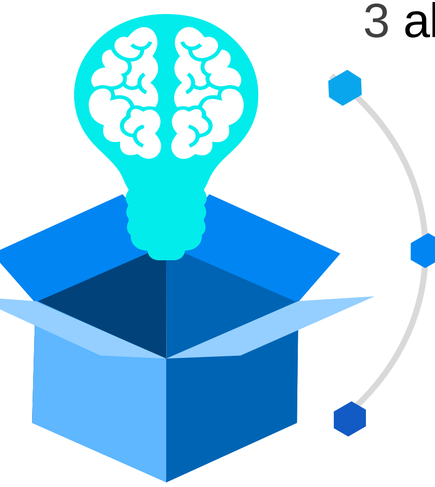
$$\vec{h}_i^{(t)} = U\left(\vec{z}_i^{(t)}, \bigoplus_{(j,i)\in E} M\left(\vec{z}_i^{(t)}, \vec{z}_j^{(t)}, \vec{e}_{ij}^{(t)}\right)\right)$$

W is a learnable projection matrix, a is an attention mechanism producing scalar coefficients, while M and U are neural networks producing vector messages. $\bigoplus$ is an elementwise aggregation operator.

The difference between GAT and MPNN lies in the way they aggregate information from neighbouring nodes. The GAT uses an attention mechanism to assign different weights to links between nodes, derived from the nodes themselves, allowing the network to weight the importance of each neighbouring node differently. In contrast, the MPNN follows a message passing approach, where each node updates its state by combining messages received from its neighbours through an aggregation function (maximum, average or sum are the ones tested in the paper), but without the explicit attention component. This formula captures precisely this difference, and the paper tested the effectiveness of architectures based on both principles.

Key Empirical Result

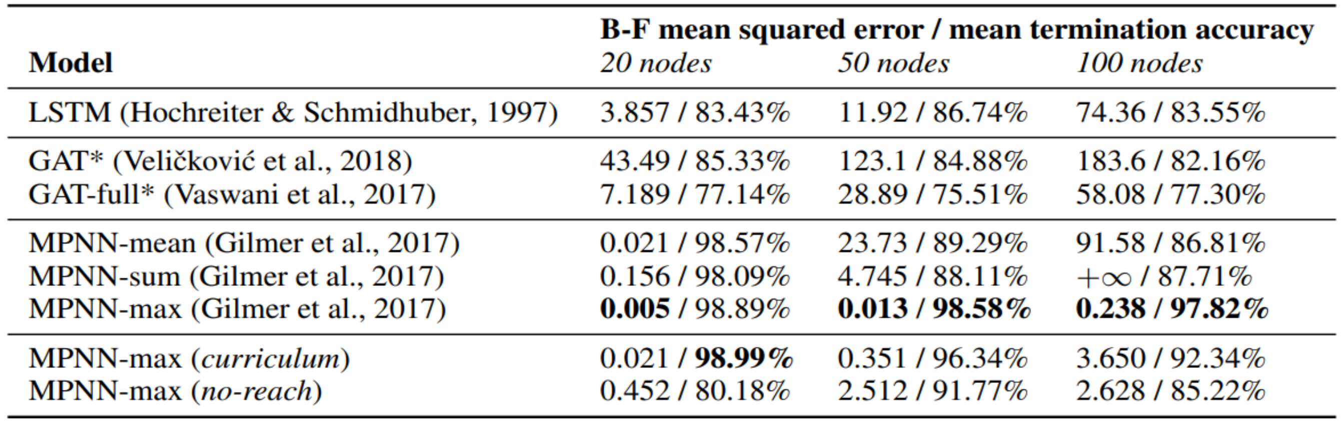
The model successfully learns to execute graph algorithms achieving excellent results in all 3 algorithms tested.



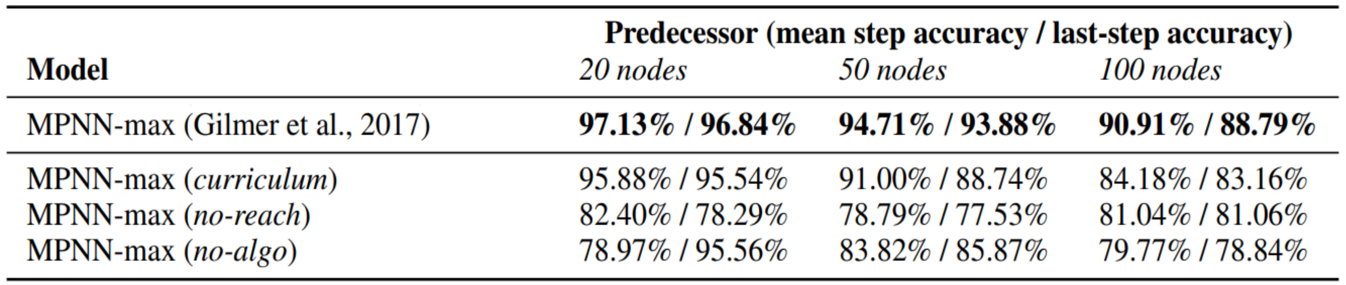
The best model is MPNN-max just as it is logical to think by reasoning how the learned algorithms work.

Performance benefits for learning algorithms simultaneously (e.g., learning shortest paths and reachability together) as shown in (2)

1 Mean squared error for predicting the intermediate distance information from Bellman-Ford, and accuracy of the termination network compared to the ground-truth algorithm



	B-F mean squared error / mean termination accuracy		
Model	*20 nodes*	*50 nodes*	*100 nodes*
LSTM (Hochreiter & Schmidhuber, 1997)	3.857 / 83.43%	11.92 / 86.74%	74.36 / 83.55%
GAT* (Veličković et al., 2018)	43.49 / 85.33%	123.1 / 84.88%	183.6 / 82.16%
GAT-full* (Vaswani et al., 2017)	7.189 / 77.14%	28.89 / 75.51%	58.08 / 77.30%
MPNN-mean (Gilmer et al., 2017)	0.021 / 98.57%	23.73 / 89.29%	91.58 / 86.81%
MPNN-sum (Gilmer et al., 2017)	0.156 / 98.09%	4.745 / 88.11%	$+\infty$ / 87.71%
MPNN-max (Gilmer et al., 2017)	**0.005** / 98.89%	**0.013** / **98.58%**	**0.238** / **97.82%**
MPNN-max (*curriculum*)	0.021 / **98.99%**	0.351 / 96.34%	3.650 / 92.34%
MPNN-max (*no-reach*)	0.452 / 80.18%	2.512 / 91.77%	2.628 / 85.22%



	Predecessor (mean step accuracy / last-step accuracy)		
Model	*20 nodes*	*50 nodes*	*100 nodes*
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%

2 Curriculum corresponds to a curriculum wherein reachability is learnt first. No-reach corresponds to training without the reachability task. No-algo corresponds to the classical setup of directly training on the predecessor, without predicting any intermediate outputs or distances.

Novelties

Neural graph algorithm execution: learning to execute graph algorithms rather than just learning the final solution, parallel algorithms as well as sequential.

Production of a highly suitable architecture based on maximisation-based message passing neural networks (MPNN), that work very well as a solution for this problem. It has been shown that these architectures perform better than the others even under disadvantageous learning conditions, such as reducing the training dataset to only one type of graph.

Strong Points & Weaknesses

Versatility of the model given the absence of assumptions on the algorithms and the diversity of the ones tested (parallel and sequential)

Demonstrated positive transfer between learning tasks. This can allow the learning of complex algorithms to benefit from the learning of simpler related algorithms.

Simpler architecture compared to other neural program synthesis approaches.

The handling of corner cases, such as negative weight cycles, is not addressed. And these cases are often not very common in the datasets and the most complicated, which could lead to learning instability and a considerable increase in task complexity.

Much longer execution time than in the case of direct resolution with NN. It must be empirically tested whether this is a negligible problem or can preserve the intractability of some large examples.