Simulating Finite Automaton using GNNs

Harshit Verma

Abstract:

Recently many developments are being made to make GNNs learn algorithms and then to simulate them.

I attempt to teach GNNs how to simulate deterministic finite automata, basically given any finite automaton, the model would predict if this automata accepts a particular string or not.

The ultimate motive of this research is to study and improve expressibility in GNNs as I hypothesize that an architecture which does better on this problem than another one is in general more expressive than the latter.

Developments in this problem can also be extended to teach GNNs non deterministic finite automaton, which can then solve a NP-Hard problem of acceptance of a string by a ndfa.

Problem Setup:

Currently I have attempted to make models which take any graph as input and the string which acceptance is to be tested is fixed.
So I have taken a random binary string of length 42 as this fixed string.
My alphabet $\Sigma=\{0,1\}$.

This problem has been posed as a regression problem.
The Models output would be a vector of 44 length, with its first 43 values as the node values (Node values range from 1 to n where n is the number of nodes) in the path taken by the string on this dfa and the 44th value would be 100 if the string is accepted and 0 if not accepted.

The motivation behind posing this problem as a regression problem rather than just making a binary classification (Accepted /Not Accepted) was to make the model learn things in accordance with the given input string, rather than just getting random things out of the automata and labeling yes/no.
So by posing the problem in this way the model learns how to traverse the automata according to the string and then label it as yes/no according to the last state reached in the graph.

Dataset:

I have made a custom dataset for this problem.

The dataset consists of 1,20,000 automatas each with number of nodes (n) ranging from 4 to 22.
Each node has 2 edges for the two symbols in the Σ, labeled 0 and 1.
Each graph has exactly one randomly selected starting state.
Number of final states range from 1 to n/3 .
At Least 1 final state is reachable from the starting state.

Node features:
Each node has two features with each feature taking the value as 0 or 1.
0 1 -> starting node
1 0 -> starting node which is also a final node
1 1 -> Final node
0 0 -> normal node

Edge features:
Each edge has two features with each feature taking the value as 0 or 1.
0 1 -> edge with label '0'.
1 0 -> edge with label '1'.

Adjacency matrix (a) where a[i][j]=1 if i and j node values are connected with alphabet '0' and a[i][j]=2 if connected with alphabet '1'.

Output Vector:
The output vector has been described above.
One reason for scaling the last value (which signifies the acceptance) to 100 instead of 1 is that the model incurs more loss if it makes the wrong prediction, since our main objective is to make a model which makes the prediction of acceptance.

Experimental Setup:

First I experimented with different convolution layers to infer which type of convolution will perform the best.

I have constructed an architecture of the following type

Convolution_1 → Convolution_2 → Convolution_3 → Global Sum Pooling → Dense output
I have experimented with different convolution layers, note that I have taken all three convolution layers of the same type.

First I train this type of model for different convolution layers and then I substitute the last layer with a message passing (aggregate=sum) layer to see if it leads to any improvements because I hypothesized that the message passing layer should do good here, and then I do this for every convolution layer.

So let's name the first type of model A where there are 3 same convolution layers, and let's name B the second type of model where this last layer is substituted with a message passing layer.

In the end I also tried the **GeneralGNN**[6] model on this problem.

Number of epochs=10, Optimizer=Adam, Batch_size=32,Loss=Mean Squared Error, train_test split =85%

Results:

Let X be the overall accuracy accounting for both accepted and not accepted and let Y be the accuracy in predicting just accepted.

For Model Type A

ConV layer Type	X in %	Y in %	Loss
Edge-conditioned convolutional layer[1]	74.02	68.13	53.94
Graph Attention layer[2]	66.49	33.4	70.5
Attention-based Graph Neural Network[3]	63.72	27.6	70.46
General convolutional layer[4]	66.77	42.34	62.63
Graph Sage[5]	68.97	35.03	61.57

For Model Type B

ConV layer Type	X in %	Y in %	Loss
Edge-conditioned convolutional layer[1]	71.18	66.43	53.94
Graph Attention layer[2]	67.02	52.46	70.5
Attention-based Graph Neural Network[3]	64.29	28.52	70.46
General convolutional layer[4]	68.2	50.95	62.63
Graph Sage[5]	69.34	40.06	61.57

GeneralGNN Model

X=72.93
Y=45.8
Loss= 59.21

Conclusion:

The experiments clearly state that the best convolution layer for this problem turns out to be Edge Conditioned convolution layer.
The next best is the GeneralGNN model.
In every case we can see that after replacing the last layer as message passing(Model B) X and Y both increase, meaning that message passing layer improves the performance in every case.

Implementation Details:

Code available at https://github.com/harshit5674/Simulating-DFA-using-GNNs

The repository contains
1 notebook to generate dataset.
1 notebook with all the models.
The dataset.

References:

(1) https://arxiv.org/abs/1704.02901
(2) https://arxiv.org/abs/1710.10903
(3) https://arxiv.org/abs/1803.03735
(4) https://arxiv.org/abs/2011.08843
(5) https://arxiv.org/abs/1706.02216
(6) https://arxiv.org/abs/2011.08843