Gated Graph Sequence Neural Networks

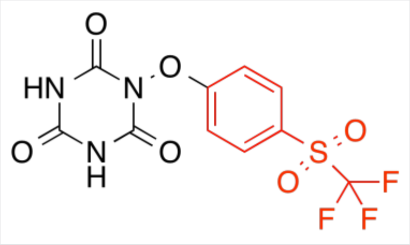
Yujia Li*

Joint work with
Danny Tarlow+, Marc Brockschmidt+ and Rich Zemel*

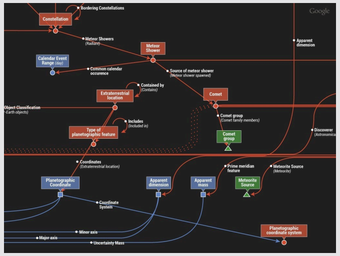
*University of Toronto
+Microsoft Research Cambridge

Many forms of graph-structured data and problems

Molecules*

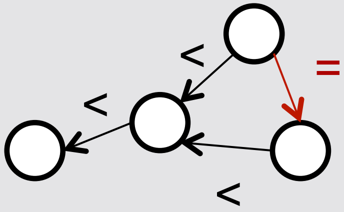


Knowledge Bases

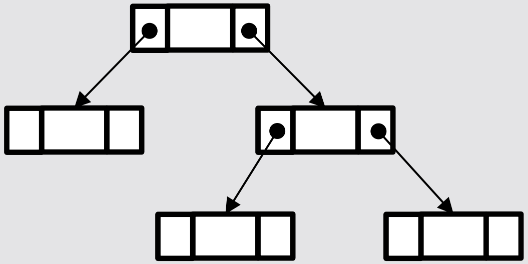


Logical Reasoning

$$\mathrm{smallerthan}(a, b) \wedge \mathrm{smallerthan}(b, c)$$
$$\Rightarrow \mathrm{smallerthan}(a, c)$$



Dynamic Data Structures



More: social networks, graphical models, etc.

*picture from Duvenaud et al. Convolutional Networks on Graphs for Learning Molecular Fingerprints. NIPS'15.

Learning Representations for Graphs

Hand crafted features, graph fingerprints, etc.
[Glem et al., 2006, Brockschmidt et al. 2015]

Graph kernels
[Shervashidze et al., 2011]

Random walks on graphs
[Perozzi et al., 2014]

Graph Neural Networks (next slide)
[Scarselli et al., 2009]

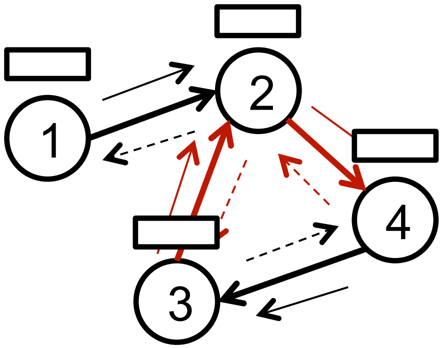
Neural graph fingerprints, conv nets on graphs
[Duvenaud et al., 2015, Bruna et al., 2013]

Graph Neural Networks (GNNs)

A propagation model to compute node representations.

An output model to make predictions on nodes.

Propagation Model



Node representation for node *v* at propagation step *t*: $\mathbf{h}_v^{(t)}$

Propagate representations along edges, allow multiple edge types and propagation on both directions

Edge type and direction

$$\mathbf{h}_v^{(t)} = \sum_{v' \in \mathrm{IN}(v)} f(\mathbf{h}_{v'}^{(t-1)}, l_{(v',v)}) + \sum_{v' \in \mathrm{OUT}(v)} f(\mathbf{h}_{v'}^{(t-1)}, \overset{\downarrow}{l_{(v,v')}})$$

Example: $f(\mathbf{h}_{v'}^{(t-1)}, l_{(v',v)}) = \mathbf{A}^{(l_{(v,v')})}\mathbf{h}_{v'}^{(t-1)} + \mathbf{b}^{(l_{(v,v')})}$

Output Model

$$o_v = g(\mathbf{h}_v^{(T)})$$

For each node v, compute an output based on final node representation. *g* can be a neural net.

Learning as proposed by Scarselli et al.:

Backpropagation through time is expensive.

Restrict the propagation model so that the propagation function is a contraction map → unique fixed point. Run the propagation until convergence.

Training with Almeida-Pineda algorithm [Almeida, 1990; Pineda, 1987].

Gated Graph Neural Networks (GG-NNs)

Unroll recurrence for a fixed number of steps and just use backpropagation through time with modern optimization methods.

Also changed the propagation model a bit to use gating mechanisms like in LSTMs and GRUs.

Benefits:

No restriction on the propagation model, does not need to be a contraction map.

Initialization matters now so problem specific information can be fed in as the input.

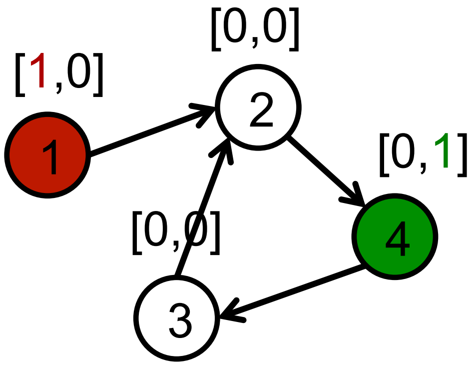
Learning to compute representations within a fixed budget.

Gating makes the propagation model better.

Initialization

Problem specific node annotations in $\mathbf{h}_v^{(0)}$

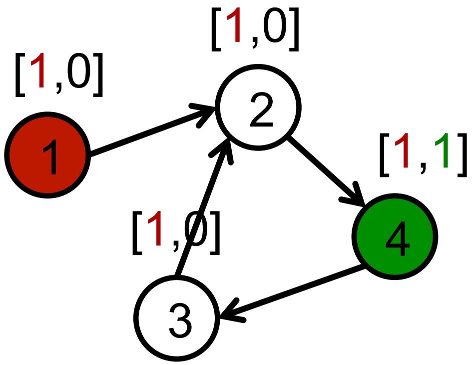
Example reachability problem: can we go from A to B?



Initialization

Problem specific node annotations in $\mathbf{h}_v^{(0)}$

Example reachability problem: can we go from A to B?



output yes

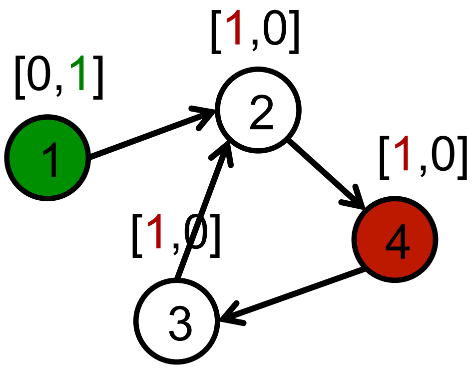
It is easy to learn a propagation model that copies and adds the first bit to a node's neighbor.

It is easy to learn an output model that outputs yes if it sees the [🟥, 🟩] pattern, otherwise no

Initialization

Problem specific node annotations in $\mathbf{h}_v^{(0)}$

Example reachability problem: can we go from A to B?



output no

It is easy to learn a propagation model that copies and adds the first bit to a node’s neighbor.

It is easy to learn an output model that outputs yes if it sees the [🟥, 🟩] pattern, otherwise no

Initialization

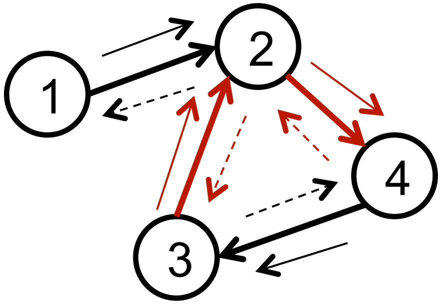
Problem specific node annotations in $\mathbf{h}_v^{(0)}$

In practice we pad node annotations with extra 0’s to add capacity to **h**, so

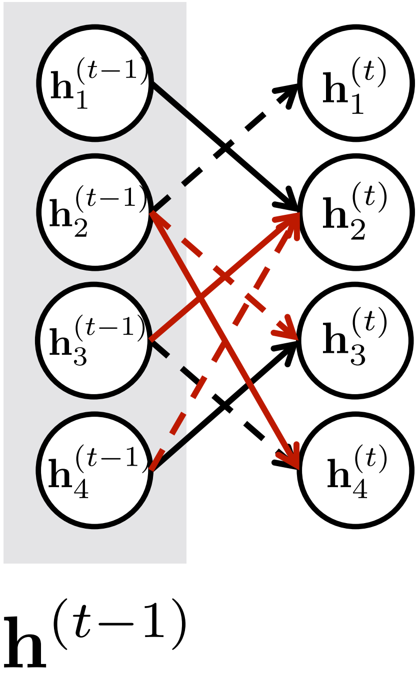
$$\mathbf{h}_v^{(0)} = [\underset{\substack{\uparrow \\ \text{Problem specific node annotations}}}{\boldsymbol{l}_v^\top}, \mathbf{0}^\top]^\top$$

Propagation Model

GNN propagation model with gating and other minor differences

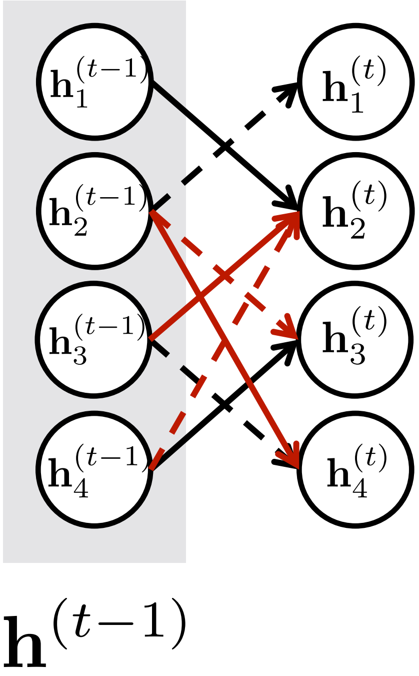


Unroll one step



Propagation Model

GNN propagation model with gating and other minor differences



$$\mathbf{A} = \begin{bmatrix} \mathbf{A}^{(\mathrm{OUT})} \\ \mathbf{A}^{(\mathrm{IN})} \end{bmatrix}$$

$$\mathbf{a}^{(t)} = \begin{bmatrix} \mathbf{a}^{(\mathrm{OUT})} \\ \mathbf{a}^{(\mathrm{IN})} \end{bmatrix} =$$

Outgoing Edges				
	B		C	
				B
		C		
Incoming Edges		B’		
				C’
		C’		
			B'	

x $\mathbf{h}^{(t-1)}$

Propagation Model

GNN propagation model with gating and other minor differences

$$\begin{aligned}\mathbf{a}^{(t)} &= \mathbf{A}\mathbf{h}^{(t-1)} + \mathbf{b} \\ \mathbf{h}_v^{(t)} &= \tanh(\mathbf{W}\mathbf{a}_v^{(t)})\end{aligned}$$

Propagation Model

GNN propagation model with gating and other minor differences

$$\mathbf{a}^{(t)} = \mathbf{A}\mathbf{h}^{(t-1)} + \mathbf{b}$$

Reset gate

$$\mathbf{r}_v^t = \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right)$$

Update gate

$$\mathbf{z}_v^t = \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right)$$

$$\widetilde{\mathbf{h}_v^{(t)}} = \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\right)\right)$$

$$\mathbf{h}_v^{(t)} = (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}$$

Output Models

Per node output same as in GNNs

Node selection output

$o_v = g(\mathbf{h}_v^{(T)}, l_v)$ computes scores for each node, then take softmax over all nodes to select one.

Graph level output

Graph representation vector* $\mathbf{h}_{\mathcal{G}} = \sum_{v \in \mathcal{G}} \sigma(i(\mathbf{h}_v^{(T)}, l_v)) \odot \mathbf{h}_v^{(T)}$

This vector can be used to do graph level classification, regression, etc.

*actual equation is slightly different from this, and more complicated.

The whole network trainable with backprop

Initialization

[0,1,0,0,0]

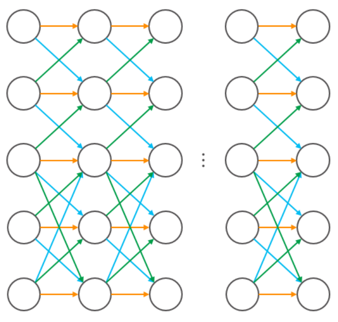
[0,0,0,0,0]

[0,0,0,0,0]

[1,0,0,0,0]

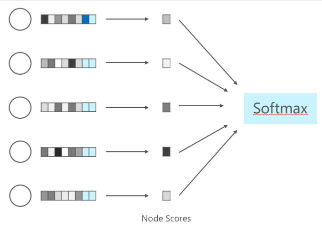
Annotate nodes with problem specific information

Propagation Model



Unroll T steps

Output Model

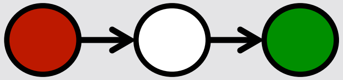


Output

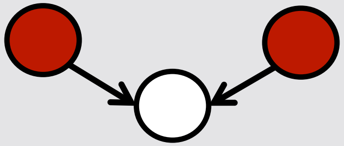
GG-NNs in Action

We first tested GG-NNs on some toy graph property tasks.

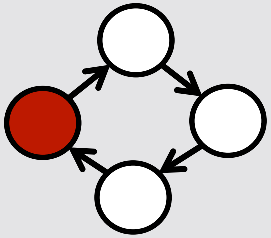
Reachability



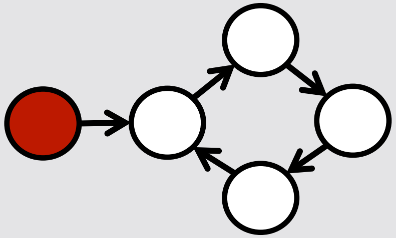
Sharing



Cyclicity



Reaching Cyclicity



More complicated toy tasks.

Some bAbI tasks [Weston et al., 2015]. We used symbolic format of the data, so results not directly comparable with other people's results.

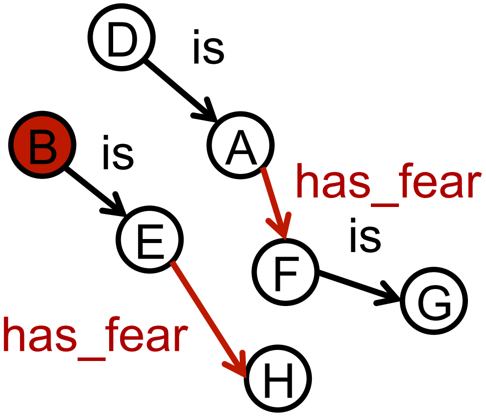
Example: bAbI Task 15 (Basic Deduction)

D is A
B is E
A has_fear F
G is F
E has_fear H
…
eval B has_fear H

Each fact is one edge

Straight forward conversion to graphs

Node-selection output



We tried GG-NNs on bAbI task 4, 15, 16 (all three are node-selection), 18 (graph-level classification) and this model is able to solve all of them to 100% accuracy with only 50 training examples and less than 600 model parameters.

Decided to use RNNs and LSTMs as reference baselines.

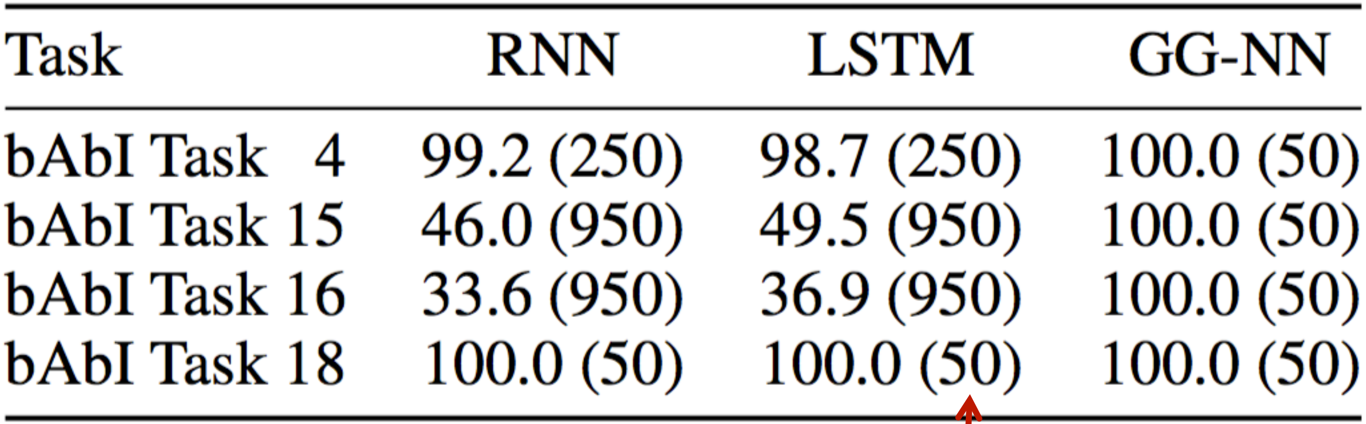
RNN/LSTM trained on token streams

Input:
<D> <is> <A> <\n> <B> …
<eval> <B> <has_fear>
Output: <A>

#parameters: RNN 5k, LSTM 30k

950 training, 50 validation (1000 trainval)
1000 test examples

Start with using only 50 training examples, then keep using more until test accuracy reaches 95% or above.

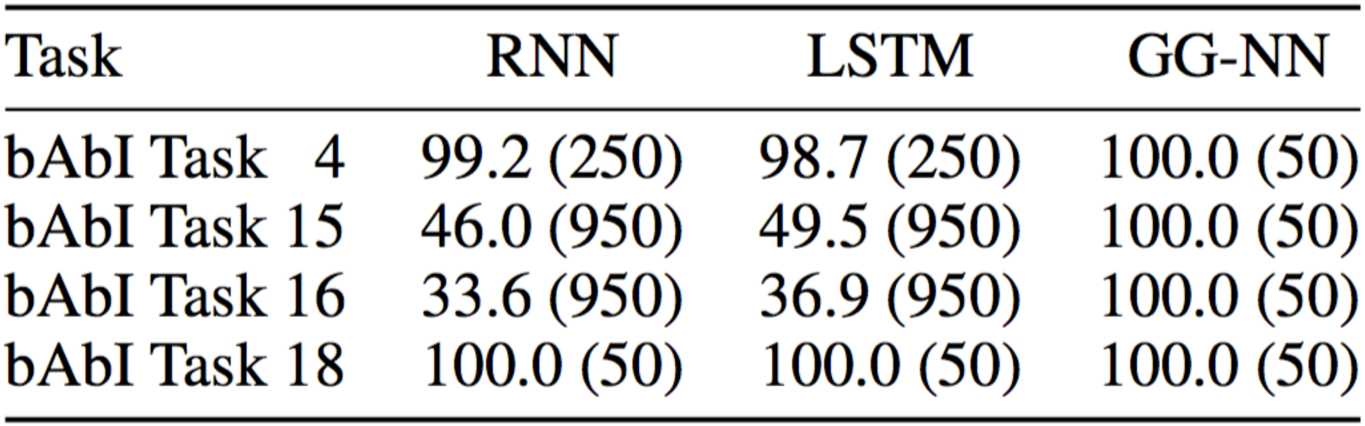


Task	RNN	LSTM	GG-NN
bAbI Task 4	99.2 (250)	98.7 (250)	100.0 (50)
bAbI Task 15	46.0 (950)	49.5 (950)	100.0 (50)
bAbI Task 16	33.6 (950)	36.9 (950)	100.0 (50)
bAbI Task 18	100.0 (50)	100.0 (50)	100.0 (50)

Number of training examples
needed to reach this accuracy

LSTM on Text
(non-symbolic data)
[Weston et al., 2015]

61 (1000)
21 (1000)
23 (1000)
52 (1000)



Task	RNN	LSTM	GG-NN
bAbI Task 4	99.2 (250)	98.7 (250)	100.0 (50)
bAbI Task 15	46.0 (950)	49.5 (950)	100.0 (50)
bAbI Task 16	33.6 (950)	36.9 (950)	100.0 (50)
bAbI Task 18	100.0 (50)	100.0 (50)	100.0 (50)

↑ Not directly comparable

A few conclusions for the results on bAbI tasks:

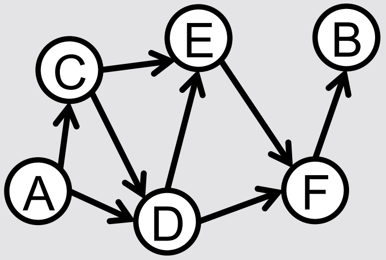
Symbolic format does make the tasks easier but still non-trivial.

We don’t claim GG-NNs can beat RNNs/LSTMs as we used more structures in the problems. But at least this shows that exploiting structures in the problems can make things a lot easier.

Gated Graph Sequence Neural Networks

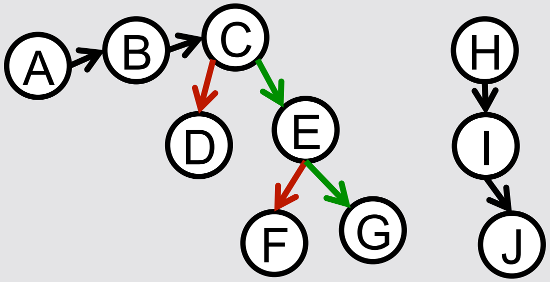
Many problems require a sequence of predictions on graphs.

Shortest path from A to B?



A – D – F – B

What are the structures in this graph?

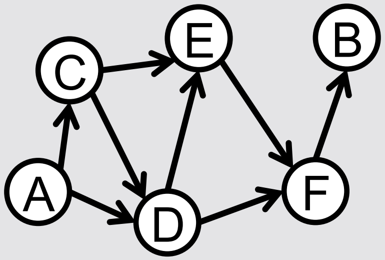


List(A,C) ∧ Tree(C) ∧ List(H, J)

Predictions in each step are made by GG-NNs.

But we need to keep track of where we are in the prediction process.

Shortest path from A to B?



A – D – F – B

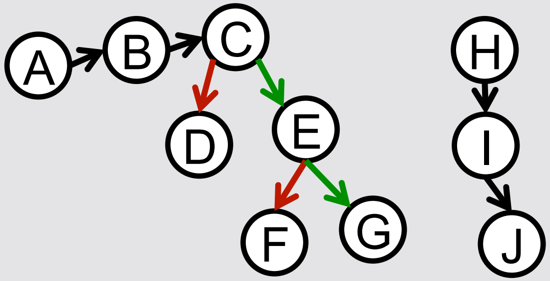
The node annotations used in initialization should be different for different prediction steps

A- (already predicted A),
A-D- (already predicted A-D) and
A-D-F- (already predicted A-D-F)

Predictions in each step are made by GG-NNs.

But we need to keep track of where we are in the prediction process.

What are the structures in this graph?



List(A,C) ∧ Tree(C) ∧ List(H, J)

Need to keep track of which parts have been predicted and which parts have not.

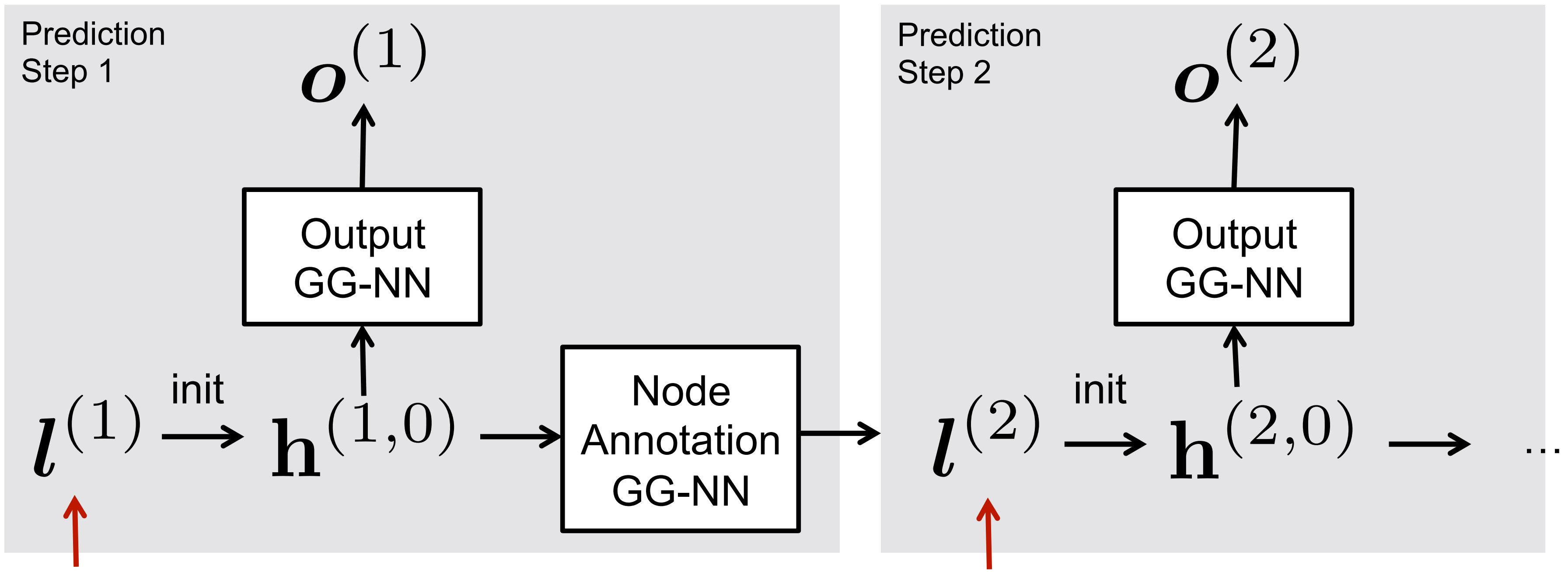
Solution

Chain multiple prediction steps up using node annotations.

Every prediction step produces an output, and produces new node annotations (per-node prediction) for the next step.

GGS-NN architecture

Trainable with backprop from end to end.



Prediction Step 1

$\boldsymbol{o}^{(1)}$

Output GG-NN

$\boldsymbol{l}^{(1)} \xrightarrow{\text{init}} \mathbf{h}^{(1,0)} \longrightarrow$

Node Annotation GG-NN

Prediction Step 2

$\boldsymbol{o}^{(2)}$

Output GG-NN

$\boldsymbol{l}^{(2)} \xrightarrow{\text{init}} \mathbf{h}^{(2,0)} \longrightarrow \dots$

Problem specific node annotations

Whatever the model decides to put there to keep track of the progress

Note: the two GG-NNs can also share a single propagation net, more details in the paper.

GGS-NNs on Simple Tasks

bAbI task 19 (path finding): find the path from one node to another on a graph, guarantee there's only one path.

We created two bAbI-like but more challenging tasks:

Shortest path: find the shortest among possibly multiple paths between two nodes on a graph.

Eulerian circuit: find the Eulerian circuit of a 2-regular connected graph (a graph which is a cycle), a distractor graph is added to make it more challenging.

When to stop

At each prediction step, a separate output GG-NN is used to make a graph-level binary classification prediction on whether to continue or stop.

RNNs/LSTMs keep predicting tokens until an <end> token is hit.

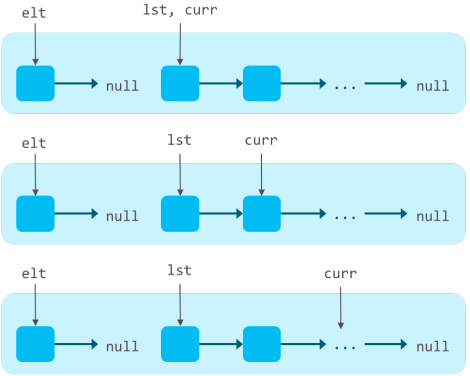
Task	RNN	LSTM	GGS-NNs		
bAbI Task 19	24.5 (950)	29.4 (950)	60.9 (50)	80.3 (100)	99.6 (250)
Shortest Path	11.1 (950)	10.7 (950)	100.0 (50)		
Eulerian Circuit	0.5 (950)	0.4 (950)	100.0 (50)		

GGS-NN for Program Verification

- What is program verification?
  - Verify correctness of a program: given inputs which satisfy some preconditions, is the program guaranteed to produce outputs satisfying some postconditions?
- Need to formally describe what happens during the execution of a program.
  - Analyze heap memory state, which is a graph.
  - Formal descriptions of the heap memory using separation logic formulas.
- Give the separation logic formula to a theorem prover, to verify if it is indeed consistent with the program and if it is strong enough to complete the proof.

The verification pipeline

```
procedure insert(lst: Node, elt: Node)
  returns (res: Node)
{
  if (lst != null)
  {
    var curr := lst;
    while (curr.next != null)
    {
      curr := curr.next;
    }
    elt.next := curr.next;
    curr.next := elt;
    return lst;
  }
}
```



Run the program, get heap memory graph examples

Generate separation logic descriptions, using Machine Learning

$$curr \neq null : elt \mapsto null$$
$$* lseg(lst, curr) * lseg(curr, null)$$

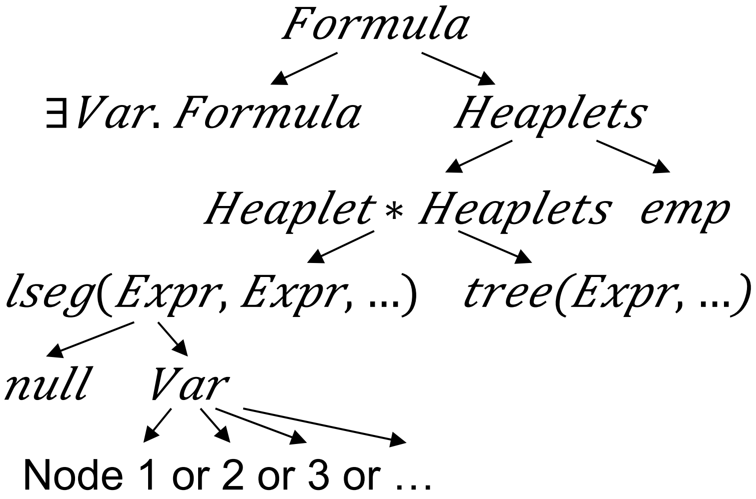
Theorem Prover

This is where the GGS-NN comes in!

From heap graph to separation logic formula

Follow the grammar, every step is either a graph-level classification or a node selection.

$$\begin{aligned}
Formula &\rightarrow \exists Var.Formula \mid Heaplets \\
Heaplets &\rightarrow Heaplet * Heaplets \mid emp \\
Heaplet &\rightarrow lseg(Expr, Expr, (\lambda Var \rightarrow Formula)) \\
&\quad \mid tree(Expr, (\lambda Var \rightarrow Formula)) \\
Expr &\rightarrow null \mid Var
\end{aligned}$$



$Formula$

$\exists Var.\, Formula$ $\quad$ $Heaplets$

$Heaplet * Heaplets$ $\quad$ emp

$lseg(Expr, Expr, \ldots)$ $\quad$ $tree(Expr, \ldots)$

$null$ $\quad$ Var

Node 1 or 2 or 3 or …

Results

We compared the GGS-NN model with an earlier approach [Brockschmidt et al., 2015] using heavily hand-engineered features using domain knowledge combined with standard classifiers.

The data set has 160,000 heap graphs generated from 327 separation logic formulas.

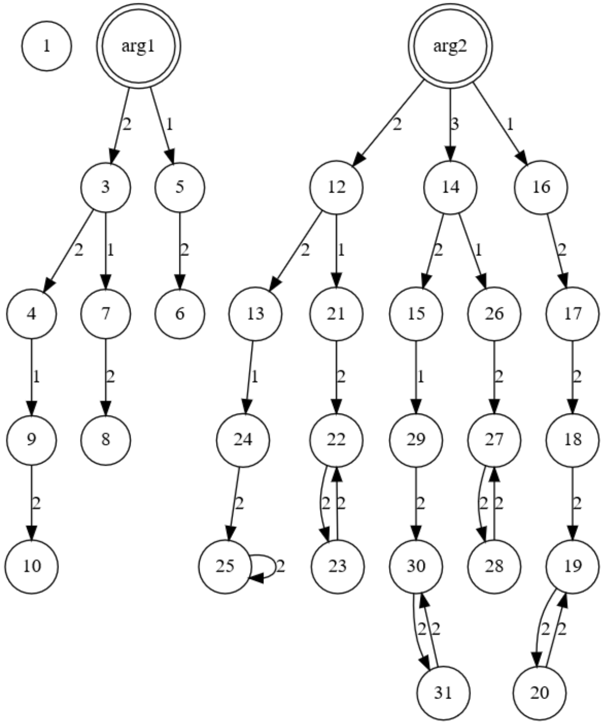
The GGS-NN achieved 89.96% accuracy without any hand engineered features, vs. 89.11% accuracy of the previous approach.

We have also integrated the GGS-NN model into a program verification pipeline. It can successfully verify a test suite of list manipulating programs in a benchmark set.

Program	Separation Logic Formula Found
Traverse1	`ls(lst, curr) * ls(curr, null)`
Traverse2	`ls(lst, curr) * ls(curr, null) * curr != null * lst != null`
Concat	`a != null * a != b * b != curr * curr != null * ls(curr, null) * ls(a, curr) * ls(b, null)`
Copy	`ls(curr, null) * ls(lst, curr) * ls(cp, null)`
Dispose	`ls(lst, null)`
Insert	`curr != null * curr != elt * elt != null * elt != lst * lst != null * ls(elt, null) * ls(lst, curr) * ls(curr, null)`
Remove	`curr != null * lst != null * ls(lst, curr) * ls(curr, null)`

Our GGS-NN model is able to predict more complicated formulas than shown here.

A more complicated example with nested data structures.



$$\mathsf{ls}(\texttt{arg1}, \texttt{NULL}, \lambda t_1 \to \mathsf{ls}(t_1, \texttt{NULL}, \top)) *$$

$$\mathsf{tree}(\texttt{arg2}, \lambda t_2 \to \exists e_1.\mathsf{ls}(t_2, e_1, \top) * \mathsf{ls}(e_1, e_1, \top))$$

Future Directions

Explore the model space, further understand this model

Other applications

Learning to construct the graph

Gated Graph Sequence Neural Networks

Q & A

Yujia Li*

Joint work with
Danny Tarlow+, Marc Brockschmidt+ and Rich Zemel*

*University of Toronto
+Microsoft Research Cambridge