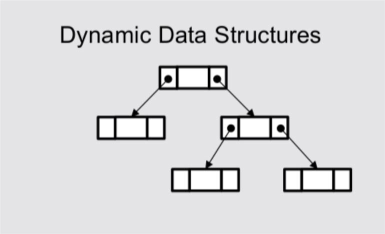
Gated Graph Sequence Neural Networks

A Tutorial

Naganand Y

Motivating Application[1]: Program Verification



- Linked lists, trees are created in heap memory
- Pointers link memory nodes
- Analyse heap memory states to know how the program behaves

[1] Yujia Li et al. (University of Toronto) , Gated Graph Sequence Neural Networks, ICLR 2016

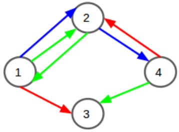
Overview

- Study feature learning techniques for graph-structured inputs
- Demonstrate capabalities on AI (bAbI) tasks
- Show it achieves state-of-the-art performance in program verification

Contribution

- Previous work focused on single outputs
- Main contribution is to output sequences
  - (Shortest) paths on a graph
- Motivating application requires outputting logical formulas

Graph Neural Networks[2]

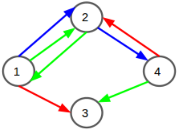


A directed graph $G = (V, E)$ that contains edge types/labels $l_e \in \{1, \cdots, d\}$

- **Propagation Model**: Learn node representations
- **Output Model**: Make predictions on nodes

[2] Scarselli et al. (University of Siena), The Graph Neural Network Model, TNN 2009

Propagation Model



- Node representation for node v at propagation step t is $h_v^{(t)}$
- Propagate representations along edges
- $$h_v^{(t)} = \sum_{u \in IN(v)} f\left(h_u^{(t-1)}, l_{(u,v)}\right) + \sum_{w \in OUT(v)} f\left(h_w^{(t-1)}, l_{(v,w)}\right)$$
  - $f\left(h_v^{(t)}, l_{(u,v)}\right) = A^{\left(l_{(u,v)}\right)} h_v^{(t)} + b^{\left(l_{(u,v)}\right)}$

Output Model

- $o_v = g\left(h_v^{(T)}\right)$
- For each node, compute output based on final node representation
- g can be a neural net

Learning the Parameters

- Propagation model can be unrolled into an RNN
- Backpropagation through time is expensive
  - Need to keep track of all intermediate states
  - Might take many iterations to converge
- Restrict propagation model so that propagation function is a contraction map
  - $||f(h) - f(h')|| \leq \rho||h - h'||$
  - **Banach Fixed Point Theorem**: f has a unique fixed point
  - Guaranteed to converge
  - Train around the fixed point using the Almeida[3]-Pineda[4]algorithm

[3] L.B. Almeida, Learning rule for asynchronous perceptrons with feedback in a combinatorial environment, ICNN 1987

[4] F.J. Pineda, Generalization of back–propagation to recurrent neural networks, Physical Review Letters, 1987

Gated Graph Neural Networks

- **Modified Approach**
  - Unroll recurrence for fixed number of steps
  - Use backpropagation through time
  - Use modern optimizers such as the Adam[5]optimizer
- **Benefits**
  - No restriction to contraction map (more power, capacity)
  - Initialization matters (problem specific node representations)
  - Learning within a fixed budget (better allignment of training and testing)
- Also use gating mechanisms like in LSTUs and GRUs

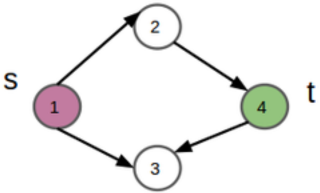
[5]Kingma et. al, Adam: A method for stochastic optimization, ICLR, 2015

Gated Graph Neural Networks

- **Modified Approach**
  - Unroll recurrence for fixed number of steps
  - Use backpropagation through time
  - Use modern optimizers such as the Adam optimizer
- **Benefits**
  - No restriction to contraction map (more power, capacity)
  - **Initialization matters (problem specific node representations)**
  - Learning within a fixed budget (better allignment of training and testing)
- Also use gating mechanisms like in LSTUs and GRUs

Node Annotations: An Example

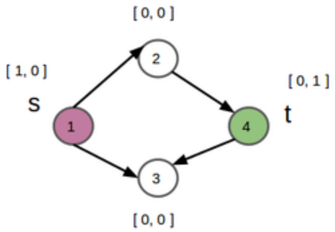
Reachability problem: Can we go from s to t?



- Problem specific node annotations in $h_v^{(0)}$
- **Propagation Model**:
- **Output Model**:

Node Annotations: An Example

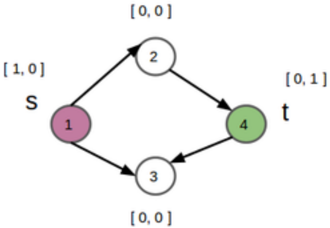
Reachability problem: Can we go from s to t?



- Problem specific node annotations in $h_v^{(0)}$
- **Propagation Model**:
- **Output Model**:

Node Annotations: An Example

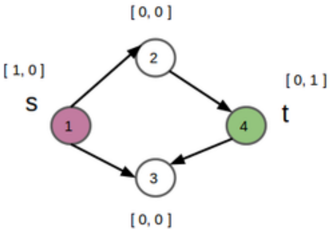
Reachability problem: Can we go from s to t?



- Problem specific node annotations in $h_v^{(0)}$
- **Propagation Model**: learn to copy first bit to every node's neighbor
- **Output Model**:

Node Annotations: An Example

Reachability problem: Can we go from s to t?



- Problem specific node annotations in $h_v^{(0)}$
- **Propagation Model**: learn to copy first bit to every node's neighbor
- **Output Model**: learn to output YES if t has $[1, 1]$ else NO

Propagation Model[6]

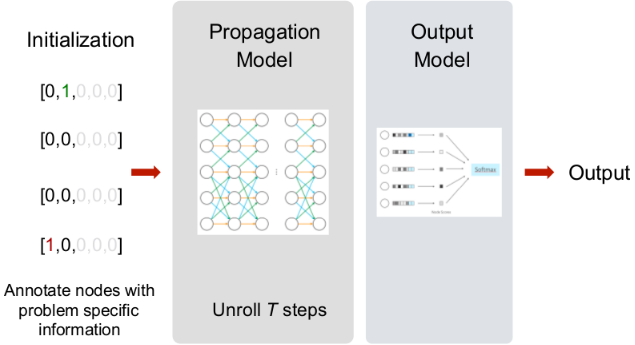
$$
\begin{aligned}
& \mathbf{a}^{(t)} &=& \ \mathbf{A}\mathbf{h}^{(t-1)} + \mathbf{b} \\
\text{Reset gate} \quad & \mathbf{r}_v^t &=& \ \sigma\left(\mathbf{W}^r \mathbf{a}_v^{(t)} + \mathbf{U}^r \mathbf{h}_v^{(t-1)}\right) \\
\text{Update gate} \quad & \mathbf{z}_v^t &=& \ \sigma\left(\mathbf{W}^z \mathbf{a}_v^{(t)} + \mathbf{U}^z \mathbf{h}_v^{(t-1)}\right) \\
& \widetilde{\mathbf{h}_v^{(t)}} &=& \ \tanh\left(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\left(\mathbf{r}_v^t \odot \mathbf{h}_v^{(t-1)}\right)\right) \\
& \mathbf{h}_v^{(t)} &=& \ (1 - \mathbf{z}_v^t) \odot \mathbf{h}_v^{(t-1)} + \mathbf{z}_v^t \odot \widetilde{\mathbf{h}_v^{(t)}}
\end{aligned}
$$

[6] Cho et. al, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation , EMNLP, 2014

Output Models

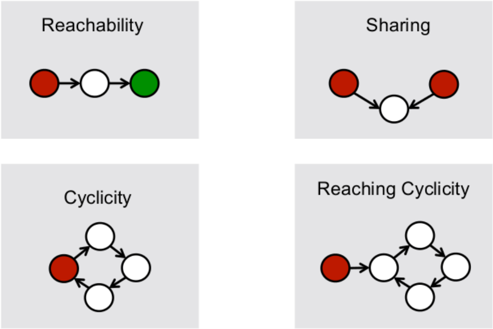
- Per node output: same as in GNNs
- Node selection output
  - Compute score for each node: $o_v = g\left(h_v^{(T)}, l_v\right)$
  - Take softmax over all nodes to select one
- Graph level output
  - Graph representation vector: weighted sum of all node representations
  - Weighting for each node is given by another neural network
  - $$h_G = \sum_{v \in G} \sigma\left(i\left(h_v^{(T)}, l_v\right)\right) \odot h_v^{(T)}$$

The Whole Network



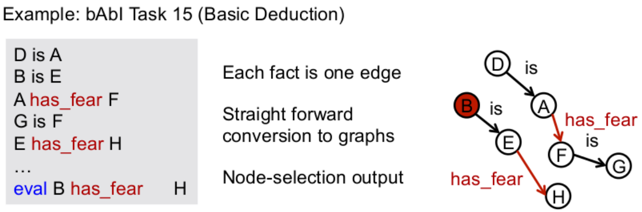
Train with backpropagation

Toy Tasks



- Model is able to learn from a few 10s of examples
- Number of parameters: 100-200

bAbI Tasks [7]



- A natural language reasoning task
- Each instance of input data has a list of facts
- Answer questions through reasoning using the facts

[7]Jason Weston et. al, Towards ai-complete question answering: a set of prerequisite toy tasks, ICLR 2016

bAbI Tasks

- 100% accurate results on bAbI tasks 4, 15, 16 (node-selection) and 18 (graph-level classification)
    - 50 training examples
    - Fewer than 600 model parameters

- Reference baselines: RNNs and LSTMs

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

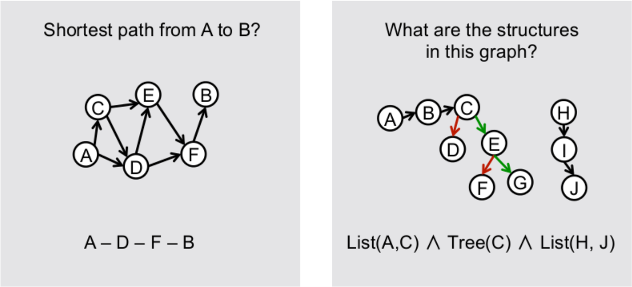
Start with using only 50 training examples, then keep using more until test accuracy reaches 95% or above.

bAbI Task Results

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

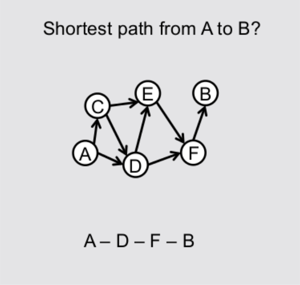
Gated Graph Sequence Neural Networks



Many problems require a sequence of predictions on graphs

Gated Graph Sequence Neural Networks

- Predictions in each step are made by GG-NNs
- Need to keep track of where we are in the prediction process



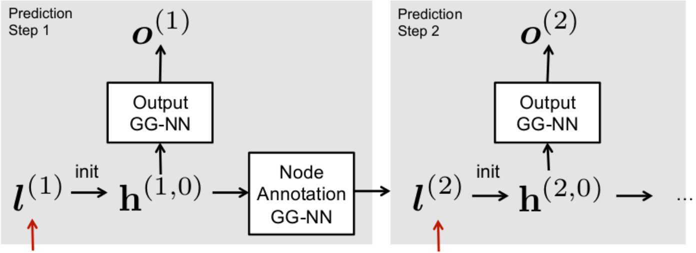
The node annotations used in initialization should be different for different prediction steps

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

Solution

- Chain multiple prediction steps up using node annotations
- **Idea**: Every prediction step produces
  - an output
  - new node annotations (per-node prediction) for the next step

The GGS-NN Architecure



Problem specific node annotations

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

bAbI Task 19

- **Path Finding**: find the path from one node to another on a graph, guarantee there's only one path
- 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

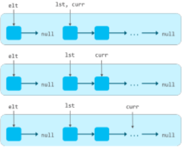
Task	RNN	LSTM	GGS-NNs		
bAbI Task 19	24.5 (950)	29.4 (950)	60.9 (50)	80.3 (100)	99.6 (250)

Program Verification

- Verify correctness of a program
  - Given inputs satisfying preconditions, is the program guaranteed to produce outputs satisfying some postconditions?
- Analyze heap memory state, which is a graph
- Formal descriptions of the heap memory using separation logic formulas

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

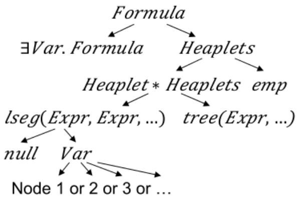
$$curr \neq null : elt \mapsto null$$
$$*lseg(lst, curr) * lseg(curr, null)$$

Theorem Prover

This is where the GGS-NN comes in!

From Heap Graph to Separation Logic Formula

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



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

Results

- Compared the GGS-NN model with an earlier approach [8]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

[8]Brockschmidt et. al, Learning to decipher the heap for program verification, CMLICML, 2015