LEARNING GRAPHICAL STATE TRANSITIONS

Daniel D. Johnson

25 April 2017

Harvey Mudd College

MOTIVATION

GRAPH-STRUCTURED DATA

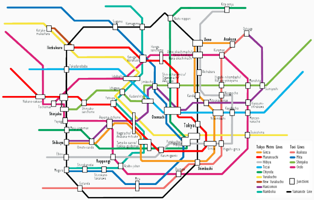
Many interesting forms of data consist of *relationships* between *entities*.

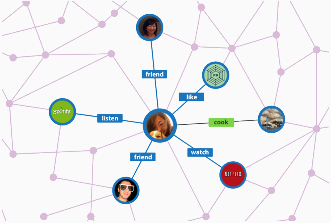
GRAPH-STRUCTURED DATA

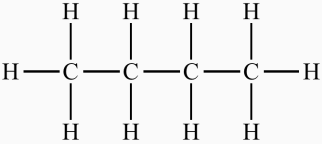
Many interesting forms of data consist of *relationships* between *entities*.

This naturally maps to a graphical representation, with *edges* between *nodes*.

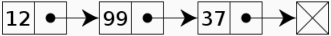
GRAPH-STRUCTURED DATA: EXAMPLES







$$\begin{array}{ccccccccccc} & & \mathrm{H} & & \mathrm{H} & & \mathrm{H} & & \mathrm{H} & & \\ & & | & & | & & | & & | & & \\ \mathrm{H} & - & \mathrm{C} & - & \mathrm{C} & - & \mathrm{C} & - & \mathrm{C} & - & \mathrm{H} \\ & & | & & | & & | & & | & & \\ & & \mathrm{H} & & \mathrm{H} & & \mathrm{H} & & \mathrm{H} & & \end{array}$$



PREVIOUS WORK

- ***Graph Neural Network*** **(Gori et al., 2005; Scarselli et al., 2009)**

PREVIOUS WORK

- *Graph Neural Network* (Gori et al., 2005; Scarselli et al., 2009)
    - Each node has a state vector, representing its neighborhood

PREVIOUS WORK

- *Graph Neural Network* (Gori et al., 2005; Scarselli et al., 2009)
    - Each node has a state vector, representing its neighborhood
    - Updates states of nodes based on states of adjacent nodes, until convergence

PREVIOUS WORK

- *Graph Neural Network* (Gori et al., 2005; Scarselli et al., 2009)
    - Each node has a state vector, representing its neighborhood
    - Updates states of nodes based on states of adjacent nodes, until convergence
    - Use states to produce output

PREVIOUS WORK

- ***Graph Neural Network*** **(Gori et al., 2005; Scarselli et al., 2009)**
    - **Each node has a state vector, representing its neighborhood**
    - **Updates states of nodes based on states of adjacent nodes, until convergence**
    - **Use states to produce output**
- ***Gated Graph Neural Network*** **(Li et al., 2016)**

PREVIOUS WORK

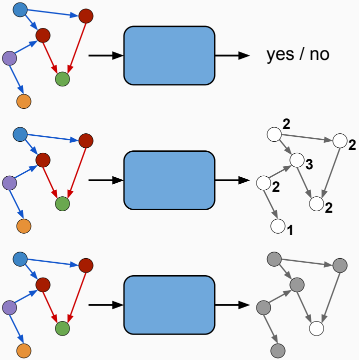
- ***Graph Neural Network*** **(Gori et al., 2005; Scarselli et al., 2009)**
  - **Each node has a state vector, representing its neighborhood**
  - **Updates states of nodes based on states of adjacent nodes, until convergence**
  - **Use states to produce output**
- ***Gated Graph Neural Network*** **(Li et al., 2016)**
  - **Like GNN, but compute states using fixed number of GRU-style updates, train with backpropagation**

PREVIOUS WORK

- *Graph Neural Network* (Gori et al., 2005; Scarselli et al., 2009)
  - Each node has a state vector, representing its neighborhood
  - Updates states of nodes based on states of adjacent nodes, until convergence
  - Use states to produce output
- *Gated Graph Neural Network* (Li et al., 2016)
  - Like GNN, but compute states using fixed number of GRU-style updates, train with backpropagation
  - Gated Graph Sequence Neural Networks: extension to produce output sequences

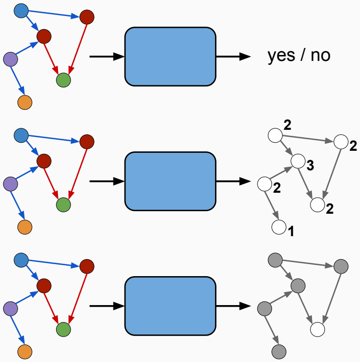
TASK OVERVIEW

Previous work



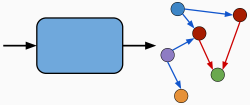
TASK OVERVIEW

Previous work

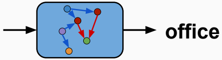


Current work

The bedroom is south of the office. The kitchen is west of the bedroom. ...



Mary went to the garden. John journeyed to the office. ... Where is John?



GOALS

- **Design a neural network architecture that can manipulate graphical states**
- **Use this architecture to solve tasks with graphical internal state and/or graphical output**

GOALS

- Design a neural network architecture that can manipulate graphical states
- Use this architecture to solve tasks with graphical internal state and/or graphical output

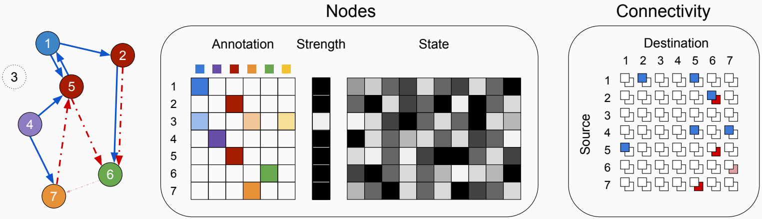
Why?

- Using graphs as an internal representation is natural to some tasks, and can help interpret network behavior
- This provides a general framework for learning to output structured data

MODEL

GRAPH REPRESENTATION

- Set of nodes $v \in \mathcal{V}$, each with:
  - a strength s_v (with $0 \leq s_v \leq 1$)
  - an annotation $\mathbf{x}_v \in \mathbb{R}^N$ where $\sum_{j=1}^{N} x_{v,j} = 1$
  - a hidden state $\mathbf{h}_v \in \mathbb{R}^D$
- Connectivity matrix $\mathcal{C} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}| \times Y}$
  - $\mathcal{C}_{v,v',y}$: strength of directed edge of type y from v to v' (with $0 \leq \mathcal{C}_{v,v',y} \leq 1$)

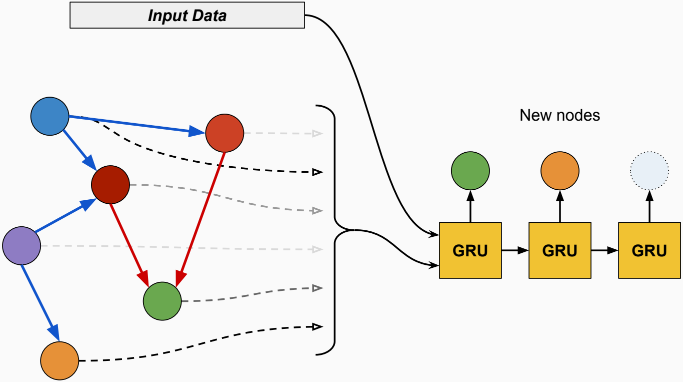


GRAPH TRANSFORMATION TYPES

- **Node addition:**
  create new nodes
- **Node state update:**
  update node states based on new input
- **Edge update:**
  add or remove edges based on node states
- **Propagation:**
  update node states based on adjacent node states
- **Aggregation:**
  combine node states into a representation vector

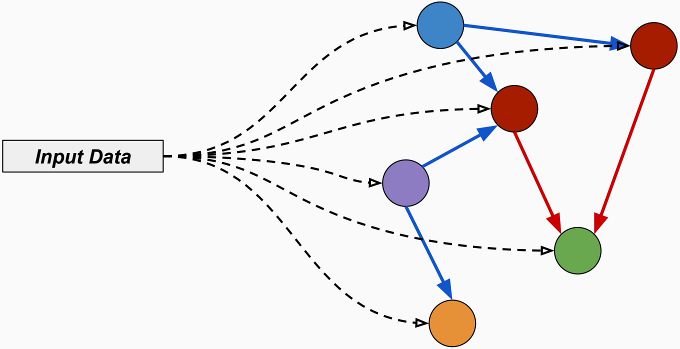
NODE ADDITION

Create new nodes conditioned on an input vector



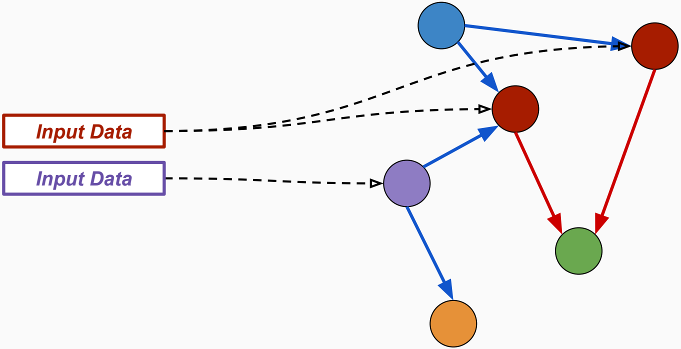
NODE STATE UPDATE

Update node states conditioned on an input vector.



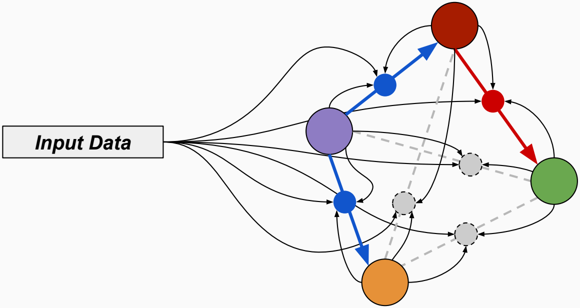
NODE STATE UPDATE: DIRECT REFERENCE

If there are different input vectors for each node type, update each node type separately.



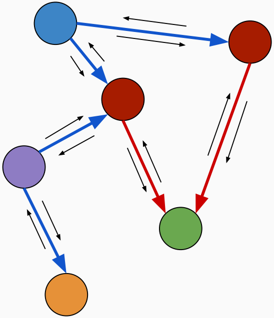
EDGE UPDATE

Add or remove edges conditioned on node states and an input vector.



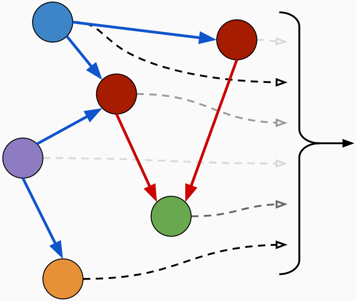
PROPAGATION

Exchange information between nodes along edges based on the node states and the edge types.

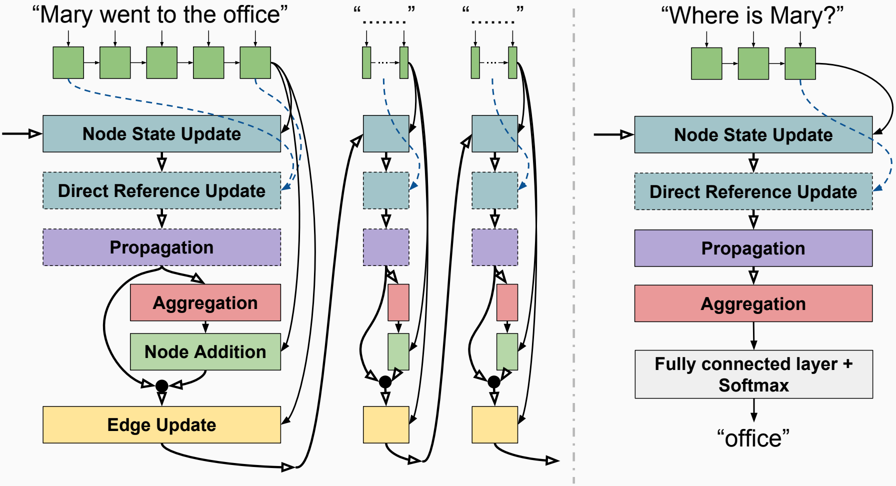


AGGREGATION

Compute a graph-level representation vector as a weighted sum of outputs from each node.



GATED GRAPH TRANSFORMER NEURAL NETWORK (GGT-NN)



TRAINING THE GGT-NN

- **Provide correct graph state after each input sentence**

TRAINING THE GGT-NN

- Provide correct graph state after each input sentence
- Train GGT-NN to
    - reproduce these graph states
    - answer query correctly using final graph state

TRAINING THE GGT-NN

- Provide correct graph state after each input sentence
- Train GGT-NN to
    - reproduce these graph states
    - answer query correctly using final graph state
- During training: substitute correct nodes and edges after each sentence

TRAINING THE GGT-NN

- Provide correct graph state after each input sentence
- Train GGT-NN to
    - reproduce these graph states
    - answer query correctly using final graph state
- During training: substitute correct nodes and edges after each sentence
- After training: use unmodified network output

EXPERIMENTS

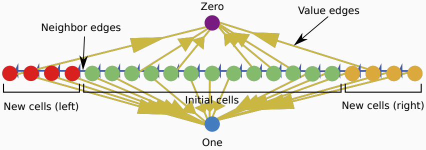
BABI TASKS

Dataset of 20 simple synthetic question-answering tasks

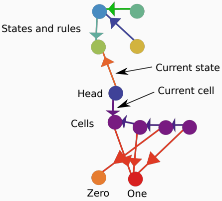
(Weston et al., 2016)

- **95% accuracy in 19 tasks using 1000 examples**
- **100% accuracy in 11 tasks using 1000 examples**
    - **Including "Basic Induction" and "Pathfinding" tasks**
- **95% accuracy in 14 tasks using 500 examples**
- **95% accuracy in 10 tasks using 250 examples**

Rule 30 Cellular Automaton (Wolfram, 2002)



Arbitrary 2-symbol 4-state Turing machine



RULE DISCOVERY: RESULTS

	Accuracy		
	Original Task	**Generalization: 20**	**Generalization: 30**
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

Automaton output at step 1:

1000 iterations

2000 iterations

3000 iterations

7000 iterations

RULE DISCOVERY: RESULTS

	Accuracy		
	Original Task	**Generalization: 20**	**Generalization: 30**
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

Automaton output at step 2:

1000 iterations

2000 iterations

3000 iterations

7000 iterations

RULE DISCOVERY: RESULTS

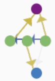
	Accuracy		
	Original Task	**Generalization: 20**	**Generalization: 30**
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

Automaton output at step 3:

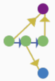
1000 iterations

2000 iterations

3000 iterations



7000 iterations



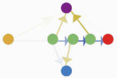
RULE DISCOVERY: RESULTS

	Accuracy		
	Original Task	Generalization: 20	Generalization: 30
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

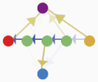
Automaton output at step 4:

1000 iterations

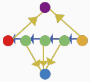
2000 iterations



3000 iterations



7000 iterations



RULE DISCOVERY: RESULTS

	Accuracy		
	Original Task	**Generalization: 20**	**Generalization: 30**
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

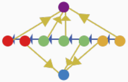
Automaton output at step 5:

1000 iterations

2000 iterations

3000 iterations

7000 iterations



RULE DISCOVERY: RESULTS

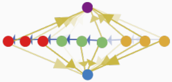
		Accuracy	
	Original Task	**Generalization: 20**	**Generalization: 30**
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

Automaton output at step 6:

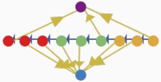
1000 iterations

2000 iterations

3000 iterations



7000 iterations



RULE DISCOVERY: RESULTS

	Accuracy		
	Original Task	Generalization: 20	Generalization: 30
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

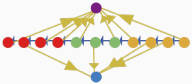
Automaton output at step 7:

1000 iterations

2000 iterations

3000 iterations

7000 iterations



RULE DISCOVERY: RESULTS

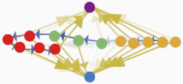
	Accuracy		
	Original Task	**Generalization: 20**	**Generalization: 30**
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

Automaton output at step 8:

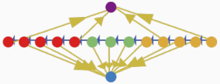
1000 iterations

2000 iterations

3000 iterations



7000 iterations



RULE DISCOVERY: RESULTS

	Accuracy		
	Original Task	**Generalization: 20**	**Generalization: 30**
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

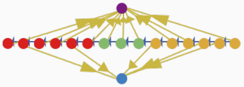
Automaton output at step 9:

1000 iterations

2000 iterations

3000 iterations

7000 iterations



RULE DISCOVERY: RESULTS

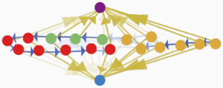
	Accuracy		
	Original Task	Generalization: 20	Generalization: 30
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

Automaton output at step 10:

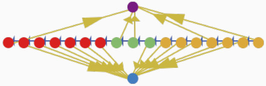
1000 iterations

2000 iterations

3000 iterations



7000 iterations



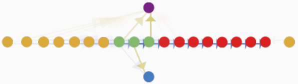
RULE DISCOVERY: RESULTS

	Accuracy		
	Original Task	Generalization: 20	Generalization: 30
Automaton	100.0%	87.0%	69.5%
Turing	99.9%	90.4%	80.4%

Automaton output at step 11:

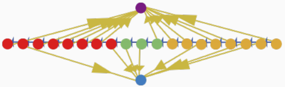
1000 iterations

2000 iterations



3000 iterations

7000 iterations



FUTURE WORK

Future work will focus on

- Reducing model supervision
- Sparse connectivity optimizations
- Extending node types

CONCLUSIONS

- GGT-NN model can construct and manipulate graphical state
- Modular graph transformations can be recombined in different ways
- GGT-NN successfully solves textual bAbI tasks and graphical rule discovery tasks, and is potentially useful for a wide variety of structured data applications

THANK YOU!