Gated Graph Sequence Neural Networks

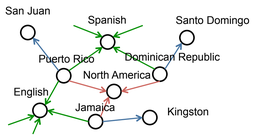
Yujia Li, Daniel Tarlow, Marc Brockschmidt, Richard Zemel

Microsoft Research

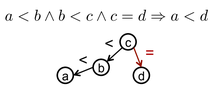
UNIVERSITY OF TORONTO

Introduction

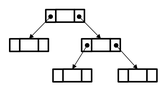
There are many graph-structured data and problems.



Knowledge Graphs



Logical Reasoning



Dynamic Data Structures

and many more…

Sample problems: entity relation reasoning, link prediction, graph property prediction, shortest-paths, etc.

In this paper we study learning representations for nodes and graphs and propose

Gated Graph Neural Networks for making single predictions on graphs.

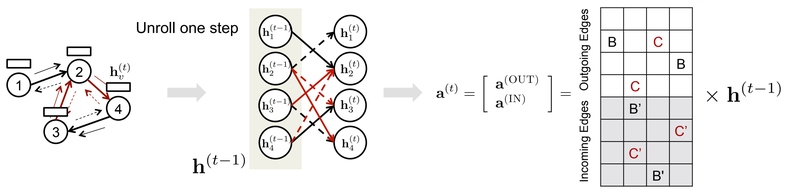
Gated Graph Sequence Neural Networks for making sequences of predictions on graphs.

Gated Graph Neural Networks

We based our model on the Graph Neural Networks [1], but made a few important changes.

Propagation model computes node representations.

Allow multiple edge types (indicated by different colors in the illustrations). Propagation in both directions. Each node v has representation $\mathbf{h}_v^{(t)}$ at propagation step t.



RNN-Style Update

GRU-Style Update

Propagation

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

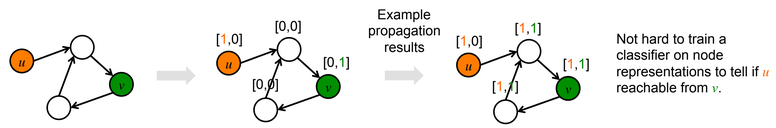
$$\mathbf{h}_v^{(t)} = \tanh(\mathbf{W}\mathbf{a}_v^{(t)} + \mathbf{U}\mathbf{h}_v^{(t-1)})$$

Nonlinear Update

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

Problem-specific node annotations are used to initialize $\mathbf{h}^{(0)}$.

Example: can we reach u from v? For this problem, we need to let the model know that u and v are special and different. $\mathbf{h}_u$=[1,0], $\mathbf{h}_v$=[0,1], others [0,0].



In practice we set $\mathbf{h}_v^{(0)} = \begin{bmatrix} l_v \\ \mathbf{0} \end{bmatrix}$ (pad with zeros) to give the model extra capacity.

node annotations

Output Model makes predictions from node representations.

Per-Node Output

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

One output for each node.

Node Selection Output

$$o = \text{Softmax}_v\{g(\mathbf{h}_v^{(T)}, l_v)\}$$

One score for each node, then select a node based on score.

Graph Level Output

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

$$o = g(\mathbf{h}_\mathcal{G})$$

Compute a graph representation vector, then do standard tasks on it.

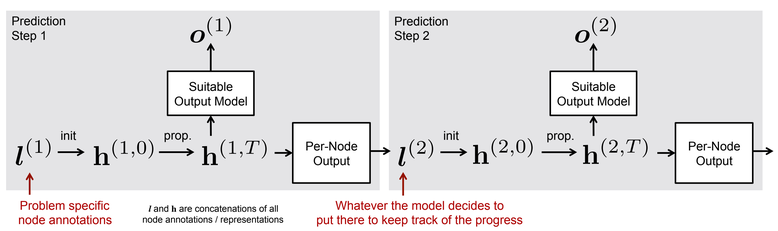
Training: unroll propagation process for fixed *T* steps and backprop through time, trained end-to-end.

[1] restricted the propagation model to contraction map and used the Almeida-Pineda [2,3] algorithm for training. We lifted this restriction and made node initialization meaningful as the propagation model does not need to be a contraction map. We proved that under the contraction map restriction the model has problems modeling long-range interactions.

Gated Graph Sequence Neural Networks

In some cases we need to make a sequence of decisions or generate a a sequence of outputs for a graph. To solve these problems on graphs: each prediction step can be implemented with a GG-NN, from step to step it is important to keep track of the processed information and states.

Solution: after each prediction step, produce a per-node state vector to be carried over to the next step.



The whole sequence model trainable is end-to-end.

When to stop: use a special <end> output or have an extra graph level output model to decide whether to continue or not after each step.

bAbI tasks and graph algorithms

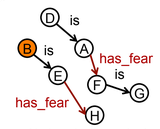
We tested the GG-NN model on 4 toy graph property tasks, all of them are solved perfectly with only a few tens of training examples.

We also evaluated on bAbI [3] tasks 4, 15, 16, 18, 19 and created two extra bAbI-like sequence prediction tasks on graphs.

We used the symbolic format of the bAbI data to get rid of the natural language parsing, and exclusively focus on reasoning. A graph is easily constructed for each story, predictions are made after reading the graph.

Example task (bAbI task 15):

```
D is A
B is E
A has_fear F
G is F
E has_fear H
...
eval B has_fear    H
```



Node selection output

Task	RNN	LSTM	GG-NN
bAbI Task 4	97.3±1.9 (250)	97.4±2.0 (250)	100.0±0.0 (50)
bAbI Task 15	48.6±1.9 (950)	50.3±1.3 (950)	100.0±0.0 (50)
bAbI Task 16	33.0±1.9 (950)	37.5±0.9 (950)	100.0±0.0 (50)
bAbI Task 18	88.9±0.9 (950)	88.9±0.8 (950)	100.0±0.0 (50)

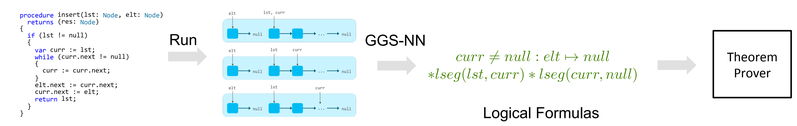
Single prediction problems. GG-NN solved all of them with only 50 training examples.

Task	RNN	LSTM	GGS-NN		
bAbI Task 19 (path finding)	24.7±2.7 (950)	28.2±1.3 (950)	71.1±14.7 (50)	92.5±5.9 (100)	99.0±1.1 (250)
Shortest Path	9.7±1.7 (950)	10.5±1.2 (950)	100.0± 0.0 (50)		
Eulerian Circuit	0.3±0.2 (950)	0.1±0.2 (950)	100.0± 0.0 (50)		

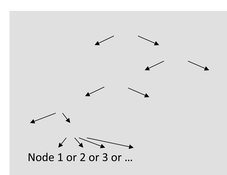
Sequence prediction problems. GGS-NN solved all of them with only very little training examples compared to RNNs and LSTMs that do not make use of the graph structure.

Program Verification Application

This work on GGS-NN is motivated by the program verification application, where we need to analyze dynamic data structures created in the heap. On a very high level, in this application a machine learning model analyzes the heap states (a graph with memory nodes and pointers as edges) during the execution of a program and comes up with logical formulas that describes the heap. These logical formulas are then fed into a theorem prover to prove the correctness of the program.



Follow the logical formula grammar. Producing one part at a time given the heap graphs. It is therefore a sequence prediction problem on heap graphs.



Compared to an earlier approach based on classifiers trained on hand-engineered features designed by domain experts, the GGS-NN approach automatically learns useful features and does as well as the hand-engineered approach. See paper for more details.

References

[1] Scarselli et al.. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009

[2] Pineda, J. Generalization of back-propagation to recurrent neural networks. *Physical review letters*, 59(19):2229, 1987

[3] Almeida, B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial neural networks*, pp. 102–111. IEEE Press, 1990.

[4] Weston et al.. Towards AI-complete question answering: a set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.