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

Example: can we reach $a$ from $v$? For this problem, we need to let the model know that $a$ and $v$ are special and different. $h_a = [1,0,0]$, $h_v = [0,1,0]$, others $[0,0,0]$.

In practice we set $h^{(0)} = [\text{node annotations}]$ (pad with zeros) to give the model extra capacity.

Output Model makes predictions from node representations.

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

Example: can we reach $a$ from $v$? For this problem, we need to let the model know that $a$ and $v$ are special and different. $h_a = [1,0,0]$, $h_v = [0,1,0]$, others $[0,0,0]$.

In practice we set $h^{(0)} = [\text{node annotations}]$ (pad with zeros) to give the model extra capacity.

Output Model makes predictions from node representations.

Per-Node Output
$$o_v = g(h^{(T)}_v, l_v)$$

One output for each node.

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

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

Graph Level Output
$$h_{G} = \sum_v o_v (h^{(T)}_v) \otimes h^{(T)}_v$$

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 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 $h_v^{(i)}$ at propagation step $i$.

Gated Gated Graph Neural Networks for making single predictions on graphs.

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

Node annotations / representations

Per-Node

Node Selection

Graph Level

Output Model

Step 1

Prediction

Step 2

Suitable

Node-specific

Logical Formulas

Follow the logical formulas and do as well as the hand-designed formulas.

Problem-specific

Problem-specific

When to stop: use a special $<\text{end}>$ 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):

<table>
<thead>
<tr>
<th>Task</th>
<th>RNN</th>
<th>LSTM</th>
<th>GG-NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAbI Task 15 (path finding)</td>
<td>71.7 ± 4.3 (50)</td>
<td>78.2 ± 4.7 (50)</td>
<td>95.0 ± 1.5 (50)</td>
</tr>
<tr>
<td>Swift Path</td>
<td>74.1 ± 2.7 (950)</td>
<td>80.2 ± 2.1 (950)</td>
<td>98.5 ± 0.7 (950)</td>
</tr>
<tr>
<td>Eulerian Circuit</td>
<td>33.4 ± 2.0 (50)</td>
<td>33.4 ± 2.0 (50)</td>
<td>100.0 ± 0.0 (50)</td>
</tr>
</tbody>
</table>

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

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.

<table>
<thead>
<tr>
<th>Code</th>
<th>GG-NN</th>
<th>Theorem Prover</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap Graphs</td>
<td>GGS-NN</td>
<td>Theorem Prover</td>
</tr>
<tr>
<td>Run</td>
<td>GGS-NN</td>
<td>Theorem Prover</td>
</tr>
<tr>
<td>Return</td>
<td>GGS-NN</td>
<td>Theorem Prover</td>
</tr>
<tr>
<td>Code</td>
<td>GGS-NN</td>
<td>Theorem Prover</td>
</tr>
</tbody>
</table>

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