Learning Convolutional Neural Networks for Graphs

Mathias Niepert  Mohamed Ahmed  Konstantin Kutzkov

NEC Laboratories Europe
Representation Learning for Graphs

Intermediate Representation: **Graphs**

Deep Learning System

(Edge) deployment
**Problem Definition**

**Input:** Finite collection of graphs

- Nodes of any two graphs are *not* necessarily in correspondence
- Nodes and edges may have attributes (discrete and continuous)

**Problem:** Learn a representation for classification/regression

**Example:** Graph classification problem

\[
\text{class}\left[\begin{array}{c}
\end{array}\right] = ?
\]
State of the Art: Graph Kernels

Define kernel based on substructures
- Shortest paths
- Random walks
- Subtrees
- ...

Kernel is similarity function on pairs of graphs
- Count the number of common substructures

Use graph kernels with SVMs
Patchy: Learning CNNs for Graphs
node sequence selection (w=6 nodes)
node sequence selection (w=6 nodes)
node sequence selection (w=6 nodes)
node sequence selection (w=6 nodes)
node sequence selection (w=6 nodes)
Patchy: Learning CNNs for Graphs

node sequence selection (w=6 nodes)
Patchy: Learning CNNs for Graphs

neighborhood assembly (at least k=4 nodes)

node sequence selection (w=6 nodes)
Patchy: Learning CNNs for Graphs

neighborhood assembly (at least k=4 nodes)

node sequence selection (w=6 nodes)
Patchy: Learning CNNs for Graphs

node sequence selection (w=6 nodes)

neighborhood assembly (at least k=4 nodes)
Patchy: Learning CNNs for Graphs

neighborhood assembly (at least k=4 nodes)

node sequence selection (w=6 nodes)
Patchy: Learning CNNs for Graphs

neighborhood assembly (at least k=4 nodes)

node sequence selection (w=6 nodes)
Patchy: Learning CNNs for Graphs

neighborhood assembly (at least k=4 nodes)

node sequence selection (w=6 nodes)
Patchy: Learning CNNs for Graphs

- Neighborhood normalization (exactly $k=4$ nodes)
- Neighborhood assembly (at least $k=4$ nodes)
- Node sequence selection ($w=6$ nodes)
Patchy: Learning CNNs for Graphs

Convolutional architecture

- normalized neighborhoods serve as **receptive fields**
- node and edge attributes correspond to **channels**

neighborhood normalization
Node Sequence Selection

We use centrality measures to generate the node sequences. Nodes with similar structural roles are aligned across graphs.

A: Betweenness centrality
B: Closeness centrality
C: Eigenvector centrality
D: Degree centrality
Neighborhood Assembly

Simple breadth-first expansion until at least $k$ nodes added, or no additional nodes to add
Nodes of any two graphs should have similar position in the adjacency matrices iff their structural roles are similar.

\[
\arg \min_{\ell} \mathbb{E}_G \left[ \|d_A \left( A^\ell (G), A^\ell (G') \right) - d_G(G, G') \| \right]
\]

Result: For several distance measure pairs it is possible to efficiently compare labeling methods without supervision.

Example: \(\|A - A'\|_1\) and edit distance on graphs.
Graph Normalization

Distance to root node
Graph Normalization

Centrality measures, etc.

Distance to root node
Graph Normalization

- Canonicalization (break ties)
- Centrality measures, etc.
- Distance to root node

Learning Convolutional Neural Networks for Graphs
Computational Complexity

- At most *linear* in number of input graphs
- At most *quadratic* in number of nodes for each graph (depends on maximal node degree and centrality measure)
Convolutional Architecture

Field size: 4, stride: 4, filters: M
Convolutional Architecture

field size: 4, stride: 4, filters: M
Convolutional Architecture

field size: 4, stride: 4, filters: M
Convolutional Architecture

field size: 3, stride: 1, N filters

field size: 4, stride: 4, filters: M
Convolutional Architecture

- Field size: 3, stride: 1, N filters
- Field size: 4, stride: 4, filters: M
Convolutional Architecture

- Field size: 3, stride: 1, N filters
- Field size: 4, stride: 4, filters: M
Experiments - Graph Classification

Finite collection of graphs and their class labels

- Nodes of any two graphs are *not* necessarily in correspondence
- Nodes and edges may have attributes (discrete and continuous)

Learn a function from graphs to class labels

\[
\text{class} = \begin{cases} 
1 & \text{class} = 0 \\
0 & \text{class} = 1 
\end{cases}
\]

\[
\text{class}\left[ \begin{array}{c}
\text{graph} \\
\end{array} \right] = ?
\]
Experiments - Convolutional Architecture

softmax

flatten, dense 128 units

field size: 10, stride: 1, filters: 8

field size: k, stride: k, filters: 16

1 2 ... 10
Classification Datasets

**MUTAG:** Nitro compounds where classes indicate mutagenic effect on a bacterium (Salmonella Typhimurium)

**PTC:** Chemical compounds where classes indicate carcinogenicity for male and female rats

**NCI:** Chemical compounds where classes indicate activity against non-small cell lung cancer and ovarian cancer cell lines

**D&D:** Protein structures where classes indicate whether structure is an enzyme or not

...


**Q:** How efficient and effective compared to graph kernels?

Apply Patchy to typical graph classification benchmark data

<table>
<thead>
<tr>
<th>Data set</th>
<th>MUTAG</th>
<th>PCT</th>
<th>NCI1</th>
<th>PROTEIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>28</td>
<td>109</td>
<td>111</td>
<td>620</td>
</tr>
<tr>
<td>Avg</td>
<td>17.93</td>
<td>25.56</td>
<td>29.87</td>
<td>39.06</td>
</tr>
<tr>
<td>Graphs</td>
<td>188</td>
<td>344</td>
<td>4110</td>
<td>1113</td>
</tr>
<tr>
<td>SP [7]</td>
<td>85.79 ± 2.51</td>
<td>58.53 ± 2.55</td>
<td>73.00 ± 0.51</td>
<td>75.07 ± 0.54</td>
</tr>
<tr>
<td>RW [17]</td>
<td>83.68 ± 1.66</td>
<td>57.26 ± 1.30</td>
<td>&gt; 3 days</td>
<td>74.22 ± 0.42</td>
</tr>
<tr>
<td>GK [38]</td>
<td>81.58 ± 2.11</td>
<td>57.32 ± 1.13</td>
<td>62.28 ± 0.29</td>
<td>71.67 ± 0.55</td>
</tr>
<tr>
<td>WL [39]</td>
<td>80.72 ± 3.00 (5s)</td>
<td>56.97 ± 2.01 (30s)</td>
<td>80.22 ± 0.51 (375s)</td>
<td>72.92 ± 0.56 (143s)</td>
</tr>
<tr>
<td>PSCN k=5</td>
<td>91.58 ± 5.86 (2s)</td>
<td>59.43 ± 3.14 (4s)</td>
<td>72.80 ± 2.06 (59s)</td>
<td>74.10 ± 1.72 (22s)</td>
</tr>
<tr>
<td>PSCN k=10</td>
<td>88.95 ± 4.37 (3s)</td>
<td>62.29 ± 5.68 (6s)</td>
<td>76.34 ± 1.68 (76s)</td>
<td>75.00 ± 2.51 (30s)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data set</th>
<th>GK [38]</th>
<th>DGK [45]</th>
<th>PSCN k=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLAB</td>
<td>72.84 ± 0.28</td>
<td>73.09 ± 0.25</td>
<td>72.60 ± 2.15</td>
</tr>
<tr>
<td>IMDB-B</td>
<td>65.87 ± 0.98</td>
<td>66.96 ± 0.56</td>
<td>71.00 ± 2.29</td>
</tr>
<tr>
<td>IMDB-M</td>
<td>43.89 ± 0.38</td>
<td>44.55 ± 0.52</td>
<td>45.23 ± 2.84</td>
</tr>
<tr>
<td>RE-B</td>
<td>77.34 ± 0.18</td>
<td>78.04 ± 0.39</td>
<td>86.30 ± 1.58</td>
</tr>
<tr>
<td>RE-M5k</td>
<td>41.01 ± 0.17</td>
<td>41.27 ± 0.18</td>
<td>49.10 ± 0.70</td>
</tr>
<tr>
<td>RE-M10k</td>
<td>31.82 ± 0.08</td>
<td>32.22 ± 0.10</td>
<td>41.32 ± 0.42</td>
</tr>
</tbody>
</table>
Q: What do learned edge filters look like?
Restricted Boltzmann machine applied to graphs
Receptive field size of hidden layer: 9

graphs sampled from RBM
small instances of graphs
weights of hidden nodes
Discussion

Pros:
- Graph kernel design not required
- Outperforms graph kernels on several datasets (speed and accuracy)
- Incorporates node and edge features (discrete and continuous)
- Supports visualizations (graph motifs, etc.)

Cons:
- Prone to overfitting on smaller data sets (graph kernel benchmarks)
- Shift from designing graph kernels to tuning hyperparameters
- Graph normalization not part of learning

code to be released: patchy.neclab.eu