Graph Matching Networks for Learning the Similarity of Graph Structured Objects

Yujia Li, Chenjie Gu, Thomas Dullien*, Oriol Vinyals, Pushmeet Kohli
Graph structured data appear in many applications

Molecules

Scene Graphs*

Programs**

Binaries

Graph structured data appear in many applications in various fields such as:

- **Molecules** for Drug Discovery
- **Scene Graphs** for Semantic Image Retrieval
- **Programs** for Code Search
- **Binaries** for Software Vulnerabilities

Finding similar graphs

Graph structures vary a lot

Nodes and edges can have attributes

Reasoning about both the graph structure and the semantics

The notion of “similarity” varies across problems

Query Graph

Candidate Graphs
The binary function similarity search problem contains vulnerability?
The binary function similarity search problem

contains vulnerability?

binary analysis

graph sizes in our dataset: from 10 to $10^3$
The binary function similarity search problem

contains vulnerability?

binary analysis

search in a library of binaries with known vulnerabilities

similar

not similar
The binary function similarity search problem

search in a library of binaries with known vulnerabilities

contains vulnerability?

binary analysis

similar

not similar

EXE
Most existing approaches

Mostly hand-engineered algorithms / heuristics with limited learning:

**Graph hashes** *(graph → descriptor)*: widely used in security applications
- human-designed hash functions that encode graph structure
- good at exact matches, not so good at estimating similarity

**Graph kernels** *(pair of graphs → similarity)*: popular in various graph-level prediction tasks
- human-designed kernels as a measure of similarity between graphs
- the design of kernels is important for performance
### Different graph similarity estimation paradigms

<table>
<thead>
<tr>
<th>Graph embedding</th>
<th>Graph matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graph → descriptor</td>
<td>Compute distance jointly on the pair of graphs</td>
</tr>
<tr>
<td>Measure distance on descriptors</td>
<td>More computation for <strong>better accuracy</strong></td>
</tr>
<tr>
<td><strong>Fast</strong> hashing based retrieval</td>
<td></td>
</tr>
</tbody>
</table>
Graph similarity learning

Learn a similarity (or distance) function

d(, ) → small

d(, ) → large
Graph similarity learning

Learn a similarity (or distance) function

\[ d(\cdot, \cdot) \rightarrow \text{small} \]

\[ d(\cdot, \cdot) \rightarrow \text{large} \]

Supervised learning on labeled pairs or triplets

\[ L_{\text{pair}} = \mathbb{E}_{(G_1, G_2, t)}[\max\{0, \gamma - t(1 - d(G_1, G_2))\}] \]

\[ t = +1 \Rightarrow G_1, G_2 \text{ similar} \Rightarrow d(G_1, G_2) \downarrow \]

\[ t = -1 \Rightarrow G_1, G_2 \text{ not similar} \Rightarrow d(G_1, G_2) \uparrow \]

\[ L_{\text{triplet}} = \mathbb{E}_{(G_1, G_2, G_3)}[\max\{0, d(G_1, G_2) - d(G_1, G_3) + \gamma\}] \]

\[ G_1, G_2 \text{ similar, } G_1, G_3 \text{ not similar} \]
\[ \Rightarrow d(G_1, G_2) \downarrow \quad d(G_1, G_3) \uparrow \]
Learning graph embeddings with Graph Neural Nets

\[ d(G_1, G_2) = \text{Euclidean/Hamming distance}(\text{embed}(G_1), \text{embed}(G_2)) \]
Learning graph embeddings with Graph Neural Nets

\[ d(G_1, G_2) = \text{Euclidean/Hamming distance}(\text{embed}(G_1), \text{embed}(G_2)) \]

\[ \text{embed}(\text{Input Graph}) = \text{Message Passing} \rightarrow \text{Aggregate over Graph} \]
Graph embedding model details

Messages: \( \mathbf{m}_{u \rightarrow v} = f_{\text{message}}(\mathbf{h}_u^{(t)}, \mathbf{h}_v^{(t)}, \mathbf{e}_{uv}) \)

Node updates: \( \mathbf{h}_v^{(t+1)} = f_{\text{node}} \left( \mathbf{h}_v^{(t)}, \sum_{u \in N(v)} \mathbf{m}_{u \rightarrow v} \right) \)

\( \mathbf{h}_G = \text{MLP} \left( \text{POOL}(\{\mathbf{h}_v\}_{v \in V}) \right) \)

Aggregation:
- sum pooling, attention pooling etc.
Graph Matching Networks

\[ h_1, h_2 = \text{embed-and-match}(G_1, G_2) \]
\[ d(G_1, G_2) = \text{Euclidean/Hamming distance}(h_1, h_2) \]
Graph Matching Networks

\[ h_1, h_2 = \text{embed-and-match}(G_1, G_2) \]
\[ d(G_1, G_2) = \text{Euclidean/Hamming distance}(h_1, h_2) \]

Attention:
Weighted difference:

\[ a_{j \rightarrow i} = \text{Softmax}_j \left( s(h_i^{(t)}, h_j^{(t)}) \right) \]
\[ \mu_{j \rightarrow i} = a_{j \rightarrow i} (h_i^{(t)} - h_j^{(t)}) \]
Graph Matching Networks

\[ h_1, h_2 = \text{embed-and-match}(G_1, G_2) \]
\[ d(G_1, G_2) = \text{Euclidean/Hamming distance}(h_1, h_2) \]

Total cross-graph message

\[ \sum_j \mu_{j \rightarrow i} = \sum_j a_{j \rightarrow i}(h_i^{(t)} - h_j^{(t)}) = h_i^{(t)} - \sum_j a_{j \rightarrow i} h_j^{(t)} \]

Effectively: match node i to the closest node in the other graph and take the difference.

\[ h_i^{(t+1)} = f_{\text{node}} \left( h_i^{(t)}, \sum_j m_{j \rightarrow i}, \sum_{j'} \mu_{j' \rightarrow i} \right) \]
Other variants

Other variants of GNNs for embedding:
- e.g. Graph Convolutional Networks (GCNs), which is a simpler variant without modeling edge features

Siamese networks:
- instead of using Euclidean or Hamming distance, learn a distance score through a neural net
- \(d(G_1, G_2) = \text{MLP}(\text{concat(embed}(G_1), \text{embed}(G_2)))\)
- learn the embedding model and the scoring MLP jointly
Graph Matching Networks — Yujia Li

Graph Embedding

Similarity score

Siamese Network

Similarity score

Graph Matching

Similarity score
Experiments

**Graph edit distance learning**

*Data:* synthetic graphs

*Similarity:* small edit distance $\rightarrow$ similar

---

**Control-flow graph based binary function similarity search**

*Data:* compile *ffmpeg* with different compilers and optimization levels.

*Similarity:* binary functions associated with the same original function $\rightarrow$ similar

---

**Mesh graph retrieval**

*Data:* mesh graphs for 100 object classes (COIL-DEL dataset)

*Similarity:* mesh for the same object class $\rightarrow$ similar
Synthetic task: graph edit distance learning

Training and evaluating on graphs of size $n$, and edge density (probability) $p$

Measuring **pair classification AUC** / **triplet prediction accuracy**.

<table>
<thead>
<tr>
<th>Graph Spec.</th>
<th>WL kernel</th>
<th>embedding model</th>
<th>matching model</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 20, p = 0.2$</td>
<td>80.8 / 83.2</td>
<td>88.8 / 94.0</td>
<td><strong>95.0 / 95.6</strong></td>
</tr>
<tr>
<td>$n = 20, p = 0.5$</td>
<td>74.5 / 78.0</td>
<td>92.1 / 93.4</td>
<td><strong>96.6 / 98.0</strong></td>
</tr>
<tr>
<td>$n = 50, p = 0.2$</td>
<td>93.9 / <strong>97.8</strong></td>
<td>95.9 / 97.2</td>
<td><strong>97.4 / 97.6</strong></td>
</tr>
<tr>
<td>$n = 50, p = 0.5$</td>
<td>82.3 / 89.0</td>
<td>88.5 / 91.0</td>
<td><strong>93.8 / 92.6</strong></td>
</tr>
</tbody>
</table>

Learned models do better than WL kernel.

Matching model better than embedding model.
Results on binary function similarity search

Hand-engineered baseline (graph hashing + locality sensitive hashing) vs GNN embedding vs GMN.

Graph topology only vs jointly over structures and features.
Results on binary function similarity search

1) **learned approaches** better than **hand-engineered solution**
2) **matching** better than **embedding** alone
3) joint modeling of **structure and features** better than **structure alone**
4) performance better with **more graph propagation steps**
### More ablation studies

<table>
<thead>
<tr>
<th>Model</th>
<th>Pair AUC</th>
<th>Triplet Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>96.09</td>
<td>96.35</td>
</tr>
<tr>
<td>GCN</td>
<td>96.67</td>
<td>96.57</td>
</tr>
<tr>
<td>Siamese-GCN</td>
<td>97.54</td>
<td>97.51</td>
</tr>
<tr>
<td>GNN</td>
<td>97.71</td>
<td>97.83</td>
</tr>
<tr>
<td>Siamese-GNN</td>
<td>97.76</td>
<td>97.58</td>
</tr>
<tr>
<td>GMN</td>
<td>99.28</td>
<td>99.18</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Pair AUC</th>
<th>Triplet Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCN</td>
<td>94.80</td>
<td>94.95</td>
</tr>
<tr>
<td>Siamese-GCN</td>
<td>95.90</td>
<td>96.10</td>
</tr>
<tr>
<td>GNN</td>
<td>98.58</td>
<td>98.70</td>
</tr>
<tr>
<td>Siamese-GNN</td>
<td>98.76</td>
<td>98.55</td>
</tr>
<tr>
<td>GMN</td>
<td>98.97</td>
<td>98.80</td>
</tr>
</tbody>
</table>

GMNs consistently better than alternatives.

**Siamese vs matching**: fusing two graphs early better than only at the end.
Learned attention patterns

We never supervise the cross-graph attention, but the model still learns some interesting attention patterns.
Learned attention patterns

When the two graphs are identical, the learned attention pattern may (not always) correspond to node matching.

Model trained on the edit distance learning task.
Learned attention patterns

Otherwise the attention pattern is less interpretable.

After 10 message passing steps

Model trained on the edit distance learning task.
Conclusions and future directions

Takeaways:
- graph similarity can be learned.
- learned graph embedding models are good and efficient models for this.
- graph matching networks are even better.

Future directions:
- make cross-graph attention and matching more efficient
- explore new architectures that can utilize the new capability of learned graph similarity
Graph Matching Networks for Learning the Similarity of Graph Structured Objects

Yujia Li, Chenjie Gu, Thomas Dullien*, Oriol Vinyals, Pushmeet Kohli

DeepMind

*Google