Towards Sparse Hierarchical Graph Classifiers

Cătălina Cangea*, Petar Veličković*, Nikola Jovanović, Thomas Kipf, Pietro Liò

NIPS 2018 Relational Representation Learning Workshop [arxiv.org/abs/1811.01287]

Graph Representation Reading Group, Mila **15 November 2018**

Graph classification

- Typical machine learning task aimed at graph-structured data (e.g. want to classify molecular structures, citation networks, social circles)
- Graphs represent the **generalization** of images (grid-like graphs) $\rightarrow$ aim to extend readily established *image classification* techniques
- Typical image classification pipeline: alternate between convolution (feature detection) and pooling (down-sampling)—want this for graphs!

Existing work

- Graph convolutional layers: plenty!
  - *Spectral*: Bruna et al.(2014), Defferrard et al.(2016), Kipf et al.(2017)
  - *Message-passing*: Gilmer et al.(2017)
  - *Attention-based*: Monti et al.(2016), Veličković et al.(2017)
- Pooling mechanisms: comparatively few...
  - Globally pool after each/the final message-passing step
  - *Progressively coarsen* the graph—most approaches assume a pre-defined assignment of nodes to clusters
  - **Learn a differentiable pooling mechanism based on the *structure of the data*: Ying et al.(2018), Anon.(2018)**

Hierarchical (pooling) strategies

- *DiffPool*: compute *soft clustering* assignments of nodes from the original graph to nodes from the resulting graph
- End-to-end trainable architecture with SOTA results
- Main issue: soft clustering assignments require **storing the assignment matrix and therefore $O(kV^2)$ memory!**

- *Graph U-Net*: does not suffer from the issue above, but the architecture was not evaluated on classification task benchmarks

Our contributions

- An intuitive architecture for graph classification that closely resembles the computation model for image classification
- Results comparable to state-of-the-art on classification benchmark tasks
- A drastic reduction in the GPU memory requirement (from $O(V^2)$ to $O(V + E)$)

Model

- Input graph $\rightarrow$ a matrix of node features, $\mathbf{X} \in \mathbb{R}^{N \times F}$, and an adjacency matrix, $\mathbf{A} \in \mathbb{R}^{N \times N}$
- $\mathbf{A}$ is binary and symmetric
- If the graph is featureless, use 1-hot encoding of node degree information to construct $\mathbf{X}$
- A CNN-inspired network for graph classification should contain the following layers: *convolutional*, ***pooling***, *readout* (i.e. flattening layer in an image CNN used for the final prediction)

Model

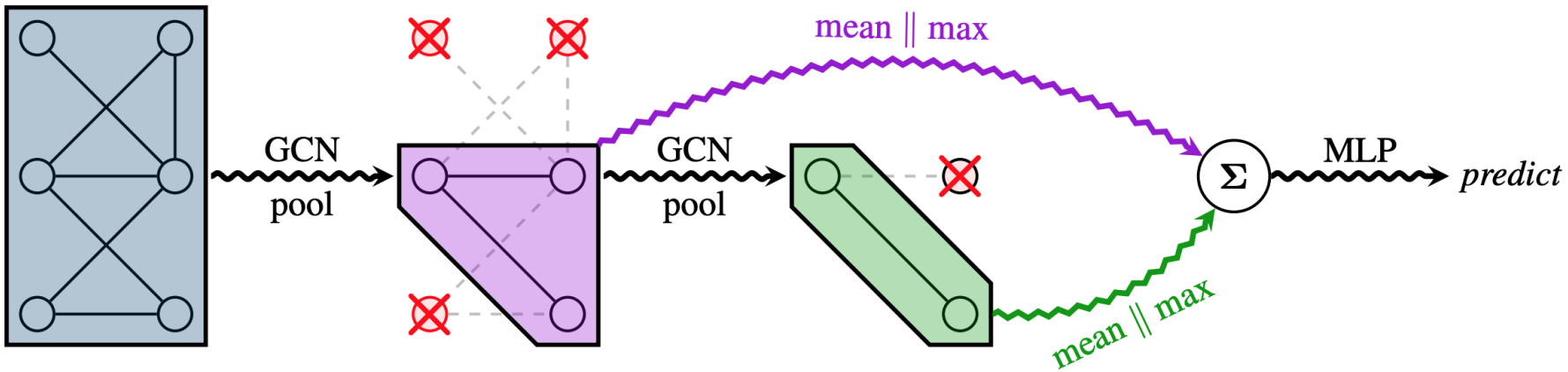


Figure 1: The full pipeline of our model (for $k = 0.5$), leveraging several stacks of interleaved convolutional/pooling layers (that, unlike DiffPool, *drop* rather than aggregate nodes), as well as a JK-net-style summary, combining information at different scales.

Convolutional layer

- Apply the *mean-pooling* propagation rule (as in GCN or Const-GAT):

$$MP(\mathbf{X}, \mathbf{A}) = \sigma\left(\hat{\mathbf{D}}^{-1}\hat{\mathbf{A}}\mathbf{X}\mathbf{\Theta} + \mathbf{X}\mathbf{\Theta}'\right)$$

- $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with self-loops
- $\hat{\mathbf{D}}$ is the degree matrix: $\hat{D}_{ii} = \sum_j \hat{A}_{ij}$

Pooling layer

- Use Graph U-Net architecture to reduce the number of nodes N via *pooling ratio* $k \in (0, 1]$
- Layer outputs $\lceil kN \rceil$ nodes
- *Learn* projection score $\vec{p}$ used as *gating values* $\rightarrow$ ensure lower scoring nodes retain comparatively less features

Pooling layer—cont'd

- Obtain pooled graph $(\mathbf{X}', \mathbf{A}')$ as:

$$\begin{aligned}
\vec{y} &= \frac{\mathbf{X}\vec{p}}{\|\vec{p}\|}, \\
\vec{i} &= \text{top-}k(\vec{y}, k), \\
\mathbf{X}' &= (\mathbf{X} \cdot \tanh(\vec{y}))_{\vec{i}}, \\
\mathbf{A}' &= \mathbf{A}_{\vec{i},\vec{i}}
\end{aligned}$$

- **This only requires a pointwise projection operation and slicing into the original matrices!**

Readout layer

- Want "flattening" operation analogous to image CNNs
- Take the average and max of all learnt node embeddings for output graph $(\mathbf{X}^k, \mathbf{A}^k)$ of the k-th block:

$$\vec{s}^k = \frac{1}{N^k} \sum_{i=1}^{N^k} \vec{x}_i^k \| \max_{i=1}^{N^k} \vec{x}_i^k$$

- Inspired by JK-net, summarise the graph by summing:

$$\vec{s} = \sum_{k=1}^{K} \vec{s}^k$$

- An MLP predicts the class for $\vec{s}$ at the "tail" of the model

Experiments

- Benchmark datasets: Enzymes, Proteins, DD, Collab
- 10-fold cross-validation, compare to Ying et al.(2018)

- Three $\{$GCN layer, pooling layer$\}$ blocks
- Preserve $k = 80\%$ of the nodes on each pool

Results

Model	Datasets			
	Enzymes	*D&D*	*Collab*	*Proteins*
Graphlet	41.03	74.85	64.66	72.91
Shortest-path	42.32	78.86	59.10	76.43
1-WL	53.43	74.02	78.61	73.76
WL-QA	60.13	79.04	80.74	75.26
PatchySAN	–	76.27	72.60	75.00
GraphSAGE	54.25	75.42	68.25	70.48
ECC	53.50	74.10	67.79	72.65
Set2Set	60.15	78.12	71.75	74.29
SortPool	57.12	79.37	73.76	75.54
DiffPool-Det	58.33	75.47	**82.13**	75.62
DiffPool-NoLP	62.67	79.98	75.63	77.42
DiffPool	**64.23**	**81.15**	75.50	**78.10**
Ours	64.17	78.59	74.54	75.46

Memory usage

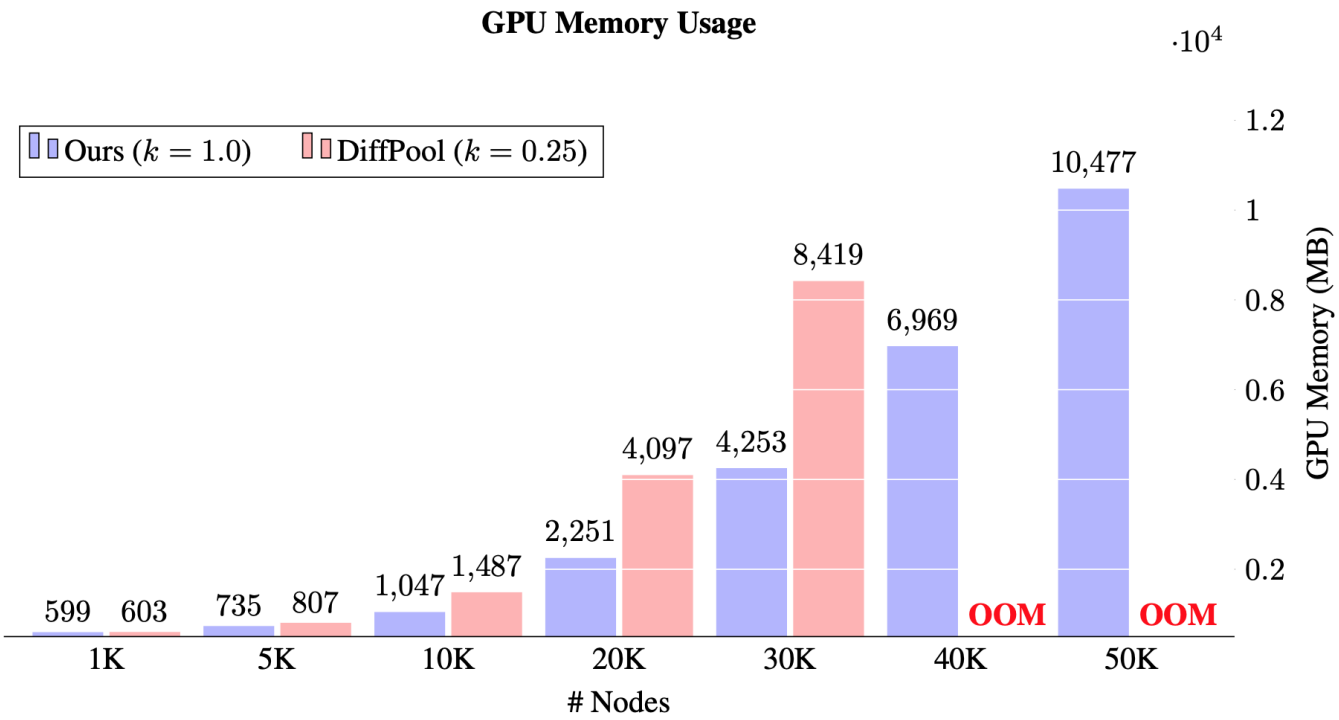
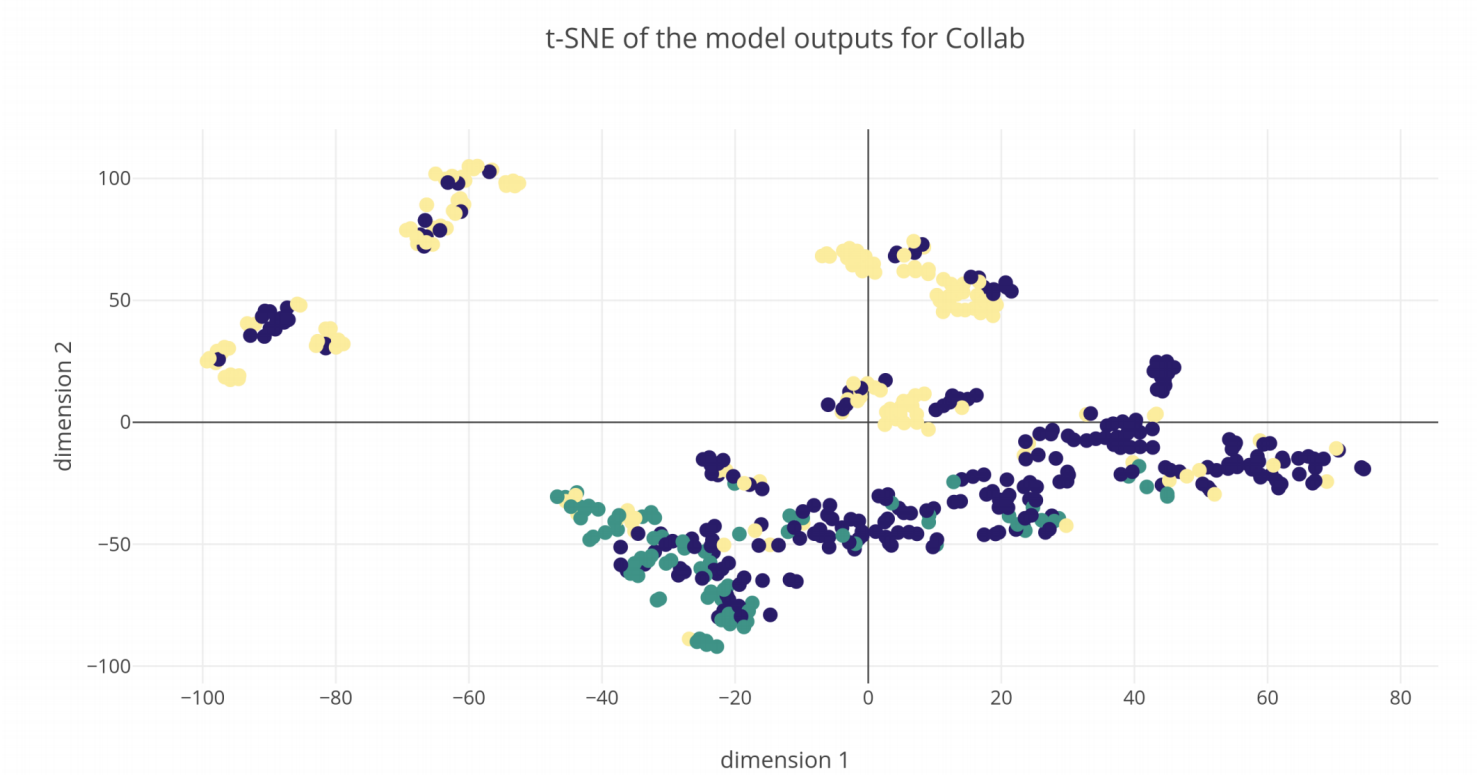


Figure 2: GPU memory usage of our method (with no pooling; $k = 1.0$) and DiffPool ($k = 0.25$) during training on Erdős-Rényi graphs [9] of varying node sizes (and $|E| = 2|V|$). Both methods ran with 128 input and hidden features, and three Conv-Pool layers. "OOM" denotes out-of-memory.

Qualitative analysis



Future directions

- Apply the computation model to **large datasets** (particularly relevant for the neuroscience domain, where *brain meshes* are encoded by graphs with +100K nodes).
- Improve the pooling mechanism (e.g. by making the pooling ratio k learnable).
- Make the “mirroring” operation (unpooling) independent of the pooling results.

Thank you!

Questions?

Catalina.Cangea@cst.cam.ac.uk
www.cst.cam.ac.uk/~ccc53/