Learning to Aggregate on Structured Data

Approach Outline

Clemens Damke

Matriculation Number: 7011488

cdamke@mail.uni-paderborn.de

December 16, 2019

This is a brief outline of the approach chosen to tackle the problem of *Learning to Aggregate* (LTA) on structured data. The problem was split into three parts:

1. **Formalization of LTA:** Before LTA can be extended, its essential characteristics have to be defined. Those characteristics should provide the terminology to formally capture the differences and similarities between LTA and existing *graph classification and regression* (GC/GR) methods.
2. **Give an LTA interpretation of existing GC/GR methods:** Using the LTA formalization, representative GC/GR approaches should be restated as LTA instances. Currently there is no comprehensive formulation of the relation between both fields of research; this is addressed by the the second goal.
3. **Define an LTA method for graphs:** Using the LTA perspective on GC/GR, hidden assumptions of the existing approaches should become clear and in which way they share the assumptions of LTA. The last goal is to use those insights to formulate an LTA-GC/GR method that combines ideas from the existing approaches with the LTA assumptions.

1 LTA Formalization

First a brief sketch of the proposed general formalization of LTA. It consists of three properties that each LTA approach has to fulfill:

1. **Decomposition:** The input composition first has to be decomposed into its constituents. In the existing approaches for multisets this property is trivially fulfilled by considering each set element as a constituent. For structured data, i.e. graphs, the decomposition problem becomes more interesting. While one could consider the set of vertices as a graph's constituents, an approach that makes use of the relational structure is most likely more suitable.

   Formally a solution to the decomposition problem is described by a function $\varphi : \mathcal{G} \to \mathcal{P}(\mathcal{C})$, where $\mathcal{G}$ is the set of all structured inputs and $\mathcal{C}$ is the set of all constituents that may be found in those inputs.

2. **Disaggregation:** The constituents that are determined by φ are scored via a function $f : \mathcal{C} \to \mathcal{Y}$. The disaggregation problem is solved by learning this function.

3. **Aggregation:** Finally the aggregation problem is about finding an aggregation function $\mathcal{A}_{\text{SI}} : \mathcal{Y}^* \to \mathcal{Y}$ that combines the constituent scores. This is the definition of a *structurally independent* (SI) aggregation function, i.e. it only depends on the local constituent scores and does not use global structural information to potentially weight the scores. The existing LTA methods necessarily learn SI-aggregators since their inputs do not contain any structural information.

   For structured inputs, a more general definition of aggregation functions can be considered. A *structurally dependent* (SD) aggregator not only gets constituent scores $f(c) \in \mathcal{Y}$ but also structural features $g(c, G) \in \mathcal{Z}$ for all constituents $c \in \varphi(G)$, i.e. $\mathcal{A}_{\text{SD}} : (\mathcal{Y} \times \mathcal{Z})^* \to \mathcal{Y}$. To guarantee that the aggregated score is based on the constituent scores, an SD-aggregator is only allowed to use the structural features to weight and filter the constituent scores:

$$\mathcal{A}_{\text{SD}}(S) = \mathcal{A}_{\text{SI}}(\{w_i \circ y_i \mid (y_i, z_i) \in S \wedge w_i = h((y_i, z_i), S) \neq \texttt{nil}\}). \tag{1}$$

   Here $\circ : W \times \mathcal{Y} \to \mathcal{Y}$ is some score scaling operator on $\mathcal{Y}$, e.g. real multiplication if $\mathcal{Y} = \mathbb{R}$. Also note that structural constituent filtering is described

via $w_i = \texttt{nil}$ which excludes the constituent c_i from the aggregated score.

To summarize, the output y_i of an LTA model for an input graph G_i has to be expressible via

$$y_i = \mathcal{A}_{\mathrm{SD}}(\{(f(c), g(c, G_i)) \,|\, c \in \varphi(G_i)\}). \tag{2}$$

One advantage of this class of models is the explainability of the outputs. An aggregated score can always be tracked back to the individual constituents that went into it.

2 LTA Interpretation of GC/GR Methods

Next a very brief sketch of existing GC/GR methods and their relation to LTA is provided here.

1. **Graph Kernels:** Graph kernels do not directly allow for an LTA interpretation; for example an SVM with a graph kernel does not fulfill the previously described disaggregation and aggregation properties of LTA methods. Many graph kernels do however effectively compute graph decompositions and can therefore serve as starting points for graph LTA methods. The following graph kernels are interesting from an LTA perspective:
   - The *Weisfeiler-Lehman* (WL) subtree kernel [1] decomposes a graph into subtrees of depth k with each vertex being the root of exactly one subtree. The subtrees correspond to the breadth-first-walk trees with vertex repetition. Subtrees are identified via their isomorphism class.
   - The Direct-Product kernel conceptually decomposes a graph into the set of all random walks with arbitrary finite length. The walks are identified by the sequence of vertex labels along each walk.
   - Other relevant kernels include the WL edge kernel, the WL shortest path kernel, the graphlet kernel and the GraphHopper kernel. Those kernels also compute graph decompositions. The details of those decompositions are not described in this summary.
2. **Graph Neural Networks:** Most existing *graph neural network* (GNN) approaches fulfill the three previously described LTA properties. All GNN

methods that are based vertex-neighborhood aggregation can be understood as continuous variants of the *1-dimensional WL algorithm* (1-WL) [2]. They are therefore closely related to the WL subtree kernel and in fact upper bounded by it in terms of their discriminative power. They effectively also compute a subtree decomposition. The constituent scoring function f is modeled as a *multilayer perceptron* (MLP). Between each MLP layer a weighted average of the features of all vertices in the constituent subtree is taken. The last layer of the MLP has to output a value which can be interpreted as an element of $\mathcal{Y}$.

Graph pooling layers like SortPooling [3] or the self-attention based SAGPooling [4] can then be added to solve the aggregation problem. SortPooling can be understood as an SI-aggregator, while SAGPooling is an SD-aggregator with $z_i = g(c, G)$ being self-attention scores for the subtrees. In SAGPool g is modeled as a second neighborhood-averaging MLP similar to f.

The details of the relation between LTA and GNNs are not described here.

3 Proposed Graph LTA method

The proposed graph LTA method is based on two main ideas:

1. While most existing GNN methods can be interpreted as LTA approaches, they use a fairly static decomposition approach. In this approach each constituent is a tree that includes all vertices with a distance of at most k from some root vertex. The vertices that are part of a constituent therefore do not necessarily have any significant relation to each other. The decompositions of existing GNN methods therefore do not have any semantic. A learnable decomposition strategy could improve upon this problem.

2. As previously mentioned, the discriminative power of most GNNs is bounded by the 1-WL algorithm. The limits of this algorithm are well understood and while 1-WL is able to distinguish most graphs, it cannot detect many graph properties that are considered to be important in various domains of graph analysis, e.g. triangle counts in the context social networks or cycle counts in the context of molecular analysis. 1-WL also fails to distinguish regular graphs with the same degree and vertex count. There are higher dimensional variants of the WL algorithm that are able to capture such aspects. The

2-dimensional WL algorithm, which aggregates edge neighborhoods instead of vertex neighborhoods, is already powerful enough to recognize triangles, cycles of length ≤ 6 and most regular graphs. A GNN based on a higher dimensional WL algorithm could therefore at least theoretically improve upon the expressive power of most existing state-of-the-art approaches.

References

[1] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. *Weisfeiler-Lehman Graph Kernels*. 2011 (cit. on p. 3).

[2] Boris Weisfeiler and Andrei A. Lehman. "A reduction of a graph to a canonical form and an algebra arising during this reduction." In: *Nauchno-Technicheskaya Informatsia* 2.9 (1968), pp. 12–16 (cit. on p. 4).

[3] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. "An end-to-end deep learning architecture for graph classification." In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018 (cit. on p. 4).

[4] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. "Self-Attention Graph Pooling." In: (Apr. 17, 2019). arXiv: `1904.08082v4 [cs.LG]` (cit. on p. 4).