Learning to Aggregate on Structured Data

Approach Outline

Clemens Damke

Matriculation Number: 7011488

cdamke@mail.uni-paderborn.de

December 17, 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. **Giving 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. **Defining 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 some combination of the constituent scores and not arbitrarily derived from the structural features, an SD-aggregator is only allowed to use the structural features to weight and filter the constituent scores:

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

   Here $\circ : W \times \mathcal{Y} \to \mathcal{Y}$ is some score scaling operator on $\mathcal{Y}$, e.g. real multipli-

cation 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

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

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.

1. **Graph Kernels:** Graph kernels do not directly allow for an LTA interpretation; an SVM with a graph kernel for example does not fulfill the previously described disaggregation and aggregation properties of LTA methods. Many graph kernels do however effectively compute graph decompositions $\varphi(G)$ 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 [3][4]. 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 [5] or the self-attention based SAGPooling [6] 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. **Extending discriminative power:** 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 practical 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* (2-WL), which aggregates edge neighborhoods instead of vertex neighborhoods, is already powerful enough to recognize triangles, cycles of length ≤ 6 and also many regular graphs [7]. 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.

2. **Learned decomposition:** While most existing GNN methods can be interpreted as LTA approaches, they use a fairly static decomposition approach. In the static 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 thus do not have any semantic attached to them, they are purely structural. A learnable decomposition strategy could improve upon this problem.

The proposed method incorporates the first idea by defining a graph convolution operator inspired by the 2-WL algorithm. Recently Morris et al. [4] have already introduced a GNN architecture based on k-WL which does however use a simplified notion of neighborhood. This simplification reduces the discriminative power of the architecture; their 2-GNN for example cannot recognize triangles in graphs. The proposed method on the other hand is closer to the original 2-WL algorithm.

While 1-WL-based GNNs propagate vertex features via the adjacency structure of an input graph, a 2-WL-based GNN propagates edge features to neighboring edges. The aggregation of edge features performed by one network layer l is described by

$$h_{ij}^{(l)} = \sigma\left(W^{(l)} h_{ij}^{(l-1)} + W_{\Gamma}^{(l)} \sum_{k=1}^{n} \kappa\left(h_{ik}^{(l-1)}, h_{kj}^{(l-1)} \right) \right), \tag{3}$$

$$\text{with } h_{ij}^{(0)} = A_{ij} \oplus \begin{cases} X_i & \text{if } i = j \\ \mathbf{0} & \text{else} \end{cases} \tag{4}$$

where $\left\{ h^{(l)} \in \mathbb{R}^{n \times n \times F^{(l)}} \right\}_{l=0}^{L}$ are the edge feature tensors after each convolution layer. The initial edge features $h^{(0)}$ are constructed by concatenating ($\oplus$) the input edge features/weights A with the input vertex features X along the diagonal, i.e. self-loops are used to carry the vertex features.

The 2-WL inspired edge convolution combines each edge (i, j) with the edges in all walks of length 2 between i and j. First each walk is aggregated via $\kappa : \mathbb{R}^F \times \mathbb{R}^F \to \mathbb{R}^F$ which has to satisfy $\kappa(a, b) = \mathbf{0}$ if $a = \mathbf{0} \vee b = \mathbf{0}$ to preserve the sparsity of $h^{(l)}$ for performance reasons[1]. Then the sum of all aggregated walks is used as a neighborhood aggregate. Finally $W^{(l)}, W_{\Gamma}^{(l)} \in \mathbb{R}^{F^{(l)} \times F^{(l-1)}}$ and some

[1] κ should not be a linear operator, since this would reduce the discriminative power of the model back to 1-WL. Element-wise multiplication would be a simple suitable choice for κ but a learned nonlinear combinator can also be used.

activation function σ are used to compute the output edge features of the layer.

The choice of 2-WL as a basis for the just described GNN model was not only motivated by its larger discriminative power but also by the idea of learned graph decompositions. If the final layer of the described model produces outputs that are interpretable as elements of $\mathcal{Y}$, the graph is effectively decomposed into up to one constituent per edge[2]. Every convolution layer increases the radius of the neighborhood that is considered in the final feature aggregate of each edge/constituent. By learning a filter on the edges one can selectively remove elements from edge neighborhoods since disconnected parts of a graph cannot become part of the same neighborhood. The proposed 2-WL convolution model is well suited for a combination with such an edge filter model since all edges already have feature vectors that can be used as inputs for the filter model. The details for this still have to be worked out.

To aggregate the edges/constituents, existing graph pooling approaches like SortPooling or SAGPooling can be used.

4 Remarks and Current Status

I have currently only evaluated the proposed 2-WL inspired GNN architecture on synthetic datasets and the MUTAG dataset[3]. On the synthetic datasets I was able to clearly demonstrate that most state-of-the-art GNNs fail to learn on datasets consisting of regular graphs as well as datasets with graph labels derived from triangle counts. The proposed architecture on the other hand is able to learn the labeling function on those synthetic datasets. On the MUTAG dataset it performed more or less like the standard GCN model proposed by Kipf and Welling [8]. However the implementation currently only uses a static arithmetic average as the aggregation function, so there is probably still room for improvement.

The next planned steps are:

1. Use sparse tensors in the implementation in order to be able to evaluate on larger real datasets.

[2] Edges with a zero feature vector are not considered to be constituents.

[3] The implementation does not yet use sparse tensors and therefore cannot handle other real datasets that contain large graphs.

2. Integrate the previously described edge filtering idea and formalize how exactly this affects the learned constituents.

3. Swap out the constituent averaging with a learned aggregation/pooling layer.

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] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How Powerful are Graph Neural Networks?" In: (Oct. 1, 2018). arXiv: `1810.00826v3 [cs.LG]` (cit. on p. 4).

[4] Christopher Morris, Martin Ritzert, Matthias Fey, et al. "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks." In: (Oct. 4, 2018). arXiv: `1810.02244v2 [cs.LG]` (cit. on pp. 4, 5).

[5] 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).

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

[7] Martin Fürer. "On the Combinatorial Power of the Weisfeiler-Lehman Algorithm." In: (Apr. 4, 2017). arXiv: `1704.01023v1 [cs.DS]` (cit. on p. 4).

[8] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." In: *ICLR* (Sept. 9, 2016). arXiv: `1609.02907v4 [cs.LG]` (cit. on p. 6).