Learning to Aggregate on Structured Data

Master Thesis Proposal & Work Plan

Clemens Damke

Matriculation Number: 7011488

cdamke@mail.uni-paderborn.de

October 12, 2019

1 Motivation

Most of the commonly used supervised machine learning techniques assume that instances are represented by d-dimensional feature vectors $x_i \in \mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$ for which some target value $y_i \in \mathcal{Y}$ should be predicted. In the regression setting the target domain $\mathcal{Y}$ is continuous, typically $\mathcal{Y} = \mathbb{R}$, whereas $\mathcal{Y}$ is some discrete set of classes in the classification setting.

Since not all data is well-suited for a fixed-dimensional vector representation, approaches that directly consider the structure of the input data might be more appropriate in such cases. One such case is the class of so-called *learning to aggregate* (LTA) problems as described by Melnikov and Hüllermeier [1]. There the instances are represented by compositions $\boldsymbol{c}_i$ of constituents $c_{i,j} \in \boldsymbol{c}_i$, i.e. variable-size multisets with $n_i = |\boldsymbol{c}_i|$. The assumption in LTA problems is that for all constituents $c_{i,j}$ a local score $y_{i,j} \in \mathcal{Y}$ is either given or computable. The set of those local scores should be indicative of the overall score $y_i \in \mathcal{Y}$ of the entire composition $\boldsymbol{c}_i$. LTA problems typically require two subproblems to be solved:

1. **Aggregation:** A variadic aggregation function $A : \mathcal{Y}^* \to \mathcal{Y}$ that estimates composite scores has to be learned, i.e. $y_i \approx \hat{y}_i = A(y_{i,1}, \ldots, y_{i,n_i})$. Typically

the aggregation function A should be associative and commutative to fit with the multiset-structure of compositions.

2. **Disaggregation:** In case the constituent scores $y_{i,j}$ are not given, they have to be derived from a constituent representation, e.g. a vector $v_{i,j} \in \mathcal{V}$. To learn this derivation function $f : \mathcal{V} \to \mathcal{Y}$, only the constituent vectors $\{v_{i,j}\}_{j=1}^{n_i}$ and the composite score y_i is given. Thus the constituent scores $y_{i,j}$ need to be *disaggregated* from y_i in order to learn f.

Overall LTA can be understood as the joint problem of learning the aggregation function A and the local score derivation function f.

Current LTA approaches only work with multiset inputs. In practice there is however often some relational structure among the constituents of a composition. This effectively turns LTA into a graph regression problem. The goal of this thesis is to look into the question of how aggregation function learning methods might be generalized to the graph setting.

2 Related Work

This thesis will be based on two currently mostly separate fields of research: **1.** Learning to Aggregate **2.** Graph classification. A short overview of the current state-of-the-art approaches in both fields will be given now.

2.1 Learning to Aggregate

Two main approaches to represent the aggregation function in LTA problems have been explored. The first approach uses *uninorms* [1] to do so. There the basic idea is to express composite scores as fuzzy truth assignments $y_i \in [0, 1]$. Such a composite assignment y_i is modeled as the result of a parameterized logical expression of constituent assignments $y_{i,j} \in [0, 1]$. As the logical expression that thus effectively aggregates the constituents, a uninorm U_λ is used. Depending on the parameter λ, U_λ interpolates between t-norms and t-conorms which are continuous generalizations of logical conjunction and disjunction respectively.

TODO: Details

Recently Melnikov and Hüllermeier [2] have also looked at an alternative class of aggregation function. Instead of using fuzzy logic to describe score aggregation, *ordered weighted average* (OWA) operators were used.

TODO: Details

2.2 Graph Classification

As previously mentioned, the addition of relations between constituents turns LTA into a graph regression problem. Most of the recent research in the field of learning from graph structured data however focused on the closely related graph classification problem. Since many ideas from the classification setting are also applicable in the regression setting, a brief overview of those ideas is given.

At a high level graph classification methods can be taxonomized into two main families:

1. **Vector representation approaches:** One way to tackle the graph classification problem is to map an input graph G to a vectorial representation. This can be done a) by either handpicking global graph features like vertex/edge count, degree distribution or graph diameter, b) via a graph embedding algorithm like node2vec [3], sub2vec [4] or graph2vec [5], c) implicitly by using a graph kernel that computes the structural similarity between graphs, e.g. the Weisfeiler-Lehman kernel [6] or the multiscale Laplacian graph kernel [7]. Graphs can then be classified via any classification algorithm that works with vectors and/or kernels.

2. **Graph neural networks:** An alternative approach is to adapt neural networks to graph inputs. The notion of a *graph neural network* (GNN) was first introduced by Gori et al. [8]. There a message-passing architecture is used to iteratively propagate vertex information to neighbors until a fixed point is reached. This process is computationally expensive.

   Recently the class of *convolutional graph neural networks* (ConvGNNs) gained traction. It generalizes the convolution operator that is used by *convolutional neural networks* (CNNs) to graphs. There are two main variants of ConvGNNs:

   a) **Spatial variant:** Standard CNNs use a convolution operator that is applied to a grid graph with diagonal edges. Spatial ConvGNNs directly generalize this idea by defining a more flexible convolution that aggregates the local neighborhood of each vertex. Spatial graph convolution aggregates the direct neighborhood of each vertex where each vertex v_i is typically described by a feature vector $x_i \in \mathbb{R}^k$. This process returns a new aggregate feature vector x_i'. By stacking multiple convolution layers, features from indirect neighbors become part of the aggregate. Analogous to how CNNs learn kernel matrices, spatial ConvGNNs learn vertex neighborhood feature aggregation functions. This approach shares similarities with the previously mentioned message-passing architecture [8] with the major different being that the number of message-passing, i.e. convolution, steps is fixed.

b) **Spectral variant:** Motivated by the theoretical foundation provided by spectral graph theory, spectral ConvGNNs learn a filter function $\hat{g}$ that is applied in the Fourier domain of a graph. Based on the convolutional theorem, a convolution operator can not only be expressed in terms of the neighborhood of vertices but also as a filter on the eigenvectors of a graph's Laplacian. Formally this can be expressed as

$$g *_G \boldsymbol{x} = U\hat{g}(\Lambda)U^\top \boldsymbol{x}$$

where $\boldsymbol{x} \in \mathbb{R}^{n\times k}$ is the matrix of all vertex features and $U\Lambda U^\top$ is the eigendecomposition of the graph Laplacian L_G. In the previously described spatial ConvGNN variant, g is learned directly in form of the feature aggregation function with a neighborhood locality constraint. In the spectral variant however the Fourier-transformed filter $\hat{g}$ is learned. Intuitively this means that vertex features x_i are not aggregated with the features of their their direct neighborhood but with the features of vertices with which they share certain structural similarities. This allows spectral ConvGNNs to incorporate the global graph structure at the cost of having to perform the computationally expensive decomposition of every input graph's Laplacian.

To tackle the performance impact of the spectral convolution approach, various simplifications of the filter $\hat{g}$ have been proposed. The so-called *graph convolutional network* (GCN) does this by reducing the expressivity of the filter. Then $\hat{g}$ can be applied to the Laplacian directly via $g *_G \boldsymbol{x} = \hat{g}(L_G)\boldsymbol{x}$ which saves the decomposition costs. This simplification of $\hat{g}$ implicitly causes its Fourier inverse g to be locality constrained just like in the spatial ConvGNN variant. Therefore the spectral GCN approach also allows for a spatial/message-passing interpretation. Recently various variants of GCN have been proposed that apply additional approximations to improve performance in certain scenarios.

After the application of either spatial or spectral graph convolutions, each vertex will have an updated aggregate feature vector. To obtain an aggregate score for the entire graph, a graph pooling layer is applied. A primitive pooling approach is to simply merge the vertex features via a fixed aggregation function like min, max or +. Better results can be obtained by learning the pooling aggregation function. Two notable graph pooling approaches are SortPooling and Self-Attention Pooling. SortPooling works by determining a top-k ordering of the vertices and then aggregates the remaining fixed-size vertex features via a standard multilayer perceptron (MLP).

3 Goals

3.1 Required Goals

3.2 Optional Goals

4 Approach

5 Preliminary Document Structure

1. Introduction
2. …

6 Time-Schedule

Figure 1: Sketch of the time schedule for the work on the thesis

References

[1] Vitalik Melnikov and Eyke Hüllermeier. "Learning to Aggregate Using Uninorms." In: *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, 2016, pp. 756–771 (cit. on pp. 1, 2).

[2] Vitalik Melnikov and Eyke Hüllermeier. "Learning to Aggregate: Tackling the Aggregation/Disaggregation Problem for OWA." In: *ACML* (2019) (cit. on p. 2).

[3] Aditya Grover and Jure Leskovec. "node2vec: Scalable Feature Learning for Networks." In: (July 3, 2016). arXiv: `http://arxiv.org/abs/1607.00653v1 [cs.SI]` (cit. on p. 3).

[4] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B. Aditya Prakash. "Sub2Vec: Feature Learning for Subgraphs." In: *Advances in Knowledge Discovery and Data Mining*. Springer International Publishing, 2018, pp. 170–182 (cit. on p. 3).

[5] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, et al. "graph2vec: Learning Distributed Representations of Graphs." In: (July 17, 2017). arXiv: `http://arxiv.org/abs/1707.05005v1 [cs.AI]` (cit. on p. 3).

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

[7] Risi Kondor and Horace Pan. "The Multiscale Laplacian Graph Kernel." In: (Mar. 20, 2016). arXiv: `http://arxiv.org/abs/1603.06186v2 [stat.ML]` (cit. on p. 3).

[8] M. Gori, G. Monfardini, and F. Scarselli. "A new model for learning in graph domains." In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* IEEE, 2005 (cit. on pp. 3, 4).

Supervisor	Student