# Learning to Aggregate on Structured Data

## Master Thesis Proposal & Work Plan

Clemens Damke

Matriculation Number: 7011488

cdamke@mail.uni-paderborn.de

October 16, 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 & regression. 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 $U_\lambda$ is used. Depending on the parameter $\lambda$, $U_\lambda$ interpolates between t-norms and t-conorms which are continuous generalizations of logical conjunction and disjunction respectively.

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. OWA aggregators work by sorting the input scores and then weighting them based on their sort position, i.e.

$$A_\lambda(y_1, \ldots, y_n) := \sum_{i=1}^{n} \lambda_i y_{\pi(i)},$$

where $\lambda$ is a weight vector with $\|\lambda\|_1 = 1$ and $\pi$ is a sorting permutation of the input scores. To deal with varying composite sizes $n$ the weights $\lambda_i$ are interpolated using a *basic unit interval monotone* (BUM) function $q : [0, 1] \to [0, 1]$. It is used by normalizing every constituent's position $\pi(i)$ to the unit interval via $\frac{\pi(i)}{n}$. The BUM function $q$ is then used to interpolate a weight for any normalized sort position. Therefore the learning objective boils down to optimizing the shape of $q$. The details of this are left out here.

## 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 useful 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 [9] 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 $x_i \in \mathbb{R}^d$. 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 [10], spectral ConvGNNs [11] 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) [12] 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 GCNs [13][14][15][16] have been proposed that apply additional approximations and sampling methods to improve the runtime as well as the accuracy 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* [17] and *Self-Attention Pooling* [18].

a) SortPooling assumes that the vertex feature vectors are obtained using a stack of spectral convolution layers that learn filters on the random walk Laplacian $L = I - D^{-1}A$. This assumption makes it possible to interpret the outputs of the graph convolution layers as a continuous generalization of *Weisfeiler-Lehman* (WL) colors. WL colors are lexicographically sortable signatures that encode the structural role of vertices; they can be used for efficient graph isomorphism falsification [19]. Using this interpretation of the vertex feature vectors, SortPooling determines a top-$k$ ranking of the vertices' WL colors and then aggregates the thus selected fixed-size vertex feature subset via a standard *multilayer perceptron* (MLP).

b) Self-Attention Pooling adds an additional graph convolution layer after an arbitrary ConvGNN. The added layer takes the aggregated vertex feature vectors as input and outputs a score for each vertex. This score

is used to determine a top-$k$ vertex ranking. Unlike SortPooling the feature vectors of the resulting subset are first added up. The resulting aggregate graph feature vector is then fed into a MLP to obtain a class.

## 2.3 Graph Regression

An overview of the recent work regarding graph classification was just given. The generalization of LTA to the graph domain is however better described as a graph regression problem since the predicted composite scores are continuous. As there currently are relatively few graph regression approaches, only few references can be provided here.

Graph regression problems have for the most part only been discussed in the context of specific domain problems. *RASAR* [20] is a graph regression method to predict the toxicity of chemicals as an alternative for animal tests. A standard logistic regression model is trained with manually chosen features that are given for the training molecule graphs. To predict the toxicity of a new chemical, $k$-means is used to average the features of chemicals with similar molecule graphs in the training set. The structural similarity between molecule graphs is measured by the Tanimoto coefficient of their chemical fingerprints. Those fingerprints encode the presence or absence of certain meaningful chemical substructures. The implementation and used dataset is unfortunately not publicly available because it is part of a commercial product[1].

*ProTox* [21][22] is another toxicity prediction approach. It works similarly to RASAR but leaves out the logistic regression step and instead directly aggregates the toxicity of similar chemicals. ProTox uses the freely available *SuperToxic* dataset [23]. While the implementation is again not publicly available, the trained model can be freely queried online[2].

# 3 Goals

As mentioned in section 1, the overall goal of the thesis is to extend the LTA approach described in section 2.1 to the graph setting. Since this goal strongly

[1] ToxTrack Inc., `https://toxtrack.com/`
[2] ProTox-II, `http://tox.charite.de/protox_II/`

overlaps with the work in graph classification described in section 2.2, both branches of research should be combined. The practical applications described in section 2.3 could be used for comparison and as a potential source of a graph regression test dataset. Now follows a concrete list of required and optional goals that should be achieved.

## 3.1 Required Goals

1. **Aggregation:** Define a class of aggregation functions that takes vertex scores as its input. It should fulfill the role of the uninorms and OWA functions in the existing LTA papers (see section 2.1).
2. **Disaggregation:** Describe how vertex features should be used to compute a constituent score that will be combined by the aggregation function.
3. **Learning:** Combine the chosen aggregation and disaggregation methods into a joint LTA model with a suitable loss function.
4. **Implementation:** A reference implementation for the designed LTA model should be developed.
5. **Evaluation:** The reference implementation should be used to evaluate the model on at least one test dataset. Due to the scarcity of graph regression datasets, the used test data might be randomly generated. Generated dataset should be obtained using a variety of representative deterministic graph scoring measures to check which kinds of local and global structural features are learnable; some examples of measures that might be part of a graph scoring functions are the count of $k$-cliques in a graph, the graph diameter, the presence of cycles or various properties of a graph's degree distribution. The results should be compared with the results of at least one other approach for learning graph scores, e.g. a ConvCNN with a fixed aggregation function instead of a learned one.

## 3.2 Optional Goals

If possible, the previous goals could be extended. For each of the previous five goal categories, optional additional ideas are listed below.

1. **Aggregation:** Multiple classes of aggregation functions could be considered and compared.
2. **Disaggregation:** As describen in section 2.2 there are multiple methods to learn vertex scores. Depending on the chosen overall approach, a few of those methods might be suitable to be combined with an aggregation function learner. Those suitable combinations could be evaluated and compared.
3. **Learning:** The joint model of aggregation and disaggregation function learning can most likely be optimized using various algorithms. Those algorithms could be compared. In case their runtime performance is impractical in certain situations, approximations and simplifications of the model could be considered.
4. **Implementation:** The chosen approach should ideally align well with the practical constraints of implementation. It should be suitable for an implementation on modern GPUs and/or distributed cluster environments.
5. **Evaluation:** It would be interesting to not only use generated data but also a real dataset for evaluation (see section 2.3).

# 4 Approach

# 5 Preliminary Document Structure

1. Introduction
2. 
3. Evaluation
4. Conclusion
5. Literature
6. Appendix

# 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).

[9] A. Micheli. "Neural Network for Graphs: A Contextual Constructive Approach." In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 498–511 (cit. on p. 4).

[10] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains." In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 83–98 (cit. on p. 4).

[11] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. "Spectral Networks and Locally Connected Networks on Graphs." In: (Dec. 21, 2013). arXiv: http://arxiv.org/abs/1312.6203v3 [cs.LG] (cit. on p. 4).

[12] 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. 5).

[13] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs." In: (June 7, 2017). arXiv: http://arxiv.org/abs/1706.02216v4 [cs.SI] (cit. on p. 5).

[14] Jie Chen, Tengfei Ma, and Cao Xiao. "FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling." In: (Jan. 30, 2018). arXiv: http://arxiv.org/abs/1801.10247v1 [cs.LG] (cit. on p. 5).

[15] Jianfei Chen, Jun Zhu, and Le Song. "Stochastic Training of Graph Convolutional Networks with Variance Reduction." In: (Oct. 29, 2017). arXiv: http://arxiv.org/abs/1710.10568v3 [stat.ML] (cit. on p. 5).

[16] Wei-Lin Chiang, Xuanqing Liu, Si Si, et al. "Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks." In: (May 20, 2019). arXiv: http://arxiv.org/abs/1905.07953v2 [cs.LG] (cit. on p. 5).

[17] 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. 5).

[18] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. "Self-Attention Graph Pooling." In: (Apr. 17, 2019). arXiv: http://arxiv.org/abs/1904.08082v4 [cs.LG] (cit. on p. 5).

[19] 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. 5).

[20] Thomas Luechtefeld, Dan Marsh, Craig Rowlands, and Thomas Hartung. "Machine Learning of Toxicological Big Data Enables Read-Across Structure Activity Relationships (RASAR) Outperforming Animal Test Reproducibility." In: *Toxicological Sciences* 165.1 (2018), pp. 198–212 (cit. on p. 6).

[21] Malgorzata N. Drwal, Priyanka Banerjee, Mathias Dunkel, Martin R. Wettig, and Robert Preissner. "ProTox: a web server for the in silico prediction of rodent oral toxicity." In: *Nucleic Acids Research* 42.W1 (2014), W53–W58 (cit. on p. 6).

[22] Priyanka Banerjee, Andreas O. Eckert, Anna K. Schrey, and Robert Preissner. "ProTox-II: a webserver for the prediction of toxicity of chemicals." In: *Nucleic Acids Research* 46.W1 (2018), W257–W263 (cit. on p. 6).

[23] U. Schmidt, S. Struck, B. Gruening, et al. "SuperToxic: a comprehensive database of toxic compounds." In: *Nucleic Acids Research* 37.Database (2009), pp. D295–D299 (cit. on p. 6).

Supervisor

Student