Vrije Universiteit Amsterdam

Bachelor Thesis

Benefits of data scalability and transfer learning for relational graph data (Machine learning - RGCN, scalability)

Author: Karim Anwar (2615855)

1st supervisor:	Michael Cochez
daily supervisor:	Michael Cochez
2nd reader:	Daniel Daza

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

July 14, 2021

Abstract

The goal of this paper is to present a framework that tackles the issue of data scalability for graph data. This framework is split into two parts, the first part consists of data summarization which consist of making a new graph by learning the connections done between node types which decrease the amount of memory needed especially that GPUs, even now that they are faster, are still, not match for CPUs when it comes to space and the learning time needed for the model will decrease as well. In the second part, transfer learning will be used to introduce the knowledge of one model to another with the original datasets, the parameters of the first model which will decrease the time needed to train, which will increase overall speed gain in the learning process.

1 Introduction

The recent success of convolutional neural networks (CNN) and deep learning in many domains is due to the rapidly developing computational resources (eg. GPU), and the availability of big training data (eg. Knowledge Bases), and the effectiveness of deep learning to extract latent representations. While deep learning effectively captures hidden patterns from Euclidean data, there is an increasing number of applications where the data is represented in the forms of graphs. From citation networks, which show how papers are linked to each other, to medical fields with representations of interactions between molecules which each represent a different graph. A new family of networks called graph neural networks (GNN) comes into existence with how graph data became ubiquitous in the Machine learning community. With the introduction of Graph Convolutional Networks (GCN) [1] which generalizes GNNs, learning can not be simpler than it is today. Knowledge Bases are growing at an explosive rate, and the same goes for the number of missing data in them. Ontologies like RDF and OWL not only have simplified the representation of these KBs, but they have also provided a common framework so data can be exchanged between applications without loss of meaning. RDF uses a format of triples which presents the relationship between two nodes, a subject, an object, and a predicate that links them, to tackle this new data Relational Graph Convolutional Network (RGCN) [2] had to be introduced which took into consideration the information that resides on the edges (relationships) of the graph. This makes the data richer, but also increases their complexity: the size of the data makes it difficult to train on it, as it is not obvious to scale it without losing information because this is where the biggest problem of graph data resides. This paper is organized as follows: In Section 2, we provide the essential concepts that will be presented throughout the paper. In Section 3 we introduce the main methods used in the paper that address the topic of scalability and in section 4 the ones of transfer learning. We provide our analysis and insights in Section 5, in section a brief discussion comparing our results to other papers, and the conclusion in Section 6.

2 Preliminaries

In this section, we will recall some key concepts, in this paper, we will consider a directed and labeled multi-graph as $G = (\mathcal{V}, \mathcal{E}, \mathcal{R})$ with nodes (entities) $v_i \in \mathcal{V}$ and labeled edges (relations) $(v_i, r, v_j) \in \mathcal{E}$, where $r \in \mathcal{R}$ is a relation type.

2.1 Relational graph convolutional networks

Introduced first as a GCN by Kipf and Welling [1] as a method to generalize graph data analysis where a layer has the form:

$$h_{v_i}^{(l+1)} = \sigma \left(\sum_j \frac{1}{c_{ij}} h_{v_j}^{(l)} W^{(l)} \right), \tag{1}$$

where j indexes the neighboring nodes of vi. cij is a normalization constant for the edge (vi,vj) which originates from using the symmetrically normalized adjacency matrix $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ which then gets used as an extension for large scale relational data, which modifies the forward pass equation:

$$h_i^{(l+1)} = \sigma \left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{c_{i,r}} W_r^{(l)} h_j^{(l)} + W_0^{(l)} h_i^{(l)} \right), \tag{2}$$

where $\mathcal{N}_i^r$ denotes the set of neighbor indices of node i under relation $r \in \mathcal{R}$. Different from regular GCNs, relation specific transformation gets introduced to take in count the information on the graph edges.

2.2 Graph Autoencoders (GAE)

Recently graph autoencoders [1] and variational graph autoencoders [3] have emerged as powerful models that showed remarkable results for (semi-)supervised and unsupervised learning to learn the graph data characteristics, the proceed by mapping (encoding) the node and edge features of the graph to a low dimensional vector to be able to capture and reconstruct the graph structure which is crucial for the data scalibility [4]. In basis, the point of using GAE is to be able to reconstruct A from Z, where A is the adjacency matrix and Z is the $n \times f$ matrix of all latent space vectors is the output of a Graph Neural Network (GNN) with $\hat{A}$ the reconstruction of the adjacency matrix:

$$\hat{A} = \sigma(ZZ^T) \quad \text{with} \quad Z = \text{GNN(X,A)}. \tag{3}$$

2.3 Entity classification

Entity classification for graph data is an RGCN classifier where its layers (2), where it has a minimum of 2 layers, the classifier takes the nodes representations and predicts the labels, the model is learned by minimizing the cross-entropy loss:

$$\mathcal{L} = -\sum_{i \in \mathcal{Y}} \sum_{k=1}^{K} t_{ik} \ln h_{ik}^{(L)}, \tag{4}$$

where $\mathcal{Y}$ is the set of node indices that have labels and $h_{ik}^{(L)}$ is the k-th entry of the network output for the i-th labeled node. t_{ik} denotes its respective ground-truth label.

3 Graph scalability

In this section, we introduce the concept of querying and summarizing graphs used for the experiment.

3.1 Graph summarization

Data summarization when applied to the machine learning has a fine line between time performance and results accuracy, having the data in RDF is a plus when we want to have the data summarized, a paper, like "Summarizing Semantic Graphs: A Survey" [5] shows a collection of methods by different researchers on RDF summarization, some specific papers like "Simplifying RDF Data for Graph-Based Machine Learning" [6] which shows remarkable performance when it comes to RDF summarization and "A Degeneracy Framework for Scalable Graph Autoencoders" [7] which presents results for speed performance and accuracy performance using graph auto-encoders and GCNs, but the paper that inspired our framework is "Graph Summarisation of Web Data: Data-driven Generation of Structured Representations"[8] which utilises SPARQL queering to obtain results via bi-simulation, where our framework uses the same queering method to summarize the data by node type to get a summary on the node collection layer of the original data [9].

3.2 Type summarization

The Resource Description Framework (RDF) s a simple format of triples that is composed of subjects, predicates, and objects, where each unique node from the collection of the subjects and objects combined represents an actual node from the data graph. Using SPARQL as querying language RDF schema can be filtered out to get from it the types of the nodes, which these types then will become the main nodes of the summarized graph, and what happens in the SPARQL query is that a PREFIX line is put in place to recognize the types and from the extracted unique subject and object types a new graph is constructed which keeps the relationships between the original nodes that become the relationship between the "Type" nodes of the new graph. Accordingly, the size of the original graph will be greatly reduced depending on the diversity of the nodes types in it upon which that diversity will determine the size of the new reduced graph.

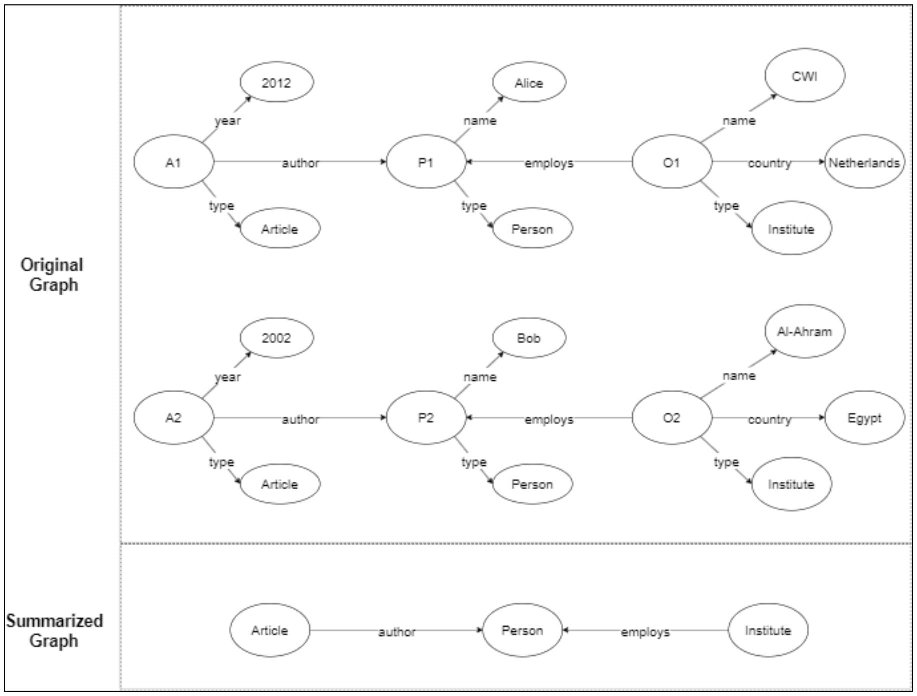

Figure 1: An example of type summarization

4 Transfer learning

The basic idea of transfer learning is to use knowledge learned from a previous source domain to improve the performance of the learned decision functions of the target domain according to the definition in [10]. Transfer learning is really useful to not reduce training time and gain remarkable results from our model by using the best-trained parameters learned from another model, a variety of pre-trained neural networks exist which were trained on a vast amount of data. The ImageNet project for image classification, for example, is one of these pre-trained models which has 14 million images that belong to more than 20,000 classes. All the parameters learned from the source model that was trained on a prior time can be transferred to the target model to gain valuable time and resources, but some criteria still have to be taken into consideration, the similarity of the datasets as well as their size. The most common case of transfer learning of a new dataset is when said dataset is smaller than the original and they are both similar so to avoid overfitting when training the classifying layer is the only layer to be fine-tuned. Another form of transfer learning is when the new dataset is larger than the one of the pre-trained model. In this latter case, we would have more confidence in not overfitting and we could fine-tune through the entire network.

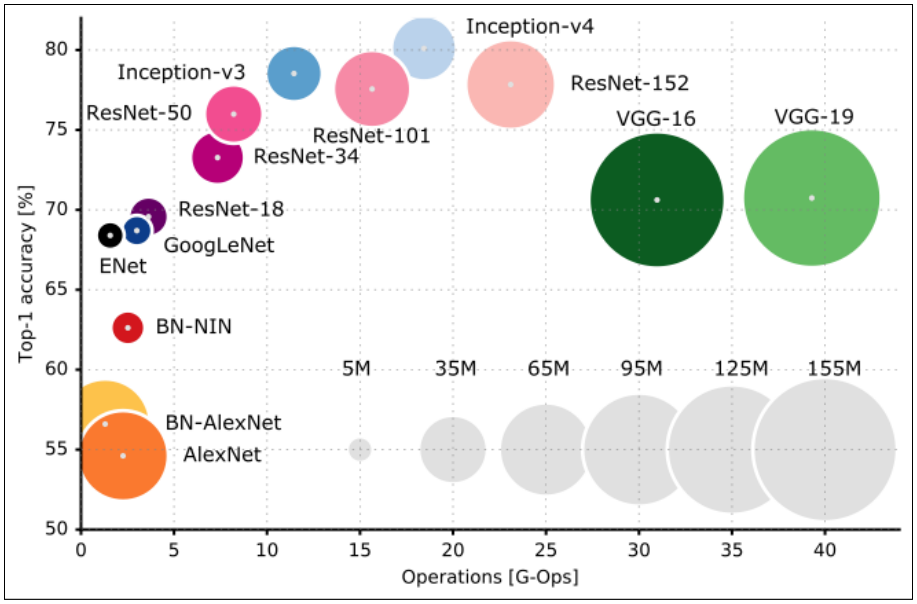

Figure 2: Collection of public NN models, on a scale of the accuracy achieved upon conception, with respect to the dataset size used for training

4.1 Inductive transfer learning

Inductive transfer learning is a form of transfer learning for where the target task is different from the source task, regardless of whether the source and target domains are identical or not [11]. Inductive learning can be categorized into two cases:

a. There is a lot of labeled data in the source domain. The inductive transfer learning setting is analogous to the multitask learning environment in this scenario. Inductive transfer learning, on the other hand, focuses solely on improving performance in the target task by transferring knowledge from the source task, whereas multitask learning tries to learn both the target and source tasks at the same time.

b. No labeled data in the source domain are available. In this case, the inductive transfer learning setting is similar to the self-taught learning setting. In the self-taught learning setting, the label spaces between the source and target domains may be different, which implies the side information of the source domain cannot be used directly. Thus, it's similar to the inductive transfer learning setting where the labeled data in the source domain are unavailable.

This then is used to apply an approach referred to as the feature-representation-transfer approach an example of that is represented in the paper of Blitzer et al. [12]. The intuitive idea behind this case is to learn a "good" feature representation for the target domain. In this case, the knowledge used to transfer across domains is encoded into the learned feature representation. With the new feature representation, the performance of the target task is expected to improve significantly.

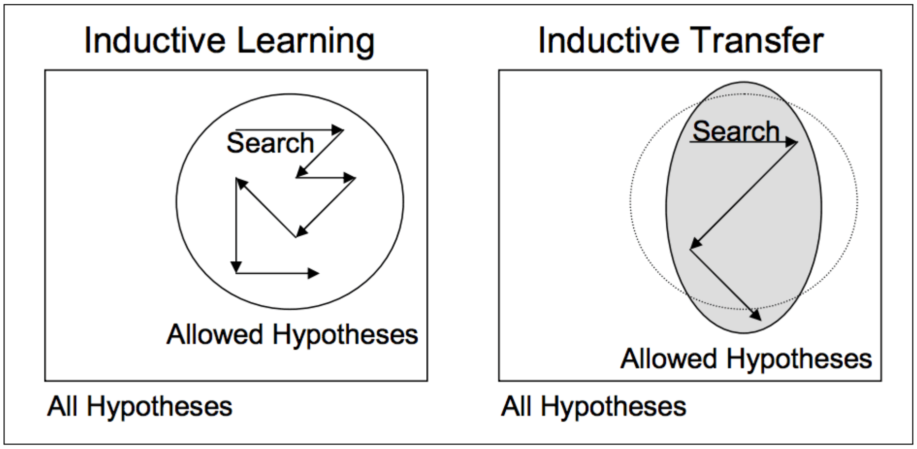

Figure 3: Comparison between Inductive learning and Inductive transfer learning

4.2 How transferable are the features?

If the target dataset is small and the number of parameters is large, fine-tuning may result in overfitting, so the features are often left frozen. On the other hand, if the target dataset is large or the number of parameters is small so that overfitting is not a problem, then the base features can be fine-tuned to the new task to improve performance. Of course, if the target dataset is very large, there would be little need to transfer because the lower-level filters could just be learned from scratch on the target dataset. [13] Typically in transfer learning, the new dataset is larger and similar to the original dataset which prevents overfitting when trying to fine-tune through the full network. In other cases, the new dataset is large and very different from the original. Since the dataset is very large, we may expect that we can afford to train a convolutional network from scratch. In practice, though, it is often still beneficial to start with weights from a trained model. We'd have enough data and confidence in this case to fine-tune the network as a whole.
However, for our problem the new dataset is small and different from the original, it is likely best to only train a linear classifier. Since the dataset is very different, it may not be optimal to train the classifier using the network's top layer, which contains more dataset-specific features. Instead, it might work better to train the classifier from activations somewhere earlier in the network.

5 Empirical evaluation

In this section, we empirically evaluate our model combined with our introduced framework.

5.1 Experimental Setting

Datasets We evaluate our model on four datasets[1] in RDF format: AIFB, MUTAG, BGS, and AM. The AIFB dataset describes the AIFB research institute in terms of its staff, research group, and publications. The dataset was first used to predict the affiliation

[1] https://github.com/rusty1s/pytorch_geometric/blob/master/torch_geometric/datasets/entities.py

(i.e., research group) for people in the dataset. The dataset contains 178 members of a research group, however, the smallest group contains only 4 people, so this one is removed from the dataset, leaving 4 classes. Moreover, we also remove the employees relation, which is the inverse of the affiliation relation from the dataset. The MUTAG dataset contains information about complex molecules that are potentially carcinogenic, which is given by the isMutagenic property. The BGS dataset was created by the British Geological Survey and describes geological measurements in Great Britain. The dataset contains around 150 named rock units with a lithogenesis, from which we used the two largest classes. The AM dataset contains information about artifacts in the Amsterdam Museum. Each artifact in the dataset is linked to other artifacts and details about its production, material, and content. It also has an artifact category, which serves as a prediction target. We have drawn a stratified random sample of 1,000 instances from the complete dataset. We also removed the material relation, since it highly correlates with the artifact category. And for each of these datasets, we use their node-type summarized counterparts for transfer tasks.

Dataset	AIFB	MUTAG	BGS	AM
Entities	8,285	23,644	333,845	1,666,764
Relations	45	23	103	133
Edges	29,043	74,227	916,199	5,988,321
Labeled	176	340	146	1,000
Classes	4	2	2	11

Table 1: Number of entities, relations, edges, and classes along with the number of labeled entities for each of the datasets. *Labeled* denotes the subset of entities that have labels and that are to be classified.

Task We consider the task of entity classification as briefly introduced in section 2 where the model acts as a classifier for the entities of the graph to find their corresponding labels which are the node types. Implementing the steps from Schlichtkrull et al. [2] with a split of 80/20 of the dataset without a validation set, we can directly extract the hyperparameters from the paper and apply them to the model directly. We track the training time (wall-clock time) of the training of the complete datasets, then we apply transfer learning to use the parameters of a pre-trained model which in this case, is a model we train on a different dataset.

	AIFB	MUTAG	BGS	AM
Test accuracy	94.02	70.58	81.46	OOM
Training time	16s	11s	5min53s	OOM

Table 2: Training time on the original graphs and the test accuracy results.

Model The original model has been based on the experiments done in the RGCN modeling paper [2], the RGCN is a basic two-layer classification networking (2). The model is trained on 50 epochs, and the training accuracy is averaged for 10 runs. The model[2] is built using Python, with the usage of Pytorch and Pytorch Geometric libraries to utilize the GPU which has, like specifications, 12.69GB of memory. The training and evaluation were done in the Google COLAB[3] environment. The summarization of the graph was done using Java code run over an Intel Core i7 QuadCore CPU. All data summarized was reshaped by padding the tensor manually to have the same shape as the layer we need to fine-tune for the transfer learning

5.2 Results

As presented before, the training results are in comparison to what other frameworks achieved. In table 3 is the information that will be used as the base of comparison from the summarized versions of the graphs and table 4 are the results we got from the transfer learning.

Dataset	AIFB	MUTAG	BGS	AM
Original size(in MB)	5.95	10.2	209	755
Summarized size (in KB)	26	44.1	5.94	7.79
Reduction of	22884%	23129%	3518518%	9691912%
Summarization time	3s	3s	2min5s	1min57s
Number of nodes	26	112	4	20
Number of edges	188	316	41	56

Table 3: Measured time for type summary and the new size of the dataset memory-wise.

		AIFB	MUTAG	BGS	AM
Save Model from Summarized Data	Test Accuracy	1.6	36.89	64.66	41.14
	Training Time	5s	2s	8s	12s
Model with Transfer Learning	Test Accuracy	48.50	66.08	65.37	OOM
	Training Time	13s	9s	3min58s	OOM

Table 4: Training time on the summarized graphs and the test accuracy results as well as their performances after transfer learning.

We observe a few interesting things about our results in table 4, we first notice slight to no improvement for the time to train both our model to transfer and our original with that model for the smaller datasets, which were the ones with the largest summarized counter parts (AIFB, MUTAG). For the dataset BGS, the transfer learning had the opposite effect, where the summary was smaller and more general, had a decrease in the training time from 5 minutes 53 seconds to 2 minutes 58 seconds. While for AM, the training on the target model still went Out Of Memory, but something we noticed is that training on the

[2] https://github.com/Karim-Anwar/thesis/

[3] https://colab.research.google.com/

summarized was possible.
None of the three successful models had progress for test accuracy but the datasets from the original lowest had a higher test accuracy compared to the AIFB dataset model.
A couple of elements can be observed, first, the model with greater reduction of the datasets for the transfer have improved in training time as well as they had the highest test accuracy, so we can believe that with a better summarization method we could have still had an improvement in time and maybe as well an improvement in the test accuracy.
The paper ”A Degeneracy Framework for Scalable Graph Autoencoders” [7] presents a framework that trains a dense subset of the nodes instead of using the entire graph with Graph (Variational) Auto-encoder. Even if the model uses GCN instead of RGCN, the results are a great comparison point, especially when it comes to the results on large graphs. The closest dataset used in comparing the original dimensions is our MUTAG dataset and their Pubmed dataset. The degeneracy framework results were as follows:

Model	**Size of input**	**Mean Perf. on Test Set (in %)**		**Mean Running Times (in sec.)**				
	***k*-core**	**AUC**	**AP**	*k*-core dec.	Model train	Propagation	**Total**	**Speed gain**
VGAE on $\mathcal{G}$	-	83.02 ± 0.13	$\mathbf{87.55 \pm 0.18}$	-	710.54	-	710.54	-
on 2-core	$9,277 \pm 25$	$\mathbf{83.97 \pm 0.39}$	85.80 ± 0.49	1.35	159.15	0.31	160.81	×4.42
on 3-core	$5,551 \pm 19$	$\mathbf{83.92 \pm 0.44}$	85.49 ± 0.71	1.35	60.12	0.34	61.81	×11.50
on 4-core	$3,269 \pm 30$	82.40 ± 0.66	83.39 ± 0.75	1.35	22.14	0.36	23.85	×29.79
on 5-core	$1,843 \pm 25$	78.31 ± 1.48	79.21 ± 1.64	1.35	7.71	0.36	9.42	×75.43
...	...	...	...	...	...	...	...	...
on 8-core	414 ± 89	67.27 ± 1.65	67.65 ± 2.00	1.35	1.55	0.38	**3.28**	**×216.63**
on 9-core	149 ± 93	61.92 ± 2.88	63.97 ± 2.86	1.35	1.14	0.38	**2.87**	**×247.57**
DeepWalk	-	81.04 ± 0.45	84.04 ± 0.51	-	342.25	-	342.25	-
LINE	-	81.21 ± 0.31	84.60 ± 0.37	-	63.52	-	63.52	-
node2vec	-	81.25 ± 0.26	85.55 ± 0.26	-	48.91	-	48.91	-
Spectral	-	83.14 ± 0.42	86.55 ± 0.41	-	31.71	-	31.71	-

Table 5: Link Prediction on Pubmed graph ($n = 19,717$, $m = 44,338$), using VGAE model, its k-core variants, and baselines

We can observe that the more the k-core increases the greater the speed gain (x247.57), but, in consequence, the mean performance decreases as well. This is explainable since the smaller the subset, the less likely the resemblance in shape to the original. For our datasets, the summarized form of MUTAG would, by mathematical calculation have a multiplier of x231 for speed gain, but then, mean performance would be around 62% which compared to the original test results in table 2 would signify a decrease of ≈10%. If it were for a larger dataset like AM we would gain more positives than negative results in that exchange to start with. The most obvious gain is the speed gain, but, as presented before for how big the size of that dataset, it could not fit in memory and went Out Of Memory (OOM), which, in this case, even if the performance would be terrible, we would have some results from it. That is why, with a combination with transfer learning, the results with the usage of small graphs like these would be interesting since GAE learns the features and structure of the summarized graph, and using these parameters optimize the model to train on the original graph which will demand in overall less in memory and time overall.

6 Conclusion

The issue with graph data is their size and complexity. When relational graph convolution networks and graph autoencoders were introduced to machine learning community, they showed real promise on the improvements that can be done or learned from graph data, but, the same old problem of scalability still resides. With the introduction of frameworks-like the one presented in this paper-of a combination of graph summarization by node type or frameworks like bi-simulation and graph degeneracy summarization started solving the size complexity with regards to retaining information from the original graph data, which would improve greatly the time and memory consumption for these big machine learning models, this opens up a whole new level of efficiency in processing large graphs and guide future researches towards improving them.

Acknowledgments

I would like to thank Mr. Micheal Cochez for the generous time that was given to comment and discuss this paper, I would also like to thank the students that were working alongside me under Mr. M.Cochez, as well as Mr. Mohamed Shahawy a good friend for their input. And special thanks to my cat Luna who kept me going, and helped me avoid a nervous breakdown.

References

[1] Thomas N Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[2] Michael Schlichtkrull et al. “Modeling Relational Data with Graph Convolutional Networks”. In: *arXiv preprint arXiv:1703.06103* (2017).

[3] Thomas N. Kipf and Max Welling. *Variational Graph Auto-Encoders.* 2016. arXiv: 1611.07308 [stat.ML].

[4] Nadezhda Fedorova, Josep Blat, and David F. Nettleton. “Can Embedding Solve Scalability Issues for Mixed-Data Graph Clustering?” In: *Euro-Par 2015: Parallel Processing Workshops.* Ed. by Sascha Hunold et al. Cham: Springer International Publishing, 2015, pp. 481–492. ISBN: 978-3-319-27308-2.

[5] Šejla Čebirić et al. “Summarizing semantic graphs: a survey”. In: *The VLDB Journal* 28.3 (June 2019), pp. 295–327. ISSN: 0949-877X. DOI: 10.1007/s00778-018-0528-3. URL: https://doi.org/10.1007/s00778-018-0528-3.

[6] P. Bloem, A. Wibisono, and G.K.D de Vries. “Simplifying RDF Data for Graph-Based Machine Learning.” English. In: *KNOW@ LOD.* 2014.

[7] Guillaume Salha et al. *A Degeneracy Framework for Scalable Graph Autoencoders.* 2019. arXiv: 1902.08813 [cs.LG].

[8] Stéphane Campinas. “Graph summarisation of web data: data-driven generation of structured representations”. In: 2016.

[9] S. Campinas et al. “Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation”. In: *2012 23rd International Workshop on Database and Expert Systems Applications*. 2012, pp. 261–266.

[10] Fuzhen Zhuang et al. *A Comprehensive Survey on Transfer Learning*. 2019. arXiv: `1911.02685 [cs.LG]`.

[11] Sinno Jialin Pan and Qiang Yang. “A Survey on Transfer Learning”. In: *IEEE Transactions on Knowledge and Data Engineering* 22 (2010), pp. 1345–1359.

[12] John Blitzer, Mark Dredze, and Fernando Pereira. “Biographies, Bollywood, Boomboxes and Blenders: Domain Adaptation for Sentiment Classification”. In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Prague, Czech Republic: Association for Computational Linguistics, June 2007, pp. 440–447. URL: `https://www.aclweb.org/anthology/P07-1056`.

[13] Jason Yosinski et al. *How transferable are features in deep neural networks?* 2014. arXiv: `1411.1792 [cs.LG]`.