Scalable Knowledge Enhancement of Graph Neural Networks

Anonymous Author(s)
Anonymous Affiliation
Anonymous Email

Abstract

Knowledge enhanced neural networks (KENNs) integrate prior knowledge in the form of logical formulae into an artificial neural network by adding knowledge enhancement (KE) layers to the network architecture. Previous results show that the model outperforms pure neural models as well as neural-symbolic models on small graphs but struggle to extend to larger ones. In this paper, we address the problem of knowledge enhancement of neural networks on large graphs and carry out experiments on the Open Graph Benchmark datasets (OGB). When dealing with large graphs, we show that neighbourhood explosion occurs and makes the full-batch training of the model unfeasible. To solve this problem, we first analyse the space complexity of the knowledge enhancement layers and propose a graph-specific mini-batching strategy to make it applicable to large-scale graphs. To show that our method is effective, we test our model on two datasets from the Open Graph Benchmark. To the best of our knowledge, this is the first approach that makes neural-symbolic methods such as KENN feasible to large graphs of arbitrary size and the first approach that uses datasets from OGB in a neural-symbolic context.

1 Introduction

Recently, remarkable progress has been made in various research domains thanks to deep learning and the application of artificial neural networks (NNs). The major strength of NNs is their ability to extract meaningful features from high-dimensional data without any expert knowledge. Despite their success, deep learning models are often criticised for their shortcomings in terms of interpretability, accountability and data-hungriness [1]. While deep learning approaches are mostly data-driven, symbolic AI models carry out logic-like reasoning steps over language-like representations such as relational data or graphs while being more data efficient and understandable by humans [2].

In order to address the limitations of deep learning, the research field of neural-symbolic integration has emerged with the aim to combine neural approaches with methods from symbolic AI. The integration of sub-symbolic (neural) approaches and symbolic approaches has a large potential for the future of AI. Neural-symbolic AI not only paves the way towards the application of AI to situations with limited data. Also, neural-symbolic integration allows to jointly use different sources of information. This leads on the one hand to richer models and a wider range of applications of AI. On the other hand, it helps to tackle the black-box property of deep learning and avoid undesired effects (such as discrimination of social groups for example) by exploiting symbolic representations. Neural-symbolic integration research has the potential to improve the accuracy, data efficiency, and interpretability of the current state of the art in AI which has the potential to lead to new insights in a wide range of research domains. [1] [3] [4].

Knowledge enhanced neural networks (KENN) [5] [6] is a neural-symbolic approach that stacks knowledge enhancement layers (KE layers) as additional layers on top of a NN. These layers modify the outputs of the so-called base neural network (base NN) with respect to some prior knowledge in the form of first-order logic (FOL) formulae. Both, the KE layers as well as the base NN are fully differentiable and are optimised jointly with backpropagation. The KE layers contain learnable *clause weights* that are modified during training and allow the model to be robust to wrong knowledge by setting the respective weights to zero. Furthermore, the clause weights indicate the importance of each clause which increases interpretability. [7]

While KENN was initially developed for multi-label classification [5], an extension of the model to relational domains has been proposed [7]. In order to evaluate the performance of KENN on relational data, the model is applied to a node classification task on the Citeseer dataset [8] where the goal is categorise scientific papers into topics. KENN allows to formulate assumptions on the papers (nodes) and citations (edges) in the graph as logical formulae in FOL and use them jointly with the numeric node features to solve the node classification task. Intuitively, when data is scarce the injection of the prior logic formulae is expected to be helpful guiding the training process. The first experiments with KENN show that it represents a promising neural-symbolic approach with an enormous potential for extension. The current scope of the results is however limited. The Citeseer dataset used in KENN [7] is rather small. Furthermore, no consensus on a split of the data in train, valid and test set exists, which makes it difficult to compare results across different approaches. In particular, a sufficiently large and informative dataset is needed to test complex models [9]. The Open Graph Benchmark (OGB) [9] is a collection of large-scale datasets from diverse domains designed for testing predictive models on large graphs. In addition to the datasets, baseline models as well as preprocessing and evaluation protocols are provided to facilitate comparability.

In order to test KE layers in the context of large-scale datasets, we apply the model to the OGB datasets ogbn-arxiv and ogbn-products for node classification. For such datasets, the full-batch training of models can become unfeasible because the required memory capacity is large. Standard mini-batch stochastic gradient (SGD) is not directly applicable to graphs. Nodes are connected by edges and consequently not independent. Thus, the division into batches must ensure that the relevant information (such as a node and its neighbourhood) is available per batch. Furthermore, the number of required neighbours grows exponentially with the number of stacked relational KE layers and can exceed the memory capacity. This problem is referred as *neighbourhood explosion* in graph neural network domain [10]. For GNNs, various sampling methods [11] have been proposed to solve this problem.

In this work, we study relational knowledge enhancement of neural networks for large graphs. In particular, we first analyse the space complexity of relational knowledge enhancement layers to identify the bottlenecks for scalability. In order to make knowledge enhancement feasible, we introduce a subgraph-oriented mini-batching method, inspired from GraphSAGE [12], which allows to control the neighbourhood explosion problem. We designed a prototype which allows to introduce KE layers on graph convolutional networks (GCNs) [13]. We show that our method is effective in node classification tasks applied to datasets from the Open Graph Benchmark (OGB) [9]. While most neural-symbolic approaches such as KENN are applied to small datasets or toy examples, this is to the best of our knowledge one of the first approaches that tackles the scalability challenges with large graphs and combines graph-specific mini-batching techniques to neural-symbolic methods. Furthermore, to our knowledge, this is the first neural-symbolic approach applied to OGB. We thereby present a way to address the lack of benchmark datasets in the neural-symbolic domain [4] [14] [7].

Outline In Section 2 we present the architecture of KE layers and their adaption to relational data. In Section 3 we formulate the neighbourhood explosion problem for KE layers and propose restrictive neighbourhood sampling as mini-batching method in Section 4. We present our experiments in Section 5. In Section 6 gives a perspective on future work.

2 Knowledge Enhanced Neural Networks

Knowledge enhanced neural networks (KENN) [5] is a neural-symbolic approach that was recently extended to relational domains [7]. At its core, the KENN consists of two components that together form an end-to-end differentiable model. A base NN produces predictions and the knowledge enhancement layers refine these predictions based on a prior knowledge. The KENN takes two types of inputs: (1) graph-structured or relational data converted to a numerical form and (2) prior knowledge expressed as first-order logic (FOL) formulae.

2.1 Graph Data

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$ connecting the nodes $v_i, v_j \in \mathcal{V}$, $(v_i, v_j) \in \mathcal{E}$. The adjacency matrix $\mathbf{A} = [\mathbf{A}_{ij}]$ with $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}|x|\mathcal{V}|}$ describes edges between nodes. Each node v_i is attributed with a feature vector $h_i^{(0)} \in \mathbb{R}^{d_0}$ and a ground truth label vector $y_i \in \mathbb{R}^m$. In matrix notation, the feature matrix is $H^{(0)} \in \mathbb{R}^{n \times d_0}$ and $y \in \mathbb{R}^{n \times m}$.

2.2 Prior Knowledge

In addition to $\mathcal{G}$, some prior knowledge $\mathcal{K}$ is defined. It is specified as a set of $|\mathcal{K}|$ FOL clauses $\mathcal{K} = \{c_1, ..., c_{|\mathcal{K}|}\}$ as disjunctions of literals.

$$c_i = \bigvee_{j=1}^{k_{c_i}} l_j \tag{1}$$

Clause $c_i \in \mathcal{K}$ contains k_{c_i} literals. In our setting, the clauses refer to the m prediction classes. The logical language contains constants, variables and predicates. Unary predicates express properties of constants, while binary predicates describe relations between two constants.

To give an example from the Citeseer dataset [7], let the unary predicate $AI(x)$ describe that a scientific paper belongs to the document class AI (from the artificial intelligence domain) and let the binary predicate $Cite(x, y)$ denote a citation between two papers x and y. The clause

$$\forall x, y : AI(x) \wedge Cite(x, y) \longrightarrow AI(y) \tag{2}$$

describes that two papers belong to the same class AI if they cite each other. In this example, the intuition of KENN will be to classify papers into the same category, if they cite each other. At the present, the logical language of KENN does not explicitly represent quantifiers and assumes all clauses to be all-quantified. The implementation of fuzzy logic operators for $\forall$ and $\exists$ quantifiers as presented in [15], for example, is a future work.

So far no uniform procedure for acquiring prior knowledge has been proposed and numerous ways are conceivable. Depending on the application, expert knowledge or general knowledge can be manually curated [5] or derived from widely available sources such as ontologies. Clauses can also be obtained from data, for example in the form of association rules [16]. In addition, assumptions can be formulated , which do not necessarily have to be fully satisfied.

2.3 Knowledge Enhancement Architecture

A brief overview of the KENN architecture given in the following. The base NN takes the input features $H^{(0)}$, transforms them in its hidden layers and produces preactivations $\mathbf{z} \in \mathbb{R}^{nxm}$ in the last layer. In principal, any kind of NN that implements a function $f : H^{(0)} \longrightarrow \mathbf{z}$ can be employed as a base NN. In this work, we consider a MLP [9] and a GCN [13] as base NN. The base NN in this experiments performs a node classification task and assigns document classes to each paper. Consequently, the output dimension of the base NN corresponds to the number of predicted document classes.

$$\mathbf{z} = f_{\text{MLP}}(H^{(0)}, \theta_{\text{MLP}})$$
$$\mathbf{z} = f_{\text{GCN}}(H^{(0)}, \mathbf{A}, \theta_{\text{GCN}})$$

Both models contain learnable parameters θ that are optimised during training.

Several KE layers can be stacked on top of each other. Each KE layer updates the preactivations $\mathbf{z}$ of the base NN with respect to the satisfaction of $\mathcal{K}$ and applies a non-linear activation function $\sigma(\mathbf{z}) = \hat{\mathbf{y}}$ in the last layer to obtain the final predictions. Each KE layer contains *clause weights* that are trained jointly with the parameters of the base NN. The clause weight w_c of a clause c corresponds to the importance of a specific clause on the output. It is robust to wrong knowledge and can be set to zero.

To be incorporated in a neural architecture, the knowledge has to be interpreted in a real-valued domain. Fuzzy logic [3] is applied to obtain truth values in a continuous interval of $[0, 1]$. The interpretation of the logic language in the real-valued domain is called *grounding*. Here, constants and variables are grounded to real-valued vectors and predicates to functions that project an input to real values between zero and one [4]. The preactivations $\mathbf{z}$ of the base NN are used as a numeric interpretation of unary predicates. T-conorm functions [17] map the truth values of grounded atoms to the truth value of a clause. To quantify the improvement of the satisfaction of a clause, the following *t-conorm boost function (TBF)* updates the initial predictions of the base NN.

$$\delta_s^{w_c}(\mathbf{z}) = w_c \cdot \text{softmax}(\mathbf{z}) \tag{3}$$

A module called *clause enhancer (CE)* implements the TBF for each clause. The changes introduced by all CEs are aggregated, added to $\mathbf{z}$ and given to the activation function. Various groundings of a clause to constants are stored in a matrix where columns represent the predicates and rows the constants. Once instantiated, the CE works on several groundings G of a clause. More details on KENN can be found in [7] and [5].

2.4 Knowledge Enhancement on Relational Data

In case of relational data, not only unary but also binary predicates are considered. Some changes are made to the model in order to support relational data [7]. While the groundings of unary predicates can be represented as a matrix that contains the objects as rows and grounded predicates as columns, the keys of the binary predicates are two-dimensional. Consequently, binary predicates are represented as a matrix $\mathbf{B}$ that has as many rows as groundings of binary predicates and columns as unary predicates. To enhance clauses that contain binary predicates, all grounded predicates have to be presented together in one matrix on which the CE can operate. Hence, unary predicates are "binarized" by ignoring one component of the input.

Given a unary predicate $P(x)$ for example, it can be extended to two binary predicates $P^X(x, y)$ and $P^Y(x, y)$. Considering the clause $c : \neg AI(x) \vee \neg Cite(x, y) \vee AI(y)$ of Section 2.2 and the two groundings $c[x/a, y/b]$ and $c[x/b, y/c]$, the preactivation of both constants $\mathbf{z}_{AI}(a)$ and $\mathbf{z}_{AI}(b)$ have to be represented in the same matrix in order to calculate the enhancements. Consequently, the relational KE layer contains a join operator that joins binary predicates and the binarized unary predicates into one matrix $\mathbf{M}$. After obtaining the changes $\delta\mathbf{M}$, a group-by layer collects the changes that apply to the same grounded propositional variable and aggregates them. The accumulated changes can then be added to the preactivations of the previous layer.

To give an example, a unary clause $c : P_1(x) \vee P_2(x)$ with $P_1, P_2 \in \mathcal{K}_U$ can have several predicates that refer to only one variable. As a consequence, the updates given by Equation 3 for a node v_i would be

$$\delta_s^{w_c}(z_{(i,1)}) = \frac{e^{z_{(i,1)}}}{e^{z_{(i,1)}} + e^{z_{(i,2)}}} \tag{4}$$

In comparison, a binary clause contains unary *and* binary predicates and can refer to two different variables, for example: $c = P_1(x) \wedge P_2(y) \wedge P_3(x, y)$ where $P_1, P_2 \in \mathcal{K}_U$ and $P_3 \in \mathcal{K}_B$. In this case, the enhancement for v_i of that clause (defined in Equation 3) is given by the following term

$$\delta_s^{w_c}(z_{(i,1)}) = \frac{e^{z_{(i,1)}}}{e^{z_{(i,1)}} + e^{z_{(j,2)}} + e^{z_{3(i,j)}}} \tag{5}$$

where $e^{z_{3(i,j)}}$ represents the preactivation of the binary predicate $P_3(x, y)$ for the grounding $P_3(i, j)$. In this setting [7], the citations between the papers are assumed to be known apriori and complete. Consequently, the groundings of the binary predicate Cite are set to a high positive value.

In summary, for graph data the changes applied to a grounded predicate depends not only on the grounding of a constant but also on the groundings of constants related by binary predicates. In other words, not only the representation of a node itself from the previous layer is needed, but also the representation of neighbouring nodes to which the node is connected to by an edge.

3 Formulation of the Scalability Challenges for Large Graphs

The processing of large graphs is not only time-consuming but also demanding in terms of space requirements, particularly when GPUs are used where the memory capacity is limited. In full-batch training the input features $H^{(0)} \in \mathbb{R}^{n \times d_0}$ and the network parameters θ for $\mathbf{L}$ layers of a NN need to be stored. This results in a space complexity of $\mathcal{O}(nd_0\mathbf{L} + d_0\mathbf{L})$. It depends on the size of the dataset n and leads rapidly to infeasibility when n is large.

Mini-batch stochastic gradient descent (SGD)[10] is a widely used solution to this problem. With a batch size of $b << n$, the space complexity can be reduced to $\mathcal{O}(bd_0\mathbf{L} + d_0\mathbf{L})$ since the feature matrices for one batch are only of size $\mathbb{R}^{b \times d_0}$.

3.1 Space Complexity in the Presence of Relational Data

For relational knowledge enhancement, the unary predicates of linked nodes need to be encoded in the matrix $\mathbf{M}$, as described in Section 2. Depending on the number of unary and binary predicates

$|\mathcal{P}_U|$ and $|\mathcal{P}_B|$ and the number of nodes $|\mathcal{V}|$, the size of $\mathbf{M}$ results in $\mathbb{R}^{|\mathcal{V}|^2 \times (2 \cdot |\mathcal{P}_U| + |\mathcal{P}_B|)}$. Here, $|\mathcal{V}|^2$ considers all possible combinations of nodes in the graph to model the grounding $False$ of a binary predicate. This would be the case if *no* edge between two nodes exists. Consequently, a KE layer in full-batch training has the space complexity $\mathcal{O}(|\mathcal{V}|^2 \cdot (2|P_U| + |P_B|) + |\mathcal{K}|)$ since $\mathbf{M}$ and the $|\mathcal{K}|$ clause weights need to be stored. Depending on the formulation of the prior knowledge, the number of considered edges in $\mathbf{M}$ can be reduced. Here, all clauses have the structure $Class(x) \vee \neg Class(y) \vee \neg Cite(x, y)$. The clause is satisfied for all groundings in which $Cite$ equals $False$, independently of the other predicates. Since this does not add additional information to the model, pairs of nodes that are *not* linked by the binary predicate $Cite$ are not included in $\mathbf{M}$. This reduces its dimension to $\mathbb{R}^{|\mathcal{E}| \times (2 \cdot |\mathcal{P}_U| + |\mathcal{P}_B|)}$ and the space complexity drop to $\mathcal{O}(|\mathcal{E}| \cdot (2|P_U| + |P_B|) + |\mathcal{K}|)$.

As for non-relational data, splitting the node set $\mathcal{V}$ into mini-batches of size $b << n$ reduces the size of $\mathbf{M}$ per batch to $\mathbb{R}^{b^2 \times (2 \cdot |\mathcal{P}_U| + |\mathcal{P}_B|)}$. The difficulty with graphs is their connectivity since linked nodes cannot be assumed to be independent. When a graph is split straightforwardly into mini-batches as with non-relational data, some node features that are required to join matrix $\mathbf{M}$ might belong to a different mini-batch and consequently be ignored. In that case, relevant information may be lost resulting in a poor learning process.

Instead of batching the node set, the entire input graph $\mathcal{G}$ can be split into subgraphs $\mathcal{G}_1(\mathcal{V}_1, \mathcal{E}_1), \ldots \mathcal{G}_S(\mathcal{V}_S, \mathcal{E}_S)$. The subgraphs can be used as mini-batches to estimate the loss and calculate the gradients for the full graph. The amount of nodes needed for the KE updates depends on the number $\mathbf{L}_{KE}$ of KE layers stacked. The first KE layer takes the preactivations $\mathbf{z}$ of the base NN as input. The enhancement of the preactivation $\mathbf{z}_i$ of node v_i by a binary clause requires the preactivations $\mathbf{z}_j$ of the constants that co-appear in the groundings of the binary clause. In the graph, the groundings correspond to the preactivations of the first-order neighbourhood $\mathcal{N}^1(v_i)$ of v_i.

When stacking a multiple KE layers, the relational enhancement recursively depends on the outputs of the previous KE layer. As a result, the enhancement of one node with $\mathbf{L}_{KE}$ KE layers requires the $\mathbf{L}_{KE}$-step neighbourhood ($\mathcal{N}^{\mathbf{L}_{KE}}(v_i)$) of this node in the graph. Its size depends not only on $\mathbf{L}_{KE}$ but also on the connectivity of the graph. In the worst case, the required neighbourhood can result in the full graph $\mathcal{N}(v_i) = \mathcal{V}$. In conclusion, the number of required neighbours grows exponentially with $\mathbf{L}_{KE}$.

$$\mathcal{O}\Big((|\mathcal{V}_s| \cdot deg^{\mathbf{L}_{KE}})^2 \cdot (2|P_U| + |P_B|) + \mathbf{L}_{KE} \cdot |\mathcal{K}|\Big) \tag{6}$$

The total space complexity of KENN (in conjunction with MLP as base NN) results in the following space complexity.

$$\mathcal{O}\Big((|\mathcal{V}_s| \cdot deg^{\mathbf{L}_{KE}})^2 \cdot (2|P_U| + |P_B|) + |\mathcal{V}_S| L d_0 + L d^2 + \mathbf{L}_{KE} \cdot |\mathcal{K}|\Big) \tag{7}$$

In the context of graph neural networks (GNN) [10], the exponential growth of the neighbourhood is referred as *neighbourhood explosion* [18]. Advanced mini-batching methods [11] have been proposed to make the application of GNNs to large graphs feasible. Since relational KE layers have structural analogies and are exposed to the same scalability challenges as GNNs, GNN-specific mini-batching techniques can be considered for relational KE layers.

4 Advanced Mini-Batching for Relational Knowledge Enhancement

In order to split a graph into subgraphs for mini-batch training, a trade-off between the subgraph size and the graph information loss has to be found. On the one hand, the subgraphs must be small enough to fit the memory requirements. On the other hand, a sufficient number of nodes and edges are requisite to approximate the training on the full graph.

In this work, we propose *restrictive neighbourhood sampling* (RNS) inspired from GraphSAGE [12] [10]. While GraphSAGE samples neighbours per GNN layer, RNS samples complete subgraphs at the preprocessing stage. The number of neighbours per node is constrained by some parameters. They have to be chosen carefully in accordance with the available memory capacity and the topology of the graph. In the following, we present RNS (see also Algorithm 2) and analyse the resulting space complexity.

RNS introduces a set of hyperparameters. The *batch size* b defines how many nodes to use as target nodes per subgraph. If b is chosen too large, the sampled graph might still exceed the available space resources. The *sampling depth* $\mathbf{l}$ defines the depth of the neighbourhood taken into account. $\mathbf{l}$ should correspond to the number of relational KE layers. The *s*ampling neighbour size n_s defines how many neighbours are sampled per sampling depth. In the first step, the node set $\mathcal{V}$ is split into batches $\mathcal{V}_s$ of batch size b regardless of the edges. We obtain $\lfloor \frac{n}{b} \rfloor + 1$ batches. The last batch might have less than b nodes. We refer to these nodes as target nodes for each batch $\mathcal{V}_s^0$. In the following, n_s first-order neighbours $\mathcal{N}^1(\mathcal{V}_s^0)$ are sampled for the target nodes of each batch and appended to form the updated node set of the total batch $\mathcal{V}_s^0 \cup \mathcal{V}_s^1 \setminus \{v_i | v_i \in \mathcal{V}_s^0 \cap \mathcal{V}_s^1\}$. Duplicate nodes are removed when a node is target and neighbouring node at the same time. The edges between the target nodes and all sampled neighbours are kept so that each batch correspond to a subgraph $\mathcal{G}_s = (\mathcal{V}_s, \mathcal{E}_s)$ with $s \in \{1, \ldots, \lfloor \frac{n}{b} \rfloor + 1\}$. To sample the second-order neighbours the algorithm iterates through the set of first-order neighbours. For each node in the set of second order neighbours,again n neighbours are sampled and appended. This procedure is repeated recursively until the sampling depth $\mathbf{l}$ is reached.

A forward pass with KENN (see Algorithm 1) iterates through the subgraphs $\mathcal{G}_s$ and returns preactivations for all nodes in the batch. The preactivations of the neighbours are needed for the knowledge enhancement are not included in the calculation of the batch loss. If one node appears multiple times in several batches and contribute to the loss more than once, bias would be introduced. Only the preactivations of target nodes are used for the loss calculation per batch.

4.1 Space Complexity for Restrictive Neighbourhood Sampling

The space complexity of relational knowledge enhancement in conjunction with RNS depends on the parameter choice for $\mathbf{l}$, b, n_s and the graph topology. The node degree $deg(v_i)$ denotes the number of neighbours. For simplicity, we assume in the following notations that every node v_i has the same degree deg. For a subgraph $\mathcal{G}_s$, the edges and the node features need to be stored. The edges $\mathcal{E}_s$ are defined as a two-dimensional index vector and can be neglected in the following. The space of the node features depends on the feature dimension d_0 and the sampled nodes in the batch $\mathcal{V}_s$. Since the feature dimension is assumed to be constant, the size of the node set is the critical component.

The set of the nodes in a batch is composed of the target nodes and the sampled neighbours without duplicates.

$$\mathcal{V}_s^0 \sqcup \mathcal{N}_s^1(\mathcal{V}_s^0) \sqcup \ldots \sqcup \mathcal{N}_s^{\mathbf{l}}(\mathcal{V}_s^0) = \bigsqcup_{\ell}^{\mathbf{l}} \mathcal{N}_s^{\ell}(\mathcal{V}_s) \tag{8}$$

If the complete neighbourhood per node is sampled, the total number of nodes in the batch is defined as follows.

$$\begin{aligned} |\mathcal{V}_s| &= |\mathcal{V}_s^0| + |\mathcal{N}_s^1(\mathcal{V}_s^0)| + \ldots + |\mathcal{N}_s^{\mathbf{l}}(\mathcal{V}_s^0)| \\ &= b \cdot (1 + deg + deg^2 + \ldots + deg^{\mathbf{l}}). \end{aligned} \tag{9}$$

When n_s neighbours are randomly sampled from the l-order neighbourhood of each node, 9 is changed to

$$|\mathcal{V}_s| = b \cdot (1 + n_s + n_s^2 + \ldots + n_s^{\mathbf{l}}) \tag{10}$$

By setting the parameters n_s, $\mathbf{l}$ and b to constant values, the exponential growth of the sampled neighbourhood problem can be controlled and adjusted to fit the memory requirements.

4.2 Information Loss with Restrictive Neighbourhood Sampling

In comparison to the sampling of all relevant neighbours, RNS introduces the risk of loosing the information of nodes that are not sampled. This information loss can be quantified. To avoid that the batches get large when the graph is densely connected and deg is high, the sampling size n_s restricts the neighbourhood size. We denote the information loss ω per sampling depth $\ell \in \{1 \ldots \mathbf{l}\}$ for node $v_i \in \mathcal{V}_s^{\ell}$ and its sampled neighbourhood $\mathcal{N}_s^{\ell}(v_i)$ as

$$\omega(v_i, l) = \begin{cases} deg(v_i) - n_s & \text{if } deg(v_i) > n_s \\ 0 & \text{else} \end{cases} \tag{11}$$

Overall, the information loss $\Omega(\mathcal{V}, \mathbf{l}, n_s, b)$ of the sampling process for all nodes and layers can be combined to

$$\Omega(\mathcal{V}, \mathbf{l}, n_s, b) = (deg - n_s) + (deg^2 - n_s^2) + \ldots + (deg^{\mathbf{l}} - n_s^{\mathbf{l}}) \tag{12}$$

When the sampling size is fixed and $deg > n_s$, we can see that the information loss grows exponentially with the constraint sampling depth.

4.3 Redundant Computations for Sampled Neighbours

As defined in Algorithm 1, all preactivations of the neighbours are required to compute the preactivations of the target nodes. This means that the calculation of the preactivation of a node found in multiple batches is redundant. In particular, if the base NN is non-relational, the calculation of the output of the base NN will be reduplicated. How often the representation of a node is recalculated depends on the number of batches that include the node as a neighbour. The likelihood that a node will be sampled depends on the degree of the node itself, the degree of the neighbouring nodes and the choice of the sampling size. A way to increase the computational efficiency of KE layers would be to calculate the preactivations of the base NN in advance and join them for the respective batch, when required. The implementation of this extension is a future work.

5 Experiments

To the best of our knowledge, the relational KE layers have only been applied to the Citeseer dataset [19] [7]. Even though Citeseer is a frequently used citation graph, it is unsatisfactory in terms of quality and quantity [9]. It is not appropriate when complex NN models are used in conjunction with large-scale datasets. A common problem in neural-symbolic integration is the lack of appropriate benchmarks with significant datasets and prior knowledge. In principle, KE layers can be stacked on any kind of neural network architecture [7]. But so far, only results of KENN in conjunction with a MLP have been reported [7].

In this work we extend the knowledge enhancement to large-scale graphs and show that it performs well on the ogbn-arxiv and obgn-products datasets from the Open Graph Benchmark [9]. This extension is based on a careful analysis of the space requirements for an knowledge enhanced model to make it scale. To this end, we stack relational KE layers on top of a graph convolutional neural network (GCN) [13] as base NN and analyse how a mini-batching strategy can be designed to enhance a GCN with prior knowledge. We show that OGB can not only be useful as a benchmark for GNNs but also for approaches from the neural-symbolic domain.

We compare the performance of the four models GCN, MLP, GCN with KE layers (KE_GCN) and MLP with KE layers (KE_MLP) on the two datasets ogbn-arxiv and ogbn-products. OGB [9] provides challenging, diverse and realistic benchmarks to check for the scalability, the robustness and the reproducibility of machine learning models. It provides a unified evaluation protocol, application-specific splits, evaluation metrics and an automated end-to-end pipeline that makes it easy to compare state-of-the-art models. Both datasets have node features and homogeneous edges which makes them suitable for a node classification task. ogbn-arxiv is a citation graph and ogbn-products a purchase network. For more details on the datasets, see Section A.1.1 and A.2 of the Appendix.

The prior knowledge required for the KE layers is derived manually as in [7]. The hypothesis is made that two documents belong to the same class when they cite each other. In context of ogbn-products, two products are supposed to belong to the same class if they are purchased together. Forty clauses are instantiated for each document class according to the following schema for ogbn-arxiv.

$$\forall x \forall y : \neg Class(x) \vee \neg Cite(x;y) \vee Class(y)$$

For ogbn-products the clauses are derived in the same way.

$$\forall x \forall y : \neg Class(x) \vee \neg CoPurchased(x;y) \vee Class(y)$$

As in [7] and as already described above, the edges and binary predicates are assumed to be known apriori. For this reason, the preactivation of the binary predicate $Cite$ (or $CoPurchased$) is set to a high value. To reduce complexity, only pairs of nodes that are connected by edges are considered. For this knowledge, all pairs of nodes for which the binary predicate is $False$ would be directly satisfied and do not add any information. In our experiments, the clause weights are initialized with a constant value (0.5).

We build the models GCN and MLP according to the proposed architecture in [9]. For KE_MLP and KE_GCN we stack KE layers on the base NNs to modify the predictions. The MLP and GCN consist of three hidden layers with hidden dimension of 256, batch normalisation layers [20] and relu activation after each hidden layer. In MLP, the hidden layers are linear layers [21] and in GCN graph convolutional layers [22]. For all models, the logarithmic softmax function [21] is used as activation of the last layer.

$$\operatorname{LogSoftmax}(x_i) = \log\left(\frac{\exp(x_i)}{\sum_j \exp(x_j)}\right) \tag{13}$$

The loss is calculated by the negative log-likelihood function [21] which is the standard loss function for multi-class classification.

$$l(\theta) = -\sum_{i=1}^{n}\left(y_i \log \hat{y}_{\theta,i} + (1-y_i)\log\left(1-\hat{y}_{\theta,i}\right)\right) \tag{14}$$

The performance of the models is evaluated using the accuracy. We compare the models on both datasets in mini-batch SGD with RNS. All experiments are performed in transductive training mode. The edges between nodes from different subsets are also sampled in the training, but only the labels of the training nodes are available. The details on the hyperparameters used in the models are given in Section A.5 of the Appendix.

5.1 Implementation

Our implementation and experiments are publicly available on GitLab[1]. We use PyTorch [21] and modules from the graph learning library PyTorch Geometric [22]. The Weights and Biases application [23] is used to monitor the experiments. For our computations we use a machine running an Ubuntu 20.4, equipped with an Intel(R) Xeon(R) Silver 4114 CPU 2.20GHz processor, 192G of RAM and Nvidia GPU Quadro P5000.

5.2 Results

We tested the knowledge enhancement of an MLP and a GCN on the datasets ogbn-arxiv and ogbn-products with RNS. The results on ogbn-products can be found in Table 1 and on ogbn-arxiv in Table 2. While the full-batch training for ogbn-arxiv is still feasible, full-batch training on ogbn-products results in an out-of-memory error (see Table 3 and 4 in the appendix). For both datasets, KE_MLP significantly outperforms the MLP. (P-values of one-sided t-test[2]: $7.21e-17$ on ogbn-products and $1.37e-06$ for ogbn-arxiv). In case of KE_GCN no major improvement can be denoted. (P-values of two-sided t-test[3]: 0.2493 on ogbn-products and 0.2277 for ogbn-arxiv.) The p-values do also not allow to reject the hypothesis of identical performance.

Overall, our results confirm the results obtained by [7] on the Citeseer dataset where KENN in conjunction with an MLP outperforms the MLP. In this work we obtain the results at scale. For the knowledge enhancement of GCN, the techniques introduced in this work allow also the algorithms to scale for large datasets. However, no significant prediction improvement has been observed yet. Though, the algorithms introduced here are at least feasible in the sense that they avoid neighbourhood explosions. Several hypotheses support this observation. The GCN can handle relational information and is therefore a more complex model compared to the MLP which only relies on node features. Consequently, a smaller performance gain is expected from the knowledge enhancement of a GCN in comparison to a MLP. Furthermore, each KE layer introduces clause weights as learnable parameters. In the case of ogbn-arxiv 40 logical formulae are yielded to be satisfied. With three KENN layers this leads to 120 additional training parameters. Under these circumstances overfitting might occur. Furthermore the prior knowledge which is only an assumption concerning the relationship between document class and citations. If this relationship is not present in the dataset, the KE layer might introduce additional noise. In order to better investigate the conjunction of graph neural networks with KE layers, further experiments with different sets of logical formulae and other datasets are part of a further work.

[1] `https://gitlab.inria.fr/tyrex/scalable_ke`

[2] One-sided t-test: $H_0 : \mu_{\text{MLP}} == \mu_{\text{KE\_MLP}}$, $H_1 : \mu_{\text{MLP}} < \mu_{\text{KE\_MLP}}$

[3] Two-sided t-test: $H_0 : \mu_{\text{GCN}} == \mu_{\text{KE\_GCN}}$, $H_1 : \mu_{\text{GCN}} \neq \mu_{\text{KE\_GCN}}$

	RNS on ogbn-arxiv			
	avg train accuracy	avg validation accuracy	avg test accuracy (stdv)	avg epoch time
MLP	0.5515	0.5370	0.5206 (0.0314)	0.09
KE_MLP	0.6395	0.5950	0.5701 (0.0067)	2.77
GCN	0.6306	0.5925	0.5473 (0.0071)	1.02
KE_GCN	0.6150	0.5781	0.5373 (0.0242)	2.94

Table 1: Results of RNS training on ogbn-arxiv: Parameter setting in Table 6, SectionA.5).

	RNS on ogbn-products			
	avg train accuracy	avg validation accuracy	avg test accuracy (stdv)	avg epoch time
MLP	0.7740	0.7433	0.5970 (0.0039)	4.17
KE_MLP	0.8250	0.7988	0.6416 (0.0029)	6.5
GCN	0.8835	0.8786	0.7224 (0.0051)	4.13
KE_GCN	0.8807	0.8761	0.7144 (0.0041)	6.78

Table 2: Results of RNS training on ogbn-products. Parameter setting in Table 7, Section A.5.

6 Conclusion and Future Work

In this paper, we studied how to achieve relational knowledge enhancement for large graphs. We first analysed the space complexity of relational knowledge enhancement in order to avoid the neighbourhood explosion introduced by the stacking of multiple enhancement layers. Specifically, we introduced restrictive neighbourhood sampling which allows to control the space requirements of mini-batches. We built an implementation which allows to apply relational knowledge enhancement layers to various GNN models. We show that our method is effective to enhance the predictions of the compared base NN MLP for the datasets ogbn-arxiv and ogbn-products on a multi-class node classification task. This work is to the best of our knowledge the first application of KENN to a large-scale benchmark from the graph neural network domain and shows that the OGB benchmark is useful in a neural-symbolic context and might help to fill the lack of benchmark datasets in the neural-symbolic domain. As explained earlier, for the enhancement of a GCN further investigations related to the composition of the prior knowledge and the hyperparameter set and are a future work. Moreover, we intend to explore knowledge enhancement on heterogeneous graphs (Graphs with multiple edge and node types). Such graphs are particularly challenging since they have node and edge features of different shapes and are likely to introduce more space and time complexity. Another line of work is to make relational knowledge enhancement layers more efficient and reduce their run time.

References

[1] Zachary Susskind, Bryce Arden, Lizy K. John, Patrick Stockton, and Eugene B. John. Neuro-symbolic ai: An emerging class of ai workloads and their characterization, 2021. URL https://arxiv.org/abs/2109.06133. 1

[2] Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29: 17–23, 2019. ISSN 2352-1546. doi: https://doi.org/10.1016/j.cobeha.2018.12.010. URL https://www.sciencedirect.com/science/article/pii/S2352154618301943. Artificial Intelligence. 1

[3] Giuseppe Marra, Michelangelo Diligenti, Francesco Giannini, Marco Gori, and Marco Maggini. Relational neural machines, 2020. URL https://arxiv.org/abs/2002.02193. 1, 3

[4] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 303:103649, feb 2022. doi: 10.1016/j.artint.2021.103649. 1, 2, 3

[5] Alessandro Daniele and Luciano Serafini. *Knowledge Enhanced Neural Networks*, pages 542–554. 08 2019. ISBN 978-3-030-29907-1. doi: 10.1007/978-3-030-29908-8_43. 1, 2, 3, 4

[6] Alessandro Daniele and Luciano Serafini. Knowledge enhanced neural networks. In Abhaya C. Nayak and Alok Sharma, editors, *PRICAI 2019: Trends in Artificial Intelligence*, pages 542–554, Cham, 2019. Springer International Publishing. ISBN 978-3-030-29908-8. 1

[7] Alessandro Daniele and Luciano Serafini. Neural networks enhancement with logical knowledge, 2020. URL https://arxiv.org/abs/2009.06087. 1, 2, 3, 4, 7, 8

[8] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 496–503. AAAI Press, 2003. ISBN 1577351894. 2

[9] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *CoRR*, abs/2005.00687, 2020. URL https://arxiv.org/abs/2005.00687. 2, 3, 7, 11

[10] Y. Ma and J. Tang. *Graph Neural Networks. In Deep Learning on Graphs*. Cambridge: Cambridge University Press., 2021. 2, 4, 5

[11] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. Sampling methods for efficient training of graph convolutional networks: A survey, 2021. URL https://arxiv.org/abs/2103.05872. 2, 5

[12] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2017. URL https://arxiv.org/abs/1706.02216. 2, 5

[13] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017. 2, 3, 7

[14] Ralph Abboud and İsmail İlkan Ceylan. Node classification meets link prediction on knowledge graphs, 2021. URL https://arxiv.org/abs/2106.07297. 2

[15] Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artificial Intelligence*, 302:103602, jan 2022. doi: 10.1016/j.artint.2021.103602. 3

[16] Akash Saxena and Vikram Rajpoot. A comparative analysis of association rule mining algorithms. *IOP Conference Series: Materials Science and Engineering*, 1099:012032, 03 2021. doi: 10.1088/1757-899X/1099/1/012032. 3

[17] Ismat Beg and Samina Ashraf. Similarity measures for fuzzy sets. *Applied and Computational Mathematics*, 8:192–202, 03 2009. 3

[18] Matthias Fey, Jan E. Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings, 2021. URL https://arxiv.org/abs/2106.05609. 5, 12

[19] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018. URL http://arxiv.org/abs/1811.05868. 7

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL https://arxiv.org/abs/1502.03167. 7

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. 7, 8

[22] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 7, 8

[23] Lukas Biewald. Experiment tracking with weights and biases, 2020. URL https://www.wandb.com/. Software available from wandb.com. 8

[24] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020. 11

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf`. 11

[26] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010. 13, 14

A Appendix

A.1 Data Source

In this work, we make use of the datasets ogbn-arxiv and ogbn-products. Detailed information and public access to the data source is given in [24] and on the OGB webpage[4].

A.1.1 ogbn-arxiv

ogbn-arxiv [9] is a citation graph extracted from the scientific platform Arxiv. Each node in the graph represents a research paper of the computer science domain. Directed edges amongst the papers indicate citations. Each paper has a 128-dimensional feature vector that is obtained with a word2vec [25] model from the text in the title and the abstract. The documents in the graph belong to one of 40 classes and the dataset is split into training, validation and test set based on the publication dates (ratio 54/18/28). On ogbn-arxiv a multi-class node classification task can be conducted in a supervised learning setting since the ground truth classes are provided. The proposed metric to evaluate the model performance is the accuracy. The dataset has the following statistics:

- Number of nodes : 169343
- Number of edges: 1166243
- Avg node degree: 13.7

A.2 ogbn-products

ogbn-products [9] is a Amazon purchasing network that contains products sold on the platform Amazon. The products are represented as nodes in the graph. Two nodes are linked by an edge if the respective products were purchased together. The graph is undirected. The dataset contains node features that are derived from the product descriptions and encoded as bag-of-word vectors. The dataset is split into training, validation and test set according to the sales rank (ratio: 8/2/90). The task is to predict one of 47 product categories (multi-class node classification).

- Number of nodes: 2.449.020
- Number of edges: 61.859.140
- Average node degree: 50.5

Algorithm 1 Forward Propagation with subgraphs sampled with RN Sampler .

```
Input
    Training Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$
    Sampling Parameters
Output
    Training loss per epoch

1: $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_s \leftarrow$ Sampled Subgraphs (RNS in Algorithm 2).
2: for $i \in \{1, \ldots, S\}$ do
3:     $\{\hat{y}_v | v \in \mathcal{V}_i\} \leftarrow$ Forward propagation for all nodes
4:     Backward Propagation loss on first $b$ nodes in $\{\hat{y}_v | v \in \mathcal{V}_i\}$
5: end for
```

Algorithm 2 Restrictive neighbourhood sampling

```
Input
    Training Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$
    Sampling Parameters: batch size $b$, sampling depth $\mathbf{l}$, sampling size $n_s$
Output
    List of sampled subgraphs (batches): $\mathcal{G}_1(\mathcal{V}_1, \mathcal{E}_1), \ldots, \mathcal{G}_S(\mathcal{V}_S, \mathcal{E}_S)$

1: $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_S \leftarrow$ target node sets of size $b$, sampled without replacement from $\mathcal{V}$. ▷ (Last batch
   might be smaller than $b$)
2: for $i \in \{1, \ldots, S\}$ do
3:     for $\ell \in \{1, \ldots, \mathbf{l}\}$ do
4:         $\mathcal{N}^\ell(\mathcal{V}_i) \leftarrow$ sample randomly $n_s$ $\ell$-order neighbours for each node in $\mathcal{V}_i$
5:         $\mathcal{V}_i \leftarrow \mathcal{V}_i \cup \mathcal{N}^\ell(\mathcal{V}_i)$                    ▷ append and remove duplicated nodes
6:     end for
7: end for
```

486 A.3 Algorithms

487 A.4 Full-Batch Training Results

	Full-batch training on ogbn-arxiv			
	avg train accuracy	avg validation accuracy	avg test accuracy (stdv)	avg epoch time
MLP	0.5852	0.5635	0.5403 (0.0061)	0.065
KE_MLP	0.6127	0.6000	0.5713 (0.1063)	0.768
GCN	0.6304	0.5924	0.5273 (0.019)	0.182
KE_GCN	0.5911	0.5647	0.4978 (0.0205)	0.888

Table 3: Results of full-batch training on ogbn-arxiv. (Parameter setting in Table 5, Section A.5).

	Full-batch training on ogbn-products			
	avg train accuracy	avg validation accuracy	avg test accuracy (stdv)	avg epoch time
MLP	OOM	OOM	OOM	-
KE_MLP	OOM	OOM	OOM	-
GCN	OOM	OOM	OOM	-
KE_GCN	OOM	OOM	OOM	-

Table 4: Full-batch on ogbn-products results in OOM error, see also [18]

[4] https://ogb.stanford.edu/

A.5 Hyperparameters

Parameters of Full-batch Training on ogbn-arxiv				
Model	MLP	GCN	KE_GCN	KE_MLP
Binary Preactivation	500.0	500.0	500.0	500.0
Dropout	0.5	0.5	0.5	0.5
Epochs	600	600	600	600
Early Stopping Enabled	True	True	True	True
Early Stopping δ	0.001	0.001	0.001	0.001
Early Stopping Patience	10	10	10	10
Evaluation Steps	10	10	10	10
Hidden Channels	256	256	256	256
Learning Rate	0.01	0.01	0.01	0.01
Number of KE layers	-	-	3	3
Number of Hidden layers (base NN)	3	3	3	3
Runs	10	10	10	10

Table 5

RNS on ogbn-arxiv				
Model	MLP	GCN	KE_GCN	KE_MLP
Batch Size	10000	10000	10000	10000
Sampling Depth	-	3	3	3
Sampling Size	-	10	10	10
Binary Preactivation	500.0	500.0	500.0	500.0
Dropout	0.5	0.5	0.5	0.5
Epochs	300	100	100	300
Early Stopping Enabled	True	True	True	True
Early Stopping δ	0.001	0.001	0.001	0.001
Early Stopping Patience	10	10	10	10
Evaluation Steps	10	10	10	10
Hidden Channels	256	256	256	256
Initialisation of KE layers	-	-	0.5	0.5
Initialisation of GC layers	-	random (glorot) [26]	-	random (glorot)
Initialisation of linear layers	random uniform	random uniform	random uniform	random uniform
Learning Rate	0.01	0.01	0.01	0.01
Number of KE layers	-	-	3	3
Number of Hidden layers (base NN)	3	3	3	3
Runs	10	10	10	10

Table 6

RNS on ogbn-products, sampling depth=1, 1 KE layer, batch size 10.000				
Model	MLP	GCN	KE_GCN	KE_MLP
Batch Size	10.000	10.000	10.000	10.000
Sampling Depth	-	1	1	1
Sampling Size	-	10	10	10
Binary Preactivation	500.0	500.0	500.0	500.0
Dropout	0.5	0.5	0.5	0.5
Epochs	300	300	300	300
Early Stopping Enabled	True	True	True	True
Early Stopping δ	0.001	0.001	0.001	0.001
Early Stopping Patience	10	10	10	10
Evaluation Steps	10	10	10	10
Hidden Channels	256	256	256	256
Initialisation of KE layers	-	-	0.5	0.5
Initialisation of GC layers	-	random (glorot) [26]	-	random (glorot)
Initialisation of linear layers	random uniform	random uniform	random uniform	random uniform
Learning Rate	0.01	0.01	0.01	0.01
Number of KE layers	-	-	1	1
Number of Hidden layers (base NN)	3	3	3	3
Runs	10	10	10	10

Table 7