Lossless Compression of Structured Convolutional Models via Lifting

Gustav Sourek*
Czech Technical University in Prague
souregus@fel.cvut.cz

Filip Zelezny
Czech Technical University in Prague
zelezny@fel.cvut.cz

Abstract

Lifting is an efficient technique to scale up graphical models generalized to relational domains by exploiting the underlying symmetries. Concurrently, neural models are continuously expanding from grid-like tensor data into structured representations, such as various attributed graphs and relational databases. To address the irregular structure of the data, the models typically extrapolate on the idea of convolution, effectively introducing parameter sharing in their, dynamically unfolded, computation graphs. The computation graphs themselves then reflect the symmetries of the underlying data, similarly to the lifted graphical models. Inspired by lifting, we introduce a simple and efficient technique to detect the symmetries and compress the neural models without loss of any information. We demonstrate through experiments that such compression can lead to significant speedups of structured convolutional models, such as various Graph Neural Networks, across various tasks, such as molecule classification and knowledge-base completion.

1 Introduction

Lifted, often referred to as *templated*, models use highly expressive representation languages, typically based in weighted predicate logic, to capture symmetries in relational learning problems [18]. This includes learning from data such as chemical, biological, social, or traffic networks, and various knowledge graphs, relational databases and ontologies. The idea has been studied extensively in probabilistic settings under the notion of lifted graphical models [15], with instances such as Markov Logic Networks (MLNs) [25] or Bayesian Logic Programs (BLPs) [14].

In a wider view, *convolutions* can be seen as instances of the templating idea in neural models, where the same parameterized pattern is being carried around to exploit the underlying symmetries, i.e. some forms of shared correlations in the data. In this analogy, the popular Convolutional Neural Networks [19] themselves can be seen as a simple form of a templated model, where the template corresponds to the convolutional filters, unfolded over regular spatial grids of pixels. But the symmetries are further even more noticeable in structured, relational domains with discrete element types. With convolutional templates for regular trees, the analogy covers Recursive Neural Networks [33], popular in natural language processing. Extending to arbitrary graphs, the same notion covers works such as Graph Convolutional Networks [16] and their variants [40], as well as various Knowledge-Base Embedding methods [38]. Extending even further to relational structures, there are works integrating parameterized relational logic templates with neural networks [35, 26, 21].

The common underlying principle of templated models is a joint parameterization of the symmetries, allowing for better generalization. However, standard lifted models, such as MLNs, provide another key advantage that, under certain conditions, the model computations can be efficiently carried out without complete template unfolding, often leading to even exponential speedups [15]. This is known as "lifted inference" [13] and is utilized heavily in lifted graphical models as well as database query engines [36]. However, to our best knowledge, this idea has been so far unexploited in the neural

*Corresponding author. Preprint. Under review.

(convolutional) models. The main contribution of this paper is thus a "lifting" technique to compress symmetries in convolutional models applied to structured data.

1.1 Related Work

From the deep learning perspective, there have been various model compression techniques proposed, such as pruning, decreasing precision, and low-rank factorization [2]. However, all the existing techniques seem lossy in nature, and do not exploit the computation symmetries. Naturally related are lifted inference techniques for templated probabilistic graphical models [13], from which the most closely related are techniques for approximate inference [30], lifting ground models by automatically detecting the symmetric structures with graph bisimulation [12]. Since we propose a lifted inference technique for neural models, we are generally related to approaches integrating standard lifted and neural models, including recent works such as Deep-Problog [21]. While Deep-Problog is capable to efficiently exploit symmetries in the logical part of the template, there is a very strict separation from the neural part, which is just a completely arbitrary gradient-based model where the symmetries do not play any role. Another recent work are Neural Markov Logic Networks [22], where the relational template is deeply dissolved in the neural model, however this also prevents from the explicit model compression by lifting. The most relevant integrative framework are Lifted Relational Neural Networks (LRNNs) [35] which however, despite the name, provide only templating capabilities without lifted inference, i.e. with uncompressed ground model computations.

2 Structured Convolutional Models

We address neural models designed to learn from (multi-)relational, structured data, while exploiting some form of parameter sharing, as "structured convolutional models"[1]. Examples of such models include Recursive Neural Networks [33, 23], Graph Neural Networks [16, 28], and various Relational Neural Networks [34, 26, 5, 31], designed for regular trees, graphs, and general relational structures, respectively. They all utilize the idea of convolution (templating), where the same parameterized pattern is carried over different subparts of the data (representation) with the same local structure. For each data sample, these models dynamically unfold a computation graph by repeatedly applying the patterns, and their parameters, over all the matching subparts, effectively introducing parameter sharing in the respective computation graphs. When applied to data exhibiting symmetric substructures, such as edges in a graph or their various conformations, this further introduces symmetric, repetitive computations, which we exploit in this work. Let us briefly review some of the most popular convolutional models sorted by increasing expressiveness.

Convolutional Neural Networks CNNs are feed-forward models utilizing the operations of *convolution* and *pooling*. Here, the convolution filter (template) is a small tensor of learnable parameters, applied by a successive element-wise multiplication with equally-sized subparts of the input tensor. The pooling operation then aggregates the resulting values over some predefined regions. The main idea behind the convolution here is to abstract away common *patterns* out of different spatial regions of the input, while pooling aims to enforce *invariance* w.r.t. translation within the regions.

Recursive Neural Networks RNNs differ significantly from CNNs in that the structure of the computation graph is not given in advance. Instead, it directly follows the structure of each input example, which takes the form of an arbitrarily shaped $k-$regular tree. Similarly to CNNs, RNNs use the same parameterized operation ("convolution") to combine nodes into their respective parents over the whole tree [32], which is further done in recursive fashion. The main idea is that a computation graph can be *dynamically* generated for each individual example, while exploiting the convolution to discover the underlying *compositionality* of learning representations in recursive structures.

Graph Neural Networks GNNs can be seen as a further extension of the principles to completely irregular graph structures. Similarly to RNNs, they dynamically unfold the computation graph, however, GNNs introduce separately parameterized hidden layers, where the structure of each layer i exactly follows the structure of the *whole* input graph. For computation of the next layer $i + 1$ representations, each node in the graph updates its own representation by aggregating ("pooling") the representations of the adjacent nodes, transformed by some parameterized operation ("convolution"),

[1] A popular recent notion for this learning setting is "geometric deep learning" [1].

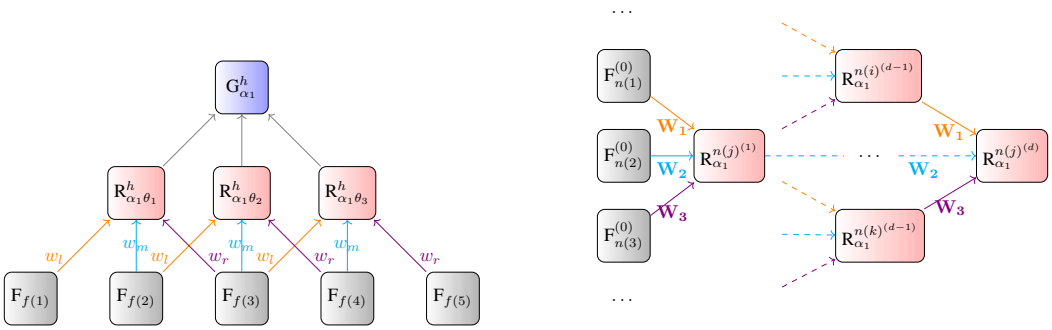

Figure 1: CNN (left) and RNN (right) computation structures, as encoded through LRNNs.

which is again reused over the whole graph at each layer. This general principle covers a wide variety of GNN models, such the popular GCNs [16], graph-SAGE [9], GIN [41], and others [42, 8], which then reduces to the respective choices of particular aggregations and transformation.

2.1 Relational Neural Networks

The principles of various relational neural frameworks differ more significantly, as they are not so established yet. We select LRNNs[2] as a representative, since it can be seen as the most direct further generalization of templating from graphs to (multi-)relational data, such as relational databases and ontologies. Similarly to lifted models, such as BLPs [14], the template in LRNNs takes the form of a parameterized logic program, i.e. a set of weighted relational rules (α_i), making it highly expressive.

Similarly to the introduced convolutional models, it dynamically unfolds the parameterized template over the relational structures, while introducing parameter sharing in the resulting computation graphs. However, the used convolution patterns can be very complex, and the matching against the data is carried out by an integrated theorem prover. The prover then derives a least Herbrand model [7], which is consequently translated into a neural model by recursively stacking three types of operations denoted as rule ("convolution") nodes (R), aggregation ("pooling") nodes (G), and atom nodes (A), respectively. Additionally, fact nodes (F) are used to represent the input data themselves[3].

2.1.1 Examples

Since explaining the internals of LRNNs is beyond the scope of this paper (for a complete description we refer to [35]), we instead exemplify its functionality by emulating the introduced, well-known, convolutional models, while demonstrating the underlying symmetries in their computations.

CNNs For simplicity, consider a one-dimensional "image" consisting of 5 pixels $i = 1, ..., 5$. While the sequential structure of the pixels is inherently assumed in CNNs, we encode it explicitly, e.g. as $next(1, 2), \ldots, next(4, 5)$. The value v_i of each pixel i can then be encoded by a corresponding weighted fact $v_i{:}f(i)$. Then consider a standard convolution filter h of size $[1, 3]$, i.e. a vector combining each three consecutive pixels ([left,middle,right]) by respective element-wise multiplications. This computation pattern, followed by pooling that aggregates all the resulting values into some hidden node "h", can then be encoded by the following weighted rule (α_1):

```
h :- w_l: f(A), w_m: f(B), w_r: f(C), next(A,B), next(B,C).
```

A computation graph unfolded from this pattern over some 5 consecutive pixels is shown in Fig. 1.

[2] which we re-implemented and extended with the compression technique, as well as other improvements. A Github repository with all the experiments is available at `https://github.com/GustikS/NeuraLifting`, with the framework itself being developed at `https://github.com/GustikS/NeuraLogic`.

[3] Note we further use this simplified notation for nodes (R,G,A,F) and rule groundings (α,θ) in the figures.

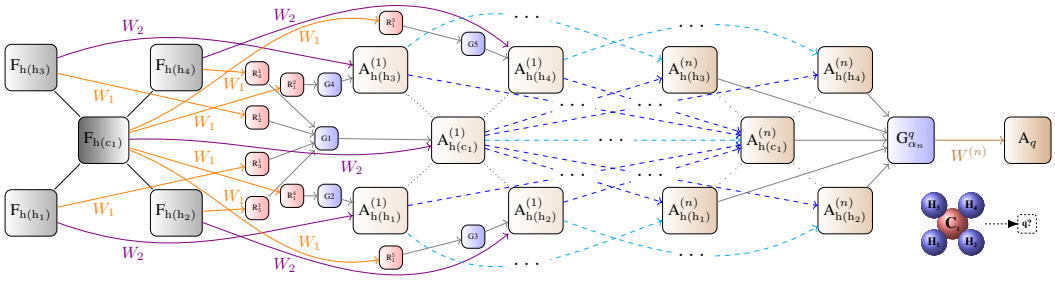

Figure 2: A multi-layer GNN structure (g-SAGE) with global readout, as encoded through LRNNs.

RNNs Consider a 3-regular tree, for which we firstly again encode the structure itself by providing a fact connecting each parent node to its child-nodes, e.g. as $parent(n_j^{i+1}, n_l^i, n_{l+1}^i, n_{l+2}^i)$. Secondly, we associate all the leaf nodes in the tree with their embedding vectors by weighted facts $[\mathbf{v_1^i}, \ldots, \mathbf{v_n^i}]$:$n(leaf_i)$. Finally, a single rule (α_1) can then be used to encode the recursive composition of representations in all trees as follows, with the unfolded computation graph again shown in Fig. 1:

```
n(P) :- W1:n(C1), W2:n(C2), W3:n(C3), parent(P,C1,C2,C3).
```

GNNs The whole generic computation at each layer i in GNNs can be represented as follows:

```
W(i):: h(i)(V) :- h(i-1)(U), edge(V,U).
```

where $edge/2$ is the binary relation of the input graphs. With proper choice of activations (G=avg,R=A=$ReLU$) this rule directly corresponds to a 1-layer GCN [16]. To further include central nodes in the update operation [40], we can split the template into 2 rules as

```
h(i)(V) :- W1(i): h(i-1)(U), edge(V,U).
h(i)(V) :- W2(i): h(i-1)(V).
```

which, by choosing respective activations (R=$ReLU$,G=max,A=$identity$), directly corresponds to a graph-SAGE layer [9]. Its computation over an example molecule of methane is shown in Fig. 2.

3 Lossless Compression via Lifting

The idea for the compression is inspired by lifted inference [13] used in templated graphical models. The core principle is that all equivalent sub-computations can be effectively carried out in a single instance and broadcasted into successive operations together with their respective multiplicities, potentially leading to significant speedups. While the corresponding "liftable" template formulae (or database queries) generating the isomorphisms are typically assumed to be given [15], we explore the symmetries from the ground structures, similarly to the methods based on graph bisimulation [30]. All the lifting techniques are then based in some form of first-order variable elimination (summation), and are inherently designed to explore *structural* symmetries in graphical models. In contrast, we aim to explore *functional* symmetries, motivated by the fact that even structurally different neural computation graphs may effectively perform identical function.

The learning in neural networks is also principally different from the model counting-based computations in lifted graphical models in that it requires many consecutive evaluations of the models as part of the encompassing iterative optimization routine. We take advantage of this fact and, instead

Algorithm 1 Neural network compression based on detecting functional symmetries

function COMPRESS (*Network*)
  $\mathcal{N} \leftarrow \bigcup$ neurons (*Network*)
  $\mathcal{N} \leftarrow$ topologicOrder ($\mathcal{N}$)
  $\mathcal{W} \leftarrow \bigcup$ weights (*Network*)
  $\overline{M} \leftarrow$ ISOCLASSES ($\mathcal{N}, \mathcal{W}$)
  $out \leftarrow \varnothing$
  for $(valueList, \mathcal{N}') \in \overline{M}$ **do**
    $out \leftarrow out \bigcup$ MERGENEURONS ($\mathcal{N}'$)
  return out
function ISOCLASSES($\mathcal{N}, \mathcal{W}$)
  $M[neuron \mapsto valueList] \leftarrow \varnothing$
  $i \leftarrow 0$
  while $i < repetitions$ **do**
    $\mathcal{W} \leftarrow$ randomInit ($\mathcal{W}$)
    $(\mathcal{N}, \mathcal{V}) \leftarrow$ inferValues ($\mathcal{N}, \mathcal{W}$)
    for $(neuron, value) \in (\mathcal{N}, \mathcal{V})$ **do**
      $valueList \leftarrow M[neuron]$
      $valueList = valueList \cup value$
    $i \leftarrow i + 1$
  $\overline{M}[valueList \mapsto \mathcal{N}'] \leftarrow$ reverseMap (M)
  return $\overline{M}$
function MERGENEURONS($\mathcal{N}$)
  $etalon \leftarrow$ takeOne ($\mathcal{N}$)
  optional: $\mathcal{N} \leftarrow$ ISOCHECK ($etalon, \mathcal{N}$)
  $allOutputs \leftarrow \bigcup$ outputs ($\mathcal{N}$)
  for $output \in allOutputs$ **do**
    $(children, \mathcal{W}') \leftarrow$ inputs ($output$) $\cap \mathcal{N}$
    replace ($output, children, etalon, \mathcal{W}'$)
  return connectedComponent($\mathcal{N}, etalon$)
function ISOCHECK($e, \mathcal{N}$)
  $\mathcal{N}' \leftarrow \varnothing$
  for $n \in \mathcal{N}$ **do**
    if $n = e$ **then**
      $\mathcal{N}' \leftarrow \mathcal{N}' \cup n$
    else if inputs(n) = inputs(e) = $\varnothing$ **then**
      if value (n) = value (e) **then**
        $\mathcal{N}' \leftarrow \mathcal{N}' \cup neuron$
    else if inputs(n) = inputs(e) **then**
      $\mathcal{N}' \leftarrow \mathcal{N}' \cup neuron$
  return $\mathcal{N}'$

of complex template analysis, propose a tightly inlined generic technique based in the neural model evaluation itself. Consequently, the technique requires but a single linear pass through each network.

Polynomial identity testing The detection of functional subgraph symmetries is then inspired by polynomial identity testing (PIT) [27]. While the complexity of PIT is an open problem, there is a simple but very efficient randomized algorithm ("PIT heuristic"). The idea is to evaluate the polynomials at random points which, if the polynomials were identical, necessarily results in the same values. The probability that they were equal by chance can then be bound based on Schwartz-Zippel Lemma [3], and rapidly decreased with repeated sampling. We basically adopt the same idea and, instead of arithmetic circuits, apply it to the neural computation graphs. The routine is explained in Algorithm 1. Apart from repetitions, the efficiency of the routine can be further increased by replacing the standard activation functions with more injective mappings to a wider output value range.

Certifying subgraph isomorphisms As the PIT algorithm is designed for functional symmetries and may produce value clashes for non-isomorphic subgraphs, one might be interested in their detection. For that we introduce an additional check, based on the following inductive statements:

1. Every two neurons with no inputs and the same output value are necessarily isomorphic.
2. Every two neurons with the same activation function and isomorphic sets of inputs (including the associated weights) are necessarily roots of isomorphic subgraphs.

Processing the neurons in topologic order then ensures that gradually bigger subpgraphs can be checked for isomorphisms in linear time w.r.t. number of the current root neuron input connections. The check, optionally incorporated in the neuron merging operation, is at the end of Algorithm 1.

Lossy compression If two subgraphs are isomorphic in the described sense, they necessarily lead to identical computations, including parameter updates, and can thus be interchanged safely. However, as discussed, this structural symmetry requirement is too strict, which is why we check for function value equivalence instead. However, even if the values for two subgraphs are not exactly equivalent but merely similar, one might also expect to observe similar learning behavior. Analogically to techniques for *approximate* lifted inference [30], we extend the idea to a lossy compression setting, which can be easily emulated by conditioning the value equality check with some numeric precision.

Finally, let us demonstrate the lossless compression (lifting) on the graph-SAGE model (sec. 2.1.1), unfolded over a sample molecule of methane in Fig.2, the result of which is displayed in Fig. 3.

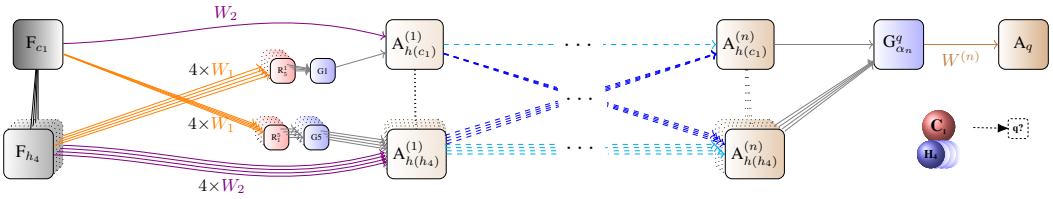

Figure 3: A compressed version of the GNN from Fig. 2, with isomorphic nodes denoted by dotting.

4 Experiments

To test the proposed compression, we selected some advanced structured convolutional models (Sec. 2), and evaluated them on a number of real datasets from the domains of (i) molecule classification and (ii) knowledge-base completion. The questions to be answered by the experiments are:

1. How numerically efficient is the PIT heuristic in achieving lossy/lossless compression?
2. What improvements does the compression provide in terms of model size and speedup?
3. Is learning accuracy truly unaffected by the, presumably lossless, compression in practice?

Models We choose GNNs as they encompass ideas from all the preceding models (convolution, pooling, dynamic structure and recursive layer stacking). Particularly, we choose well-known instances of GCNs and g-SAGE introduced in Sec. 2.1, each with 2 layers. Additionally, we include recent Graph Isomorphism Networks (GIN) [41], which follow the same computation scheme with 5 layers, but their particular operations (A=MLP,R=$identity$,G=sum) are theoretically substantiated in the expressiveness of the Weisfeiler-Lehman test [39]. This is interesting in that it should effectively distinguish non-isomorphic substructures in the data by generating consistently distinct computations, and should thus be somewhat more resistant to our proposed compression. Finally, we include a sample relational template ("graphlets") introduced in [34], which generalizes GNNs to aggregate small 3-graphlets instead of just neighbors.

Datasets For structure property prediction, we used 78 organic molecule classification datasets reported in previous works [24, 10, 20]. Nevertheless, we show only the (alphabetically) first 3 for clarity, as the target metrics were extremely similar over the whole set. We also extended GCNs with edge embeddings to account for the various bond types, further *decreasing* the symmetries. For knowledge base completion (KBC), we selected commonly known datasets of Kinships, Nations, and UMLS [17] composed of different object-predicate-subject triplets. We utilized GCNs to learn embeddings of all the items and relations, similarly to R-GCNs [29], and for prediction of each triplet, we fed the three embeddings into an MLP, such as in [4], which we denote as "KBE".

Experimental Protocol We approached all the learning scenarios under simple unified setting with standard hyperparameters, none of which was set to help the compression (mostly on the contrary). We used the re-implemented LRNN framework to emulate all the models, and also compared with popular GNN frameworks of PyTorch Geometric (PyG) [6] and Deep Graph Library (DGL) [37]. If not dictated by a model, we set the activation functions simply as R = A = $\frac{1}{1+e^{-x}}$ and G=avg. We trained against MSE using 1000 steps of ADAM, and evaluated with a 5-fold crossvalidation.

4.1 Results

Firstly, we tested efficiency of the neural PIT heuristic itself (Sec. 3), for which we used scalar weight representation in the models to detect symmetries on the level of individual neurons (rather than layers), reflecting the original PIT more closely. We used the most complex graphlets model, where we checked the functional symmetries to overlap with the structural symmetries. The results in Fig. 4 then show that the heuristic is already able to perfectly distinguish all isomorphisms with but a single initialization within less than 12 significant digits. While more initializations indeed improved the efficiency rapidly, in the end they proved unnecessary (but could be used in cases where the available precision would be insufficient). Moreover this test was performed with the actual low-range logistic

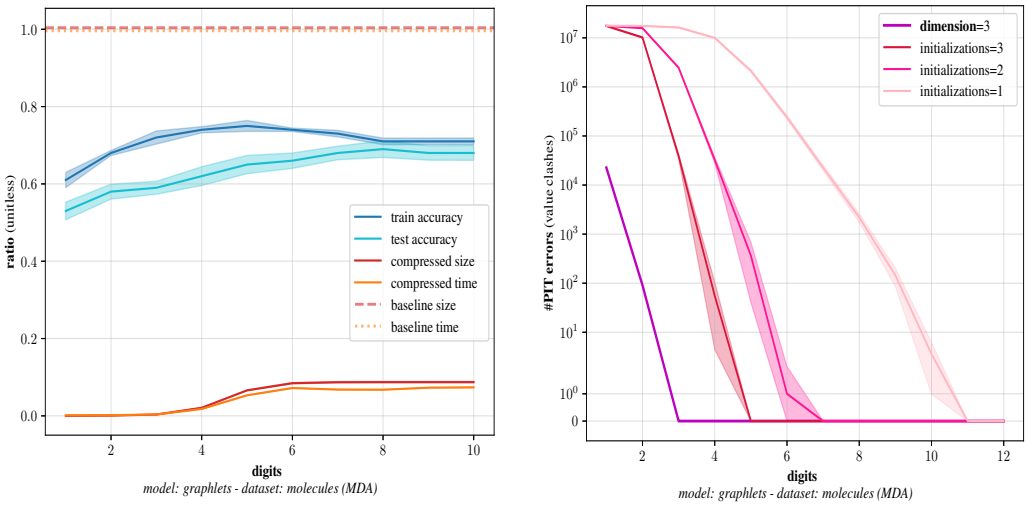

Figure 4: Compression of a *scalar*-parameterized graphlets template on a molecular dataset. We display progression of the selected metrics w.r.t. increasing number of significant digits (inits=1) used in value comparisons (left), and number of non-isomorphic subgraph value clashes detected by the additional check w.r.t. the digits, weight re-initializations, and increased value dimension (right).

Table 1: Training times *per epocha* across the different models and frameworks over 3000 molecules. Additionally, the startup model creation time (theorem proving) overhead of LRNNs is displayed.

Model	Lifting (s)	LRNNs (s)	PyG (s)	DGL (s)	LRNN startup (s)
GCN	**0.25** $\pm$ **0.01**	0.75$\pm$ 0.01	3.24 $\pm$ 0.02	23.25 $\pm$ 1.94	35.2 $\pm$ 1.3
g-SAGE	**0.34** $\pm$ **0.01**	0.89$\pm$ 0.01	3.83 $\pm$ 0.04	24.23 $\pm$ 3.80	35.4 $\pm$ 1.8
GIN	**1.41** $\pm$ **0.10**	2.84$\pm$ 0.09	11.19 $\pm$ 0.06	52.04 $\pm$ 0.41	75.3 $\pm$ 3.2

activations. The displayed (10x) training time improvement (Fig.4 - left) in the scalar models was then directly reflecting the network size reduction, and could be pushed further by decreasing the numeric precision at the expected cost of degrading the learning performance.

Secondly, we performed similar experiments with standard tensor parameterization, where the isomorphisms were effectively detected on the level of whole neural "layers", since the vector output values (of dim=3) were compared for equality instead. This further improved the precision of the PIT heuristic (Fig. 4 - right), where merely the first 3 digits were sufficient to achieve lossless compression in all the models and datasets (Figure 5). However, the training (inference) time was no longer directly reflecting the network size reduction, which we account to optimizations used in the vectorized computations. Nevertheless the speedup (app. 3x) was still substantial.

We further compared with established GNN frameworks of PyG [6] and DGL [37]. We made sure to align the exact computations of GCN, g-SAGE, and GIN, while all the frameworks performed equally w.r.t. accuracy. For the comparison, we further increased all (tensor) dimensions to a more common dim=10. The compression effects, as well as performance edge of the implemented LRNN framework itself, are displayed in Tab. 1 for a sample molecular dataset (MDA). Note that the compression was truly least effective for the GIN model, nevertheless still provided app. 2x speedup.

Finally, the results in Fig. 6 confirm that the lossless lifting, with either the isomorphism checking or simply high enough precision, indeed does not degrade the learning performance in terms of training and testing accuracy (both were close within margin of variance over the crossvalidation folds).

Note that the used templates are quite simple and do not generate any symmetries on their own (which they would, e.g., with recursion), but rather merely reflect the symmetries in the data themselves. The speedup was lowest for the sparse graph of Nations, further decomposed by the 2 distinct relation types, and higher for the Kinships dataset, representing a more densely interconnected social network. The improvement was then generally biggest for the highly symmetric molecular graphs

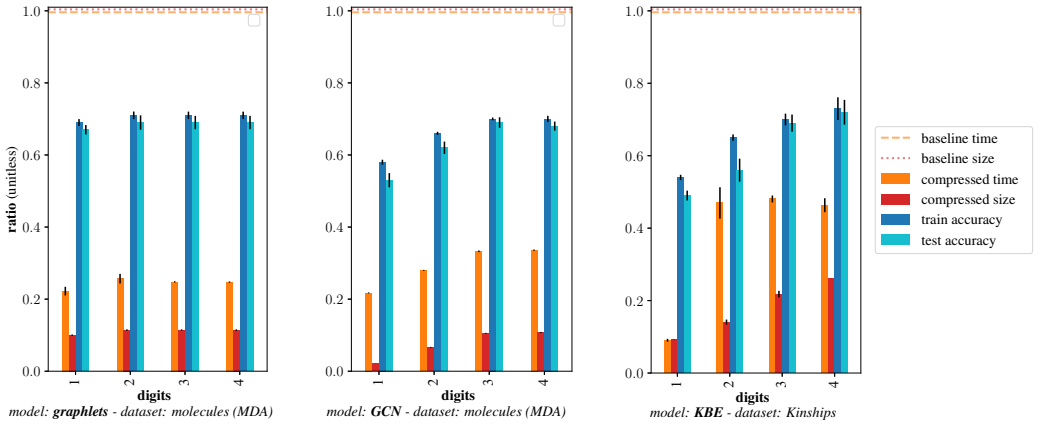

Figure 5: Compression of 3 *tensor*-parameterized templates for graphlets (left), GCNs (middle) and KBEs (right) over the molecular (left, middle) and Kinships (right) datasets, with progression of selected metrics shown against the increasing number of significant digits used for the PIT checking.

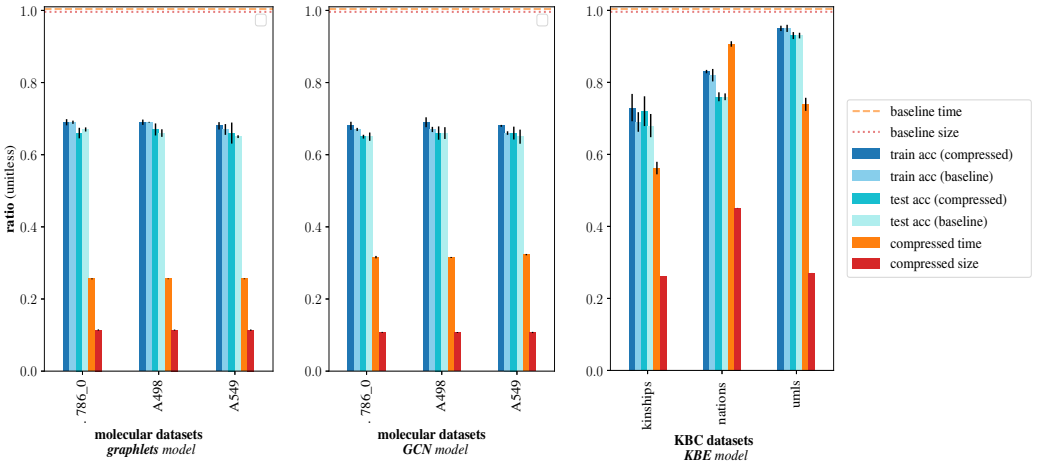

Figure 6: Comparison of 3 different baseline models of graphlets (left), GCNs (middle), and KBEs (right) with their compressed versions over molecule classification (left, middle) and KBC (right).

where, interestingly, the compression often reduced the neural networks to a size even smaller than that of the actual input molecules. Note we only compressed symmetries within individual computation graphs, and the results are thus not biased by a potential existence of isomorphic samples [11], however potentially much higher compression rates could be also achieved with (dynamic) batching.

5 Conclusion

We introduced a simple, efficient, lossless compression technique for structured convolutional models inspired by lifted inference. The technique is very light-weight and can be easily adopted by any neural learner, but is most effective for neural models utilizing weight sharing schemes to target relational data, such as in various graph and relational neural networks. We have demonstrated with existing models and datasets that a significant inference and training time reduction can be achieved without affecting the learning results, and possibly extended beyond for additional speedup.

References

[1] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[2] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[3] Richard A DeMillo and Richard J Lipton. A probabilistic remark on algebraic program testing. Technical report, Georgia Inst. of Technology, Atlanta School of Information and Computer science, 1977.

[4] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

[5] Richard Evans and Edward Grefenstette. Learning Explanatory Rules from Noisy Data. *To Appear in Journal of Artificial Intelligence Research Submitted*, 02, 2017.

[6] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[7] Jean H Gallier. *Logic for computer science: foundations of automatic theorem proving*. Courier Dover Publications, 2015.

[8] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[9] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[10] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001.

[11] Sergei Ivanov, Sergei Sviridov, and Evgeny Burnaev. Understanding isomorphism bias in graph data sets. *arXiv preprint arXiv:1910.12091*, 2019.

[12] Paris C Kanellakis and Scott A Smolka. Ccs expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990.

[13] Kristian Kersting. Lifted probabilistic inference. In *ECAI*, pages 33–38, 2012.

[14] Kristian Kersting and Luc De Raedt. Towards combining inductive logic programming with bayesian networks. In *Inductive Logic Programming, 11th International Conference, ILP 2001, Strasbourg, France, September 9-11, 2001, Proceedings*, pages 118–131, 2001.

[15] A Kimmig, L Mihalkova, and L Getoor. Lifted graphical models: a survey. *Machine Learning*, 99(1):1–45, 2015.

[16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[17] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *Proceedings of the 24th International Conference on Machine Learning*, pages 433–440, 2007.

[18] Daphne Koller, Nir Friedman, Sašo Džeroski, Charles Sutton, Andrew McCallum, Avi Pfeffer, Pieter Abbeel, Ming-Fai Wong, David Heckerman, Chris Meek, et al. *Introduction to statistical relational learning*. MIT press, 2007.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[20] Huma Lodhi and Stephen Muggleton. Is mutagenesis still challenging. *ILP-Late-Breaking Papers*, 35, 2005.

[21] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, pages 3749–3759, 2018.

[22] Giuseppe Marra and Ondřej Kuželka. Neural markov logic networks. *arXiv preprint arXiv:1905.13462*, 2019.

[23] Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105, 1990.

[24] Liva Ralaivola, Sanjay J. Swamidass, Hiroto Saigo, and Pierre Baldi. Graph kernels for chemical informatics. *Neural Netw.*, 18(8):1093–1110, 2005.

[25] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 2006.

[26] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, 2017.

[27] Nitin Saxena. Progress on polynomial identity testing. *Bulletin of the EATCS*, 99:49–79, 2009.

[28] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[29] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer, 2018.

[30] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Bisimulation-based approximate lifted inference. *arXiv preprint arXiv:1205.2616*, 2012.

[31] Xujie Si, Mukund Raghothaman, Kihong Heo, and Mayur Naik. Synthesizing datalog programs using numerical relaxation. *arXiv preprint arXiv:1906.00163*, 2019.

[32] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934, 2013.

[33] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts, et al. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013.

[34] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, and Ondrej Kuzelka. Lifted relational neural networks. *arXiv preprint arXiv:1508.05128*, 2015.

[35] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezny, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *Journal of Artificial Intelligence Research*, 62:69–100, 2018.

[36] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis lectures on data management*, 3(2):1–180, 2011.

[37] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv preprint arXiv:1909.01315*, 2019.

[38] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[39] Boris Weisfeiler. *On construction and identification of graphs*, volume 558. Springer, 2006.

[40] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[41] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[42] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.