A Computational Perspective on Neural-Symbolic Integration

Gustav Šír
Czech Technical University in Prague, Czech Republic
E-mail: gustav.sir@cvut.cz

Abstract. Neural-Symbolic Integration (NSI) aims to marry the principles of symbolic AI techniques, such as logical reasoning or planning, with the learning capabilities of deep neural networks. In recent years, there have been many systems proposed to address this integration in a seemingly efficient manner. However, from the computational perspective, this is in principle impossible to do. Specifically, some of the core symbolic AI tasks are provably hard, hence a general neural-symbolic system necessarily needs to adopt this computational complexity, too. This might seem as a substantial downside from the perspective of modern large-scale deep learning, which most of the NSI systems try to circumvent by dropping parts of the symbolic AI capabilities. In this position paper, we argue that it might be better to openly accept and address this complexity as a foundational part of the NSI challenge, enabling to properly cover both the existing paradigms as a special case. From the computational perspective, this has important implications on the data and learning representations, the structure of the resulting computation graphs, and the underlying hardware and software stacks. Particularly, we argue that the currently prominent, tensor-based deep learning is insufficient as a foundation for general NSI, which we discuss in a wider context of established symbolic and relational learning methods, and outline some promising computational directions towards fully expressive and efficient NSI.

Keywords: Neural-Symbolic Integration, Computation Graphs, Learning Representations, Hardware Acceleration, Deep Learning Frameworks, Relational Learning, Logical Reasoning, Computational Complexity

1. Introduction

During the past decade, deep learning (DL) has taken over the wider domain of artificial intelligence (AI) so rapidly that many people now commonly confuse the fields as equivalent, expecting neural networks to eventually solve all AI problems at hand. This has only been accentuated with the recent rise of foundation (neural) models with applications such as ChatGPT, which exhibit many AI capabilities that have previously been though as out of reach for DL. Indeed, the dramatic progress in difficult, symbolic reasoning benchmarks has left a lot of people wondering whether there are even some limits to the current strategy of scaling large Transformers [1].

However, from the computational perspective, one thing remains striking - these neural models arrive at the answer to every possible input after a *fixed number* of (the same) computation steps, which is clearly suboptimal. For simple inputs this is a waste of compute, and for complex inputs there will always be a limit where this is insufficient. Particularly, many (symbolic) AI capabilities, such as reasoning or planning, are provably NP-hard (or worse) and thus require a number of computation steps exponential in the size of the input. Hence, this DL strategy is clearly not universal. However, such a fixed form of computation is the core characteristic of DL that perpetuates throughout the whole domain, as it is crucially required to make use of the underlying hardware-specific, data-parallel acceleration.

Researchers often like to think of their models and algorithms independently of the underlying hardware (HW) and software (SW). However, the resurrection of modern DL was a very clear demonstration that these worlds are

deeply intertwined. And while the recent computing evolution in the direction of specialized HW accelerators, particularly the Graphical Processing Units (GPUs), has been transformational for the success of DL, it unfortunately started to further deepen the gap between DL and the, more general, symbolic AI compute, making the exploration of novel neural-symbolic methods aimed at their integration increasingly more difficult. In NSI, this has been reflected by a gradual shift from the symbolic to the, insufficiently expressive, tensor-based neural processing methods, or towards compositional (unintegrated) methods, which largely loose benefits of the integrative approach.[1]

In this position paper, we will very briefly overview the evolution of the neural-symbolic approaches to AI, with a particular focus on *relational* logic representations, and describe the differences in their underlying computation structures. On the one hand, we will have the symbolic, logic-based approaches to AI working in the regime of discrete optimization, with long historical roots reaching all the way back to the origins of AI and computing itself, involving its very fathers such as Shannon, von Neumann, and Turing. These make heavy use of stateful computation with conditional branching, leading to very sparse and irregularly structured computation graphs that change dynamically from input to input. This form of compute builds on the universal capabilities of the Central Processing Unit (CPU) architecture, which dominated computing throughout the history. On the other hand, we will have the more recent, tensor-based neural approaches, working in the simpler regime of continuous optimization, which now depend crucially on the GPU architecture that, however, lacks support for many of the computing capabilities crucial for the symbolic methods.

This underlying dichotomy is then heavily reflected on the SW layer, too, with largely incompatible learning representations, models and methods. Nevertheless, building on the insights from the associated field of (statistical) relational learning, we will conclude this paper with a vision of how the field of NSI might benefit from the undergoing compute evolution surrounding the related, and increasingly popular, (deep) graph representation learning domain.

2. A Brief History of Neural-Symbolic Computation

Historically, the mainstream vision for AI has been fluctuating quite dramatically from the almost purely symbolic approaches in the early days to the almost purely neural approaches we see today, both evolving in a largely separate manner. However, history teaches us that whenever there are two streams of very opposing schools of thought, the resolution is typically somewhere in the middle.

2.1. From Neurons to Symbols

Perhaps surprisingly, the correspondence between the neural and logical calculus has been well-established throughout history. Dating all the way back to the introduction of the first computational neuron by McCulloch and Pitts in their 1943's paper "*A logical calculus of the ideas immanent in nervous activity*" [2], we can see that it was actually thought to emulate *logic* gates over input (binary-valued) propositions. The idea was based on the, now commonly exemplified, fact that logical connectives of conjunction and disjunction can be easily encoded by binary threshold units with weights (Fig. 1) — i.e., the perceptron, an elegant learning algorithm for which was introduced shortly.

While stacking these on top of each other was a clear path towards representation of more complex, nested logical functions, it took until the 1980's to reintroduce the chain rule for differentiation of nested functions as the "backpropagation" method to calculate gradients in such "neural networks" which, in turn, could be efficiently trained with gradient descent. For that, however, researchers had to replace the original binary threshold units with differentiable activation functions, such as the logistic sigmoid, which started digging a gap between the neural networks and their crisp logical interpretations.

[1] such as those explained in Sec. 3.3 and Sec. 5.2

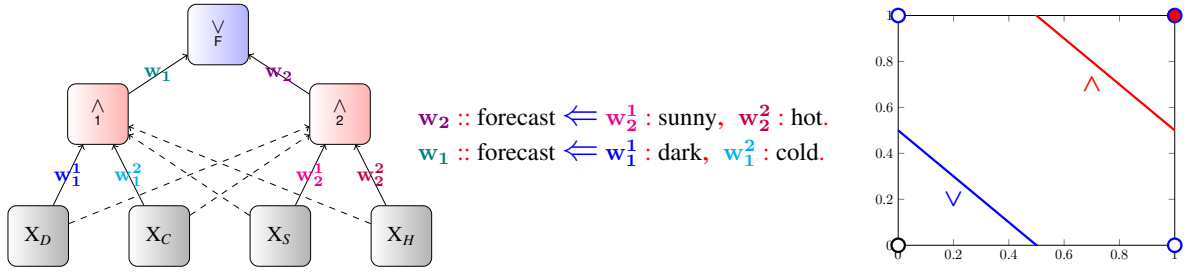

Fig. 1. An example correspondence between a neural network (left) and a propositional logic program (middle), based on conjunctive and disjunctive activations emulated by the respectively weighted neurons (right), in the spirit of KBANN [4].

2.2. *From Deep Learning to Logic*

These historical parallels might seem outlandish in the context of modern DL. However, interestingly, even the modern idea of DL was not originally described as bound to the neural networks, but rather universally to methods modeling hierarchical composition of useful concepts that are reused in different inference paths of target variables from the input samples [3]. And while this concept has most commonly been instantiated through "hidden layers" in DL, such hierarchical abstractions are generally very common to human thinking and logical reasoning, too.

In fact, *logic* is the very science of deduction of useful concepts from simpler premises in a hierarchical (nested) fashion. While constructing a proof in logic, auxiliary lemmas are often created to reduce the complexity of the theory in scope, in a fashion very similar to DL. Thus, instead of hidden neural layers, these hierarchical abstractions may also be thought of as "*complicated propositional formulae re-using many sub-formulae*".[2]

From the NSI perspective, this is not just some metaphor, but a number of actual systems reflecting this paradigm started to emerge throughout the '90s, such as the popular Knowledge-Based Artificial Neural Network (KBANN) [4] (Fig. 1). However, as imagined by Bengio, such a direct neural-symbolic correspondence was insurmountably limited to the aforementioned *propositional* setting, popularly referred to as "propositional fixation" of neural networks, coined by J. McCarthy [5].

2.3. *Problems with Relational Logic*

While the aforementioned computational correspondence between the propositional logic programs and neural networks was very direct, transferring the same principle to the *relational*[3] setting has remained a major challenge [6]. The issue is that in the propositional setting, only the (binary) values of the existing input propositions are changing, while the structure of the logical program stays fixed. This is easy to think of as a boolean circuit (neural network) sitting on top of a propositional interpretation (feature vector).

However, the relational input interpretations can no longer be thought of as individual values over a fixed (finite) number of propositions (interpretation vector), but an unbound set of related atoms that are true in a given world (a Herbrand model [7]). Consequently, also the structure of the logical inference unfolded upon this representation can no longer be represented by a fixed boolean circuit. Naturally, when proving a query in relational (first-order) logic, we generally do not know the number of objects that will form input into the proof, we do not know the number of auxiliary lemmas that will be created in the process, and we do not know the length of the proof nor the overall size of the computation. Hence, there is just no way of mapping that relational computation into any neural model with a fixed number of inputs, neurons in each layer, and overall depth, which is, however, the core design pattern in current deep learning.

[2] quotation from the abstract of "Learning Deep Architectures for AI" by Y. Bengio [3]

[3] We use the common term "relational logic" to refer to first-order logic without function symbols, which would unnecessarily complicate the computational perspective taken in this paper by leading further to infinite interpretations.

2.4. Modern NSI Systems

In recent years, a large number of new NSI systems building directly on top of popular deep learning frameworks emerged [8]. Many of these systems are designed to operate with some level of first-order expressiveness, seemingly resolving the old issue of propositional fixation [5] with the efficient, modern-day, tensor-based deep learning. However, it is instructive to remind ourselves that, from the computational perspective, this is in principle impossible to do. Specifically, many of the core relational (logic) reasoning tasks are provably hard, hence a neural-symbolic system that properly captures such expressiveness necessarily needs to adopt this computational complexity, too. Consequently, in one way or another, these efficient, modern neural-symbolic systems are necessarily dropping parts of the symbolic (relational) capabilities, often somewhat inconspicuously.

This issue with the relational logic expressiveness comes down to a problem commonly known as *variable binding* [6], which was not present in the propositional setting. A logical variable effectively represents an *unbound set* of objects, enabling to make general statements about different individuals, which facilitates the flexibility and variability in symbolic AI computations. In turn, this enables for systematic generalization over unbound structures, which is a core power of symbols, in contrast to the fixed-size vectors (tensors)[4] used in DL.[5] However, by building upon the modern DL foundation, one always needs to define some mapping from the possibly infinite variations of the relational inference structures into the necessarily finite, fixed-tensor computation of the existing DL frameworks [10, 11]. The choice of building upon this foundation certainly makes much sense from the practical perspective, alleviating researchers from the chore of implementing the differentiable calculus and optimizations, enabling for rapid prototyping of efficient NSI systems directly on top of existing (pre-trained) neural models [12]. However, there is a principled issue with any such approach based on fixed-size numeric tensor transformations in that these are inherently insufficient to capture the unbound structures of relational (logic) reasoning.

While this computational problem seems like an open (understudied) issue in NSI, it is intimately related to the long-studied problem of "propositionalization" [13][6] from the associated field of Relational Machine Learning [16].

3. Relational Machine Learning

We have got so used to pre-processing our data representations into the numeric vectors (tensors), expected at the input of almost any machine learning (ML) library, that it might be hard to see that this is not a universal representation. Indeed, vectors are highly appealing, since treating each data sample as an independent point in an $n-$dimensional space allows to directly adopt classic results from multivariate statistics [17] and the efficient computing machinery of linear algebra [18]. However, real-world data is not commonly stored in numeric vectors, but in the interlinked structures of the internet pages, knowledge graphs, and databases, which are inherently *relational* data representations.

3.1. Propositionalization

This representation dichotomy leads to a natural urge to turn these structures into the familiar form of the vectors, and continue with our standard ML pipelines [19], following the common cognitive bias of *"If all you have is a hammer, everything looks like a nail"*. In relational ML, this approach is generally referred to as "propositionalization" [13], which is an effective strategy that often works well in practice, however, it is conceptually limited.

Firstly, it is clear that one cannot map from unbound relational structures into the fixed-size vectors or tensors without (unwanted) loss of information. This obvious limitation is commonly addressed by limiting oneself to a fixed-size domain, seemingly bounding the maximum size of the resulting (neural) computation [20]. However, even if we restrict ourselves to fixed-size structures, designing a suitable learning representation in the form of a numeric

[4] We do not make much differences between vectors and tensors since, from the representation expressiveness point of view, they are the same.

[5] While vector representation may provide significant information-storing capacity, it is not unbound, unless relying on infinite precision numbers [9], which is completely impractical for any learning purposes.

[6] A similar term of "tensorization" has been used in NSI [14, 15], too.

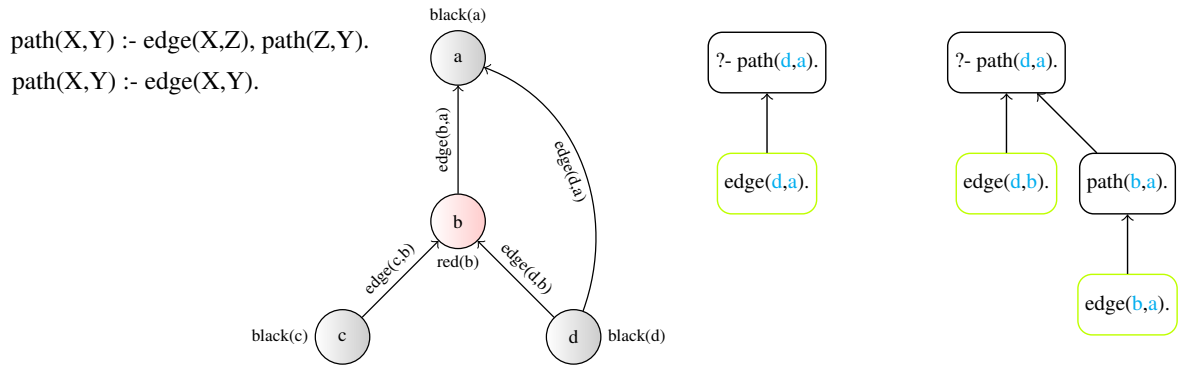

Fig. 2. An example learned ILP model of a path (left), being unfolded on a labeled graph sample encoded in relational logic (middle), resulting into *two different* computation graphs for the same target query "path(d, a)" (right).

vector or tensor is still deeply problematic. Let us take even just the graph data structures, which are a restricted form of relational data. If there was a definite way to map a graph into the standard learning form of a (fixed-size) numeric vector (or tensor), it would just trivially solve the graph isomorphism problem,[7] and any mapping that ignores the isomorphism symmetry will naturally break the underlying logical semantics that is invariant to it.

Of course, one can simply decide to ignore this problem and choose one of the plethora of the (NSI) approximations. However, if we want to address this learning problem properly, it is instructive to leave the space of numeric vector (tensor) transformations, and see how it can be addressed in the original space of symbolic logic.

3.2. Inductive Logic Programming

Much out of the lights of the ML mainstream, there has been a community of Inductive Logic Programming (ILP) [21], concerned with proper ways of learning (interpretable) models from exactly such relational data and knowledge representations. For decades [22] this, rather unorthodox, relational ML approach has been the premier venue for learning with data that do not succumb themselves to the standard form of i.i.d. feature vectors or tensors. This allowed ILP to properly explore some very general learning problems involving variable structures, beyond the scope of standard statistical ML based on the numeric vectors. For illustration, let us see how the graph-structured learning problems can be approached with the relational logic representations. To represent a graph, we simply define a binary "*edge*" relation, with a set of instantiations $edge(x, y)$ for all the adjacent nodes x, y. Additionally, we may also use other (unary) relations to assign attributes to the sets of nodes, such as $red(x)$, etc.

The models are then represented with *relational* logic rules, which may learn most various things, from characteristic substructures in molecules to paths in a network. Thanks to the high expressiveness, declarative nature, and inherent use of recursion in relational logic, the learned models are then often very compact and elegant (Fig. 2). For instance, a (learned) relational model *correctly* capturing paths in a graph, can be easily learned with a basic ILP system from but a few examples. The simplicity of this problem within the relational representation formalism of ILP is then in direct contrast to the neural models operating on the propositional foundation of fixed-size vectors (tensors). There, in order to properly embed the same problem into the hypothesis space of a classic, fixed neural architecture, we would generally need to restrict ourselves to fixed-size graphs, design an exponentially large network to capture all the possible (inference) paths, in which it would be close to impossible for gradient descent to find the correct solution, and we would still not be able to properly generalize to different (larger) graphs. Moreover, the learning difficulties remain even if we move to theoretically expressive (Turing-complete) neural architectures, such as the "Differentiable Neural Computer" from Deepmind [23], which required an extreme number of examples and additional hacking (e.g., pruning out invalid path predictions) to approximate the very same path-finding task through the standard (propositional) tensor representation, emulating a "differentiable memory" [24].

[7] Since to test if two graphs are isomorphic or not, it would suffice to turn them into such vectors and compare these for equality instead.

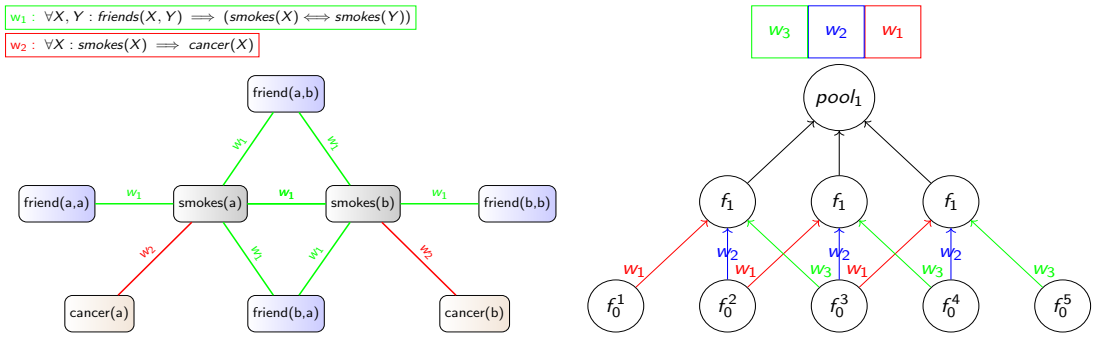

Fig. 3. Illustration of two forms of relational expressiveness, with CNNs (right) capturing a spatial pattern of a 3-neighborhood, and an MLN (left) capturing a social pattern of smoking habits amongst friends, both reflected by the symmetries in the respective ground model computations.

While ILP is able to correctly capture the expressiveness of relational logic, it is not well suited for dealing with noise and uncertainty, commonly present in real-world learning tasks. To address that issue, a number of methods arose to merge ILP with statistical learning under the notion of Statistical Relational Learning (SRL) [25]. Generally, there have been two major streams of approaches in SRL — (i) probabilistic logic programs [26] and (ii) lifted graphical models [27]. While the former builds heavily on the classic, symbolic ILP formalism, the latter is much closer to standard statistical models, the expressiveness of which it "lifts" from the propositional into the relational setting, while retaining proper logical semantics. Naturally, this is highly relevant to the aforementioned problem of propositional fixation of neural networks and relational-level NSI (Sec. 2.3).

3.3. Lifted Graphical Models

As opposed to standard ("ground") machine learning models, such as neural networks, *lifted* models do not specify a particular computation structure, but rather a logical template from which the standard models are being unfolded as part of the inference (evaluation) process, given the varying context of the relational input data. For instance, the (arguably) most popular lifted model — a Markov Logic Network (MLN) [28] may be seen as such a template for the classic Markov networks. For example, such an MLN template may express a general prior that "*friends of smokers tend to be smokers*" and "*smoking may lead to cancer*". The learning data may then describe a social network of people with a subset of smokers labeled as such. The lifted modeling paradigm of MLNs then allows to induce the smoking probabilities of all the other people based on their social relationships, as if modeled by a regular Markov network, yet correctly generalizing over social networks of *diverse structures and sizes*, overcoming the propositional fixation of the base Markov model.

The crucial ability to capture the possibly infinite variability of the relational computation structures with a finite computation template necessarily induces some form of repeated regularities, i.e. *symmetries*, within any such unfolded computation. In relational logic, these symmetries stem from the use of logical variables within the relations, i.e. subsets of the Cartesian products defined over the respective sets of objects. That is, unless the computation is effectively propositional as a corner case, one should be able to observe these repeated computational substructures, including shared parameterization, in the unfolded computation of the (ground) model. And while the, relational logic-based, lifted graphical models make this observation very explicit, we can observe the same principle being effectively exploited in many successful DL architectures, too, such as the popular CNNs, exploiting the symmetry of translational equivariance.[8] From this computational view, the CNNs can thus be seen as a particular instance of an (implicit) relational modeling template (Fig. 3).

The proper logical semantics of the lifted graphical models is, however, redeemed by computational difficulties. Due to the explicit treatment of the logic variables, these models are commonly viewed as computationally expensive. Nevertheless, from the perspective of a general NSI system, this view is somewhat misleading since the

[8] or invariance with the use of the pooling operator

computation is exactly as complex as necessary to properly capture the respective relational (logic) expressiveness.[9] However, the problematic factor is the *structure* of the resulting computation.

4. Structured Computation Graphs

For the sake of delving deeper into the (relational) computing structures in NSI, let us now define, a bit more formally, the common DL abstraction of a *computation graph*. A computation graph $\mathrm{G} = (\mathcal{N}, \mathcal{E}, \mathcal{F}, \mathcal{W})$, composed of nodes $\mathcal{N}$, edges $\mathcal{E}$, activation functions $\mathcal{F}$, and parameters $\mathcal{W}$, is a general way to represent nested parameterized functions using the language of graph theory. The neural networks are then commonly conveyed by the means of directed, differentiable, data-flow graphs where the data, commonly represented as the numeric vectors (or tensors) $\boldsymbol{x}^k$, "flow" through the directed edges $e \in \mathcal{E}$ while being successively transformed by the $\mathcal{W}$-parameterized, differentiable activation functions $f \in \mathcal{F}$ associated with the nodes $N \in \mathcal{N}$, commonly referred to as "neurons".

4.1. Basic Deep Learning

By far the most common computation paradigm in deep learning is to structure the computation graphs regularly into homogeneous "*layers*", which are used to refer to the sets of neurons $\{N \mid \mathsf{depth}_{\mathrm{G}}(\mathsf{N}) = \mathsf{k}\}$ residing at the same depth k in G. An input layer $k = 0$ is then commonly used to represent the feature values $\boldsymbol{x}$ of the input data samples $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ themselves. An output layer $k = \mathsf{depth}(\mathrm{G})$ then corresponds to the target values $\boldsymbol{y}$. The arguably most popular computation structure is then a "fully-connected" design pattern, where the interconnections between nodes in subsequent layers k and $k+1$ follow $N^k, N^{k+1} : (N^k, N^{k+1}) \in \mathcal{E}$, forming a complete bipartite graph. Moreover, each edge here is associated with a unique weight as $\mathcal{E} \xrightarrow{1:1} \mathcal{W}$. Consequently, assuming the common vector form of the input data samples $\boldsymbol{x}_i$, the computation can be efficiently reduced to a linear series of full (dense) matrix W_k^{k+1} multiplications, each followed by an element-wise application of some non-linear function f^{k+1}.

4.2. Relational Neural Models

The regular, dense, and homogeneous structure of the computation graph G, mapping to the basic matrix (tensor) operations, is a salient feature of the basic neural models. Note that from the aforementioned perspective of the lifted (graphical) models from SRL (Sec. 3.3) such computation, lacking any symmetries, is effectively propositional. However, there are also neural models designed to operate with forms of relational data structures, such as sequences, trees and graphs, which necessarily need to reflect some level of relational expressiveness. As a consequence, following the SRL reasoning from Sec. 3.3, their computation will, in contrast to the basic neural models, need to reflect some type of sparsity, heterogeneity, or irregularity, within the scope of the necessary symmetries reflected by some form of regular weight-sharing patterns within the graphs. Let us inspect some common representatives of these neural models.

A *Recursive* Neural Network (RecNN) [29, 30] is a neural architecture the computation graphs G_i of which directly follows the structure of each input example x_i, which takes the form of a $k-$regular *tree*. This enables to learn neural networks *directly* from differently-structured regular tree examples x_i, as opposed to using some transformation into the fixed-size tensors $\boldsymbol{x}$ (Sec. 3.1).[10] Here, the (vector) representations of every c leaf nodes $x^j, \ldots, x^{j+c}$ with the same parent node N_j^1 in the respective computation tree G_i are consequently combined by a given $\boldsymbol{c}$-parameterized operation c, such as a $\boldsymbol{c}$-weighted dot-product, to compute the representation of the parent N_j^1. This combining operation c then continues recursively for all the interior nodes, until the representation for the root node $N^{k=d}$ is computed, which forms the output of the model for x_i. Similarly to the convolution in CNNs, this parameterized combining operation c over the children nodes is shared across the tree [31] however, differently from CNNs, the resulting computation graph can be highly and *irregularly sparse* (Fig. 4).

[9] For instance, there are many useful relational templates (patterns) the computational complexity of which is simply *linear*, such as the CNNs and almost all other weight-sharing neural models.

[10] which can also be seen as graphs with completely regular grid topologies

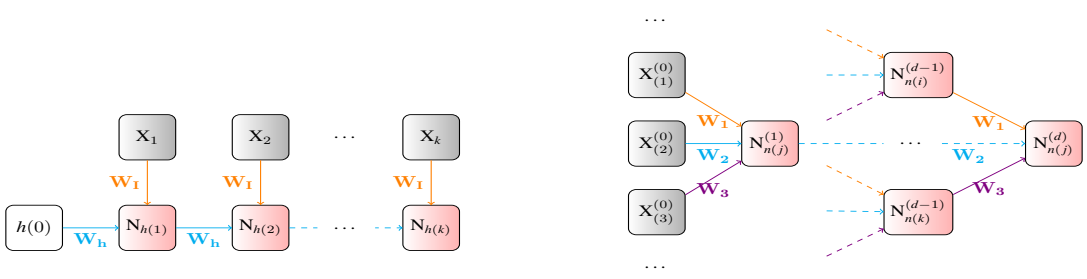

Fig. 4. The symmetric weight-sharing patterns in the structured computation graphs of recurrent (left) and *recursive* (right) neural models.

Note that the basic form of the more common *Recurrent* Neural Networks (RNNs) [32] can then be seen as a restriction of the RecNN computation to *sequential structures* x_i (Fig. 4). There, the computation graph G in the form of a linear binary tree is successively unfolded along the input sequence to compute the hidden representation for each node N_i based on the previous node's N_{i-1} representation, and the current input features $\boldsymbol{x_i}$.

The most popular recent addition in this category are then Graph Neural Networks (GNN) [33, 34] which, in contrast to the RNNs, can be seen as an extension of RecNNs to completely *irregular graph data* $x_i = \{\mathcal{N}_i, \mathcal{E}_i\}$. For that purpose, they unfold the computation graph G_i structure from each input structure x_i, similarly to the RecNNs. However, a GNN still follows the standard layered approach (Sec. 4.1), where the structure of *each layer* k exactly follows the structure of the *whole* input graph x_i . For computation of the next layer $k+1$ representations of the nodes in G_i, each node N calculates its own value $h(N)$ by *aggregating* A ("pooling") the (vector) values of the nodes $M : (N, M) \in \mathcal{E}_i$ adjacent in the input graph x_i ("message passing"), transformed by some parametric function C_{W_1} ("convolution"), which is again being reused with the same parameterization W_1^k within each layer k (Fig. 5). However, in contrast to RecNNs, a different parameterization is typically used at each layer.

5. Hardware and Software Lottery

It is instructive to note again the RecNNs architecture, which can be seen as a strict generalization of the common RNNs, adding significant flexibility by exploiting also the sentence structures in NLP, while retaining comparable computation complexity.[11] Moreover, in contrast to the RNNs, the RecNNs can, in principle, be parallelized, thanks to the recursive decomposition. Nevertheless, they have never reached anywhere near the popularity of the simpler RNNs, as they were notoriously difficult to effectively implement in the mainstream, fixed-size tensor-based deep learning frameworks.[12] A similar story then followed with the subsequent replacement of the RNNs with Transformers [36], the core "attention" concept of which can be seen simply as a restriction of the GNN computation to fixed-size, fully connected graphs. This hardwired restriction removes the relational flexibility w.r.t. the GNNs and restricts the length of the input sequence w.r.t. the RNNs, while also substantially increasing the computation complexity from linear to quadratic. Nevertheless, it is exactly these apparent drawbacks that provide the architecture one crucial advantage - it is perfectly aligned with the current HW stack, particularly the GPUs.

The GPU architecture is inherently based on the "single instruction, multiple data" (SIMD)[13] concept [37], enabling for high-throughput data-parallel processing of dynamic values through static operations. With neural networks, this generally means static, regular, and dense structure of the computation graph, which maps perfectly to the homogeneous, fixed-size tensor processing underlying the mainstream deep learning architectures and frameworks. On the contrary, the dynamic, sparse, irregular, and heterogeneous nature of the discussed (lifted) neural

[11] depending on the specific parse/dependency tree representation, but generally $O(n \cdot log(n))$ at worst

[12] A notable, yet largely unsuccessful, attempt to improve this situation was, e.g., Tensorflow Fold [35].

[13] A bit more precise term with modern GPUs is SIMT, combining SIMD with multi-threading which, however, shares the same limitation as all the threads effectively execute the same logic.

models, reflecting some level of relational expressiveness, is highly problematic to this mainstream regime of compute. And the situation is further considerably worse with the symbolic (logic) computation (Sec. 3.2), commonly relying on conditional branching and looping (recursion).

5.1. Symbolic computation

The maturity of the existing HW&SW DL ecosystem naturally incentivizes NSI researchers away from the expressive symbolic computation towards "tensorization" of their systems [14], even in cases where such a choice would otherwise be conceptually inappropriate, which has a self-reinforcing side-effect of regressing to the same models [38]. This is becoming noticeable with more and more NSI methods operating simply as modules on top of existing tensor-based frameworks [39]. However, the symbolic computation, as exploited in the classic (relational) AI tasks, is often inherently sequential, stateful, and conditional, leading to dynamically sized and structured computation graphs, the forms of which are just impossible to properly embed into the common fixed-size tensor processing on GPUs.

For instance, the computation of a relational (lifted) model (Sec. 3.3) may completely change from sample to sample, making it principally incompatible with the SIMD philosophy of the GPU architecture. Importantly, this does not mean that the relational (symbolic) compute strategy would itself be somehow inferior, or impossible to accelerate. On the contrary, akin to batching in DL, one can always naively parallelize in the data dimension across multiple CPU threads, utilizing its "multiple instructions, multiple data" (MIMD) capabilities. Moreover, the symbolic model representation also offers a much broader range of interesting acceleration strategies, such as lifted inference [40], in contrast to the comparatively brute-force approach of the GPU-based DL.

5.2. Accelerating Graph Neural Networks

An interesting borderline example between these two, largely separate, worlds of symbolic and neural compute are the aforementioned GNN models, which can be seen as an instance of deep learning evolving in the direction of relational learning. Since the GNNs are able to (partially) capture graph-structured data, they necessarily reflect some level of (symbolic) relational logic expressiveness. Particularly, their expressive power corresponds to a guarded fragment of C2 logic [41, 42].[14] Similarly to the other relational models, this increased, albeit still relatively limited, expressiveness already takes its toll in their compute structure. Particularly, the variability of the input graph data, reflected in the irregularly sparse interconnection patterns within the GNN layers, is problematic for the classic GPU acceleration.

However, the significant rise in popularity of the GNN-like architectures, driven by the omnipresence of the graph data structures in real-world problems, has come with an interesting paradigm shift in which the SW and HW developers started to actively reflect back on these alternative computing concepts in AI, too. Consequently, there have been new SW frameworks emerging [43, 44], and most of the latest GPU cards support some levels of sparsity [45]. This is an interesting step in a good direction for NSI [46], however, it is still far form a proper foundation for the symbolic compute, since the GNNs are considerably limited w.r.t. the (relational) logic expressiveness [41, 42], which can never fully fit the SIMD design of the GPUs. The corresponding low-level optimizations here are still largely incremental, building heavily on the legacy of the matrix-multiplication-based thinking. Moreover, they tend to be increasingly specific, e.g., enforcing particular, fixed sparsity patterns [45], or even containing hardwired acceleration for particular models, such as the Transformers [47].

In contrast, the high-level, HW-independent, symbolic acceleration techniques are much more principled and reusable. For instance, in the spirit of NSI, the symbolic techniques of lifted inference (Sec. 3.3) can be transferred into deep learning to effectively accelerate relational neural models, such as the GNNs, by exploiting the inherent symmetries contained in their computation graphs. This results into explicitly smaller neural computation that effectively performs the exact same function (Fig. 5), which was unprecedented in deep learning [48]. Consequently,

[14] Relational logic with neighbor "guards", counting quantifiers, and at most 2 variables [41].

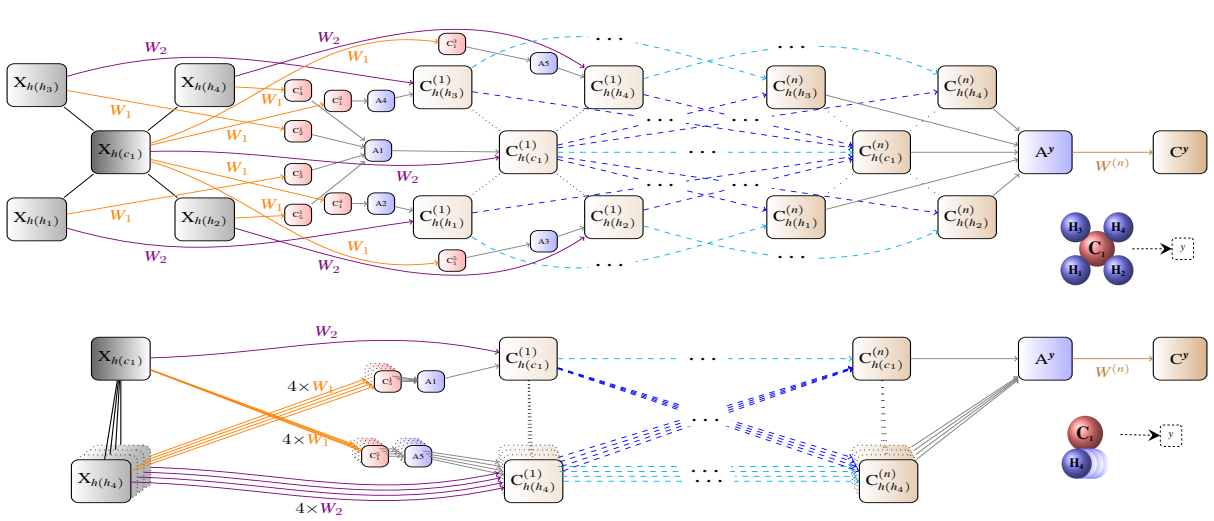

Fig. 5. Compressing a typical GNN model computation (up) based on the contained computation graph symmetries, resulting into a much smaller yet functionally equivalent computation (down), in the spirit of (symbolic) lifted inference [48].

using this symbolic technique, even a naive CPU implementation of a GNN model can become faster than a specialized GPU-accelerated implementation [48].[15] However, should we completely fixate on the GPU-based deep learning as the foundation, our possibilities for such a beneficial transfer of symbolic techniques into deep learning (and vice-versa) might gradually become extinct.

6. Conclusion - A Vision of Neuro-Symbolic HW

In the past, AI researchers used to design their algorithms independently of the underlying HW, relying on the universal capabilities of the CPU architecture, and its all-encompassing acceleration through the Moore's law [49]. However, in this new era of AI compute, an inadequate alignment with the existing specialized HW can easily turn conceptually strong ideas upside down. The prospect of the GPU acceleration, together with the mature DL SW ecosystem built upon that foundation, naturally incentivizes researchers to adjust their ideas to this mainstream regime of compute in order to reap its benefits. However, as argued throughout this position paper, this choice has important consequences, especially for NSI.

While the SIMD architecture of the GPUs is a perfect fit for the current, tensor-based neural models, the principles of the symbolic methods are closely connected to the universal flexibility of the CPUs. The vision of a proper, fully expressive *and efficient* NSI computation thus calls for a much tighter integration between the CPU and GPU features than currently present. Instead of incremental improvements of the, principally limited, SIMD approach, for NSI it would be much more meaningful to transfer the MIMD principles from CPUs onto a significantly more parallel, high-throughput, low-latency foundation.

While the AI industry is currently dominated by the GPU-based applications of DL with largely predictable business potential, it is highly unlikely that we will see the same amount of interest in acceleration of the, more demanding, symbolic computing. The NSI community is just too small of a niche to attract the astronomical investments needed for development of such a new HW architecture. However, with the rapid growth in popularity of the GNN models, reflecting some of the computing demands of the symbolic methods, we start to see first signs of the needed HW convergence with newly emerging architectures, such as the Intelligence Processing Unit (IPU) [50], finally breaking beyond the SIMD legacy of the GPUs, and currently leading the GNN performance benchmarks [51]. While still far from meeting all the compute demands of a general NSI system, it might be very interesting for NSI researchers to watch this emerging stream of novel computing architectures.

[15] Particularly in relational domains with highly symmetric graph structures, such as molecular chemistry.

References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., Language models are few-shot learners, *Advances in neural information processing systems* **33** (2020), 1877–1901.

[2] W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *The Bulletin of Mathematical Biophysics* **5**(4) (1943), 115–133. ISBN 0007-4985.

[3] Y. Bengio, Learning Deep Architectures for AI, *Foundations and Trends in Machine Learning* **2**(1) (2009), 1–127. ISBN 2200000006.

[4] G.G. Towell, J.W. Shavlik and M.O. Noordewier, Refinement of approximate domain theories by knowledge-based neural networks, in: *Proceedings of the eighth National conference on Artificial intelligence*, Boston, MA, 1990, pp. 861–866.

[5] J. McCarthy, Epistemological challenges for connectionism, *Behavioral and Brain Sciences* **11**(1) (1988), 44–44.

[6] S. Bader and P. Hitzler, Dimensions of Neural-symbolic Integration - A Structured Survey, *arXiv preprint* (2005).

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

[8] P. Hitzler and M.K. Sarker, Neuro-symbolic artificial intelligence: The state of the art (2022).

[9] S. Hölldobler, Y. Kalinke and H.-P. Störr, Approximating the semantics of logic programs by recurrent neural networks, *Applied Intelligence* **11**(1) (1999), 45–58.

[10] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., Tensorflow: A system for large-scale machine learning, in: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016.

[11] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga and A. Lerer, Automatic differentiation in pytorch (2017).

[12] I. Donadello, L. Serafini and A.D. Garcez, Logic tensor networks for semantic image interpretation, *arXiv preprint arXiv:1705.08968* (2017).

[13] M.-A. Krogel, S. Rawles, F. Železný, P.A. Flach, N. Lavrač and S. Wrobel, *Comparative evaluation of approaches to propositionalization*, Springer, 2003.

[14] A. Garcez, M. Gori, L. Lamb, L. Serafini, M. Spranger and S. Tran, Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning, *Journal of Applied Logics* **6**(4) (2019), 611–631.

[15] L. De Raedt, S. Dumančić, R. Manhaeve and G. Marra, From Statistical Relational to Neuro-Symbolic Artificial Intelligence, *arXiv preprint arXiv:2003.08316* (2020).

[16] L. De Raedt, *Logical and Relational Learning*, Springer, 2008.

[17] D. Haussler and M. Warmuth, *The probably approximately correct (PAC) and other learning models*, Springer, 1993.

[18] J.J. Dongarra, J. Du Croz, S. Hammarling and I.S. Duff, A set of level 3 basic linear algebra subprograms, *ACM Transactions on Mathematical Software (TOMS)* **16**(1) (1990), 1–17.

[19] M.V. Franca, G. Zaverucha and A. Garcez, Fast relational learning using bottom clause propositionalization with artificial neural networks, *Machine learning* **94**(1) (2014), 81–104.

[20] M. Vitor, M.V.M. França and A.S. d'Avila Garcez, Neural Relational Learning Through Semi-Propositionalization of Bottom Clauses, *2015 AAAI Spring Symposium in Knowledge Representation and Reasoning: Integrating Symbolic and Neural Approaches* (2015), 53–56. ISBN 1412920388.

[21] S. Muggleton and L. De Raedt, Inductive logic programming: Theory and methods, *The Journal of Logic Programming* **19** (1994).

[22] A. Cropper, S. Dumančić and S.H. Muggleton, Turning 30: New ideas in inductive logic programming, *arXiv preprint arXiv:2002.11002* (2020).

[23] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S.G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou et al., Hybrid computing using a neural network with dynamic external memory, *Nature* **538**(7626) (2016), 471–476.

[24] A. Graves, G. Wayne and I. Danihelka, Neural turing machines, *arXiv preprint arXiv:1410.5401* (2014).

[25] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning*, 2007, p. 586. ISBN 0262072882.

[26] L. De Raedt and A. Kimmig, Probabilistic (logic) programming concepts, *Machine Learning* **100** (2015), 5–47.

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

[28] M. Richardson and P. Domingos, Markov logic networks, *Machine learning* (2006).

[29] R. Socher, A. Perelygin, J.Y. Wu, J. Chuang, C.D. Manning, A.Y. Ng, C. 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)*, Vol. 1631, Citeseer, 2013, p. 1642.

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

[31] R. Socher, D. Chen, C.D. Manning and A. Ng, Reasoning with neural tensor networks for knowledge base completion, in: *Advances in neural information processing systems*, 2013.

[32] Z.C. Lipton, J. Berkowitz and C. Elkan, A critical review of recurrent neural networks for sequence learning, *arXiv preprint arXiv:1506.00019* (2015).

[33] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner and G. Monfardini, The graph neural network model., *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* **20**(1) (2009), 61–80.

[34] M.M. Bronstein, J. Bruna, Y. LeCun, A. Szlam and P. Vandergheynst, Geometric deep learning: going beyond euclidean data, *IEEE Signal Processing Magazine* **34**(4) (2017), 18–42.

[35] M. Looks, M. Herreshoff, D. Hutchins and P. Norvig, Deep learning with dynamic computation graphs, *arXiv preprint arXiv:1702.02181* (2017).

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser and I. Polosukhin, Attention is All you Need, *Neural Information Processing Systems* (2017).
[37] M.J. Flynn, Very high-speed computing systems, *Proceedings of the IEEE* **54**(12) (1966), 1901–1909.
[38] S. Hooker, The hardware lottery, *Communications of the ACM* **64**(12) (2021), 58–65.
[39] K. Ahmed, T. Li, T. Ton, Q. Guo, K.-W. Chang, P. Kordjamshidi, V. Srikumar, G. Van den Broeck and S. Singh, PYLON: A PyTorch framework for learning with constraints, in: *NeurIPS 2021 Competitions and Demonstrations Track*, PMLR, 2022, pp. 319–324.
[40] G. Van den Broeck, N. Taghipour, W. Meert, J. Davis and L. De Raedt, Lifted probabilistic inference by first-order knowledge compilation, in: *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, AAAI Press/International Joint Conferences on Artificial Intelligence; Menlo . . . , 2011, pp. 2178–2185.
[41] M. Grohe, The logic of graph neural networks, in: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, IEEE, 2021, pp. 1–17.
[42] G. Šourek, F. Železný and O. Kuželka, Beyond graph neural networks with lifted relational neural networks, *Machine Learning* **110**(7) (2021), 1695–1738.
[43] M. Innes, Flux: Elegant Machine Learning with Julia, *Journal of Open Source Software* (2018). doi:10.21105/joss.00602.
[44] M. Fey and J.E. Lenssen, Fast graph representation learning with PyTorch Geometric, *arXiv preprint arXiv:1903.02428* (2019).
[45] J. Choquette, W. Gandhi, O. Giroux, N. Stam and R. Krashinsky, NVIDIA A100 tensor core GPU: Performance and innovation, *IEEE Micro* **41**(2) (2021), 29–35.
[46] L.C. Lamb, A.S. d'Avila Garcez, M. Gori, M.O.R. Prates, P.H.C. Avelar and M.Y. Vardi, Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective, in: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, C. Bessiere, ed., ijcai.org, 2020, pp. 4877–4884.
[47] J. Choquette, Nvidia hopper h100 gpu: Scaling performance, *IEEE Micro* (2023).
[48] G. Šourek, F. Železný and O. Kuželka, Lossless Compression of Structured Convolutional Models via Lifting, *International Conference on Learning Representations* (2021).
[49] R.R. Schaller, Moore's law: past, present and future, *IEEE spectrum* **34**(6) (1997), 52–59.
[50] Z. Jia, B. Tillman, M. Maggioni and D.P. Scarpazza, Dissecting the graphcore ipu architecture via microbenchmarking, *arXiv preprint arXiv:1912.03413* (2019).
[51] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong and J. Leskovec, Ogb-lsc: A large-scale challenge for machine learning on graphs, *arXiv preprint arXiv:2103.09430* (2021).