Deep Symbolic Learning: Discovering Symbols and Rules from Perceptions

Alessandro Daniele[1] , **Tommaso Campari**[1, 2] , **Sagar Malhotra**[1, 3] and **Luciano Serafini**[1]
[1]Fondazione Bruno Kessler, Trento, Italy,
[2]Università degli Studi di Padova, Italy
[3]TU Wien, Austria
{daniele, tcampari, smalhotra, serafini}@fbk.eu

Abstract

Neuro-Symbolic (NeSy) integration combines symbolic reasoning with Neural Networks (NNs) for tasks requiring perception and reasoning. Most NeSy systems rely on continuous relaxation of logical knowledge, and no discrete decisions are made within the model pipeline. Furthermore, these methods assume that the symbolic rules are given. In this paper, we propose *Deep Symbolic Learning* (DSL), a NeSy system that learns *NeSy-functions*, i.e., the composition of a (set of) *perception functions* which map continuous data to discrete symbols, and a *symbolic function* over the set of symbols. DSL simultaneously learns the perception and symbolic functions while being trained only on their composition (NeSy-function). The key novelty of DSL is that it can create internal (interpretable) symbolic representations and map them to perception inputs within a differentiable NN learning pipeline. The created symbols are automatically selected to generate symbolic functions that best explain the data. We provide experimental analysis to substantiate the efficacy of DSL in simultaneously learning perception and symbolic functions.

1 Introduction

Neuro-Symbolic (NeSy) Systems combine deep neural networks and symbolic reasoning so that learning and reasoning can occur in a symbiotic fashion. The fundamental goal of NeSy systems is to incorporate and potentially learn the symbolic rules while still exploiting neural networks (NNs) for interpreting perception and guiding exploration in the combinatorial search space. In general, a NeSy system can be seen as a composition of *perception functions* and *symbolic functions*. Perception functions map perception, usually represented as real-valued tensors, to symbols, whereas symbolic functions map symbols to other symbols. The first challenge to any NeSy system is to reconcile the dichotomy between the intrinsically discrete nature of symbolic reasoning and the implicit continuity requirement of gradient descent-based learning methods. Recent works have tried to resolve this problem by exploiting different types of continuous relaxations to logical rules. However, with few exceptions, most such works assume the symbolic functions to be given a priori, and they use these functions to guide the training of a perception function, parameterized as a NN. A key challenge to such systems is the lack of a method capable of performing symbolic manipulations and meaningfully associating symbols to perception inputs, also known as the *Symbol Grounding Problem* [Harnad, 1990].

In this paper, we introduce the concept of *NeSy-function*, i.e., a composition of a set of perception and symbolic functions. Moreover, we propose *Deep Symbolic Learning (DSL)*[1], a framework that can jointly learn perception and symbolic functions while supervised only on the NeSy function. This is done by introducing policy functions, similar to Reinforcement Learning (RL) [Sutton and Barto, 1998], within the neural architecture. The policy function chooses internal symbolic representations to be associated with the perception inputs based on the confidence values generated by the neural networks. The selected symbols are then combined to form a unique prediction for the NeSy function, while their confidences are interpreted under fuzzy logic semantics to estimate the confidence of such a prediction. Moreover, DSL can learn symbolic functions by applying the same policy to select their outputs. The key contributions of DSL are:

- *Learning the symbolic and the perception function through supervision only on the NeSy function*. To the best of our knowledge, DSL is the first NeSy system that can simultaneously learn symbolic and perception functions in an end-to-end fashion, from supervision only on their composition and with minimal biases on the symbolic function. It has been shown that previous such claims [Wang *et al.*, 2019] contained some form of label leakage leading to supervision on the individual perception functions [Chang *et al.*, 2020], and the system completely fails (with 0% accuracy on visual-sudoku task) when supervision on the perception function is removed. Furthermore, later works on this idea rely on clustering-based pre-processing [Topan *et al.*, 2021] and do not constitute an end-to-end system.
- *Symbol Grounding Problem* (SGP) refers to the prob-

[1]Project webpage: https://dkm-fbk.github.io/projects/dsl.html

lem of associating symbols to abstract concepts without explicit supervision [Harnad, 1990] on this association. The SGP is considered a major prerequisite for intelligent agents to perform real-world logical reasoning. Recent works [Chang *et al.*, 2020] have also provided extensive empirical evidence on the non-triviality of this task, even on the simplest of problems. In DSL we can create internal (interpretable) symbolic representations that are then associated to perception inputs (e.g., handwritten digits) while getting supervision only on higher order operations (e.g., the sum of the digits). Furthermore, unlike previous works [Topan *et al.*, 2021], DSL does not rely on clustering based pre-processing. This is important as such pre-processing informs the system about the number of symbols, whereas DSL can infer such number and create meaningful associations between symbols and perception inputs.

- *Differentiable Discrete Choices.* DSL is the first NeSy architecture that provides a method for making discrete symbolic choices within an end-to-end differentiable architecture. It achieves this by exploiting a *policy function* that, given confidence values on an arbitrarily large set of symbols, is able to discretely choose one of them. Furthermore, the policy function can be changed to exploit varying strategies for the choice of symbols.

Finally, we provide extensive empirical verification of the aforementioned claims by testing DSL on three different tasks. Firstly, we test our system on a variant of the MNIST [LeCun *et al.*, 1998] Sum task proposed in [Manhaeve *et al.*, 2018], where the knowledge about the sum operation is not given but learned (see Example 1). Moreover, we also test DSL with no prior information on the number of required internal symbols, showing that DSL can correctly associate them with perception inputs while learning the summation rules. DSL provides competitive results, even in comparison to systems that exploit prior knowledge.

Finally, in the last two experiments, we test a recursive variant of DSL on the Visual Parity (see Example 2) and the Multi-digit Sum tasks (see Example 3). In these experiments, DSL shows great generalization capabilities. Indeed, we trained it on short sequences, finding that it can generalize to sequences of any length.

2 Related Works

NeSy has emerged as an increasingly exciting field of AI, with several directions [Besold *et al.*, 2021]. Approaches like Logic Tensor Networks [Badreddine *et al.*, 2022] and Semantic-Based Regularization [Diligenti *et al.*, 2017] encode logical knowledge into a differentiable function based on fuzzy logic semantics, which is then used as a regularization in the loss function. Semantic Loss [Xu *et al.*, 2018], also aims at guiding the NN training through a logic-based differentiable regularization function based on probabilistic semantics, which is obtained by compiling logical knowledge into a Sentential Decision Diagram (SDD) [Darwiche, 2011]. In comparison to DSL these approaches assume that the symbolic function is already given and is not learned from data. Furthermore, the symbolic function is only used to guide the learning of the perception function and does not influence the NN predictions at test time.

A parallel set of approaches incorporates NN's as atomic inputs to the conventional symbolic solvers. DeepProbLog [Manhaeve *et al.*, 2018], a neural extension to ProbLog [Bruynooghe *et al.*, 2010], admits neural predicates that provide the output of an NN, interpreted as probabilities. The system then exploits SDDs enriched with gradient semirings to provide an end-to-end differentiable system for learning the NN and the program parameters simultaneously. Recent works have aimed at providing similar neural extensions to other symbolic solvers. DeepStochLog [Winters *et al.*, 2022] and NeurASP [Yang *et al.*, 2020] provide such extensions to Stochastic Definite Clause Grammars and Answer Set Programming respectively. In comparison to the regularization-based approaches, these approaches are able to exploit the symbolic function at inference time. However, they also assume the symbolic function to be given. NeSy methods like NeuroLog [Tsamoura *et al.*, 2021], ABL [Dai *et al.*, 2019] and ABLSim [Huang *et al.*, 2021] are based on abduction-based learning framework, where the perception functions have supervision on assigning symbolic labels to perception data. However, the reasoning framework provides additional supervision to make the perception output consistent with the knowledge base i.e., the symbolic function. An abduction based approach closely related to our work is MetaAbd [Dai and Muggleton, 2021] where latent symbols are associated to perception inputs, while simultaneously learning a logical theory and latent symbols based on a probability-based score function. However, MetaAbd assumes given knowledge of a series of primitives and learns their composition, while DSL fixes the compositional structure and learns the primitives. Furthermore, MetaAbd samples the space of logical hypothesis, whereas DSL is an E2E differentiable framework learning logical theories and perception symbols within the same differentiable pipeline. Apperception Engine [Evans *et al.*, 2021] also aims at learning logical theories from raw data. However, when raw data is continuous, i.e., consists of a perception-based tasks like recognizing images, they use pre-trained NNs. Hence, unlike DSL, Apperception Engine cannot simultaneously learn to create symbols for perception inputs and learn logical theories on those symbols.

Another paradigm of NeSy integration consists of works that aim at learning the symbolic function, with either no perception component or with supervision on the perception function. Neural Theorem Prover [Rocktäschel and Riedel, 2017] uses soft unification to learn symbol embeddings to correctly satisfy logical queries. Logical Neural Networks [Riegel *et al.*, 2020] is a NeSy system that creates a 1-to-1 mapping between neurons and elements of a logical formulae. Hence, treating the entire architecture as a weighted real-valued logic formula. SATNet [Wang *et al.*, 2019] aims to learn both the symbolic and perception functions. It does so by encoding MAXSAT in a semi-definite programming based continuous relaxation, and integrating it into a larger deep learning system. However, it has been shown that it can only learn the symbolic function when supervision on perception is given [Chang *et al.*, 2020]. [Topan *et al.*, 2021] extends SATNet to learn perception and symbolic functions, aiming at

resolving the symbol grounding problem in SATNet. This extension relies on a pre-processing pipeline that uses InfoGAN [Chen *et al.*, 2016] based latent space clustering. Besides not being end-to-end, their method assumes that the number of symbols (i.e., the number of digits in their experiments) is given apriori. [Aspis *et al.*, 2022] is another approach that exploits latent space clustering for extracting symbolic concepts. Furthermore, they assume the number of symbols and the logical rules to be given apriori. In DSL, only an upper bound needs to be provided on the number of required symbols. If the amount of symbols provided is higher than the correct one, it learns to ignore the additional symbols, mapping the perceptions to only the required number of symbols.

Most of the NeSy systems in literature have distinct perception and symbolic components. To the best of our knowledge, none of these systems can learn both the components from supervision provided only on their composition. In DSL, we provide an approach that is able to learn both the symbolic functions and the perception functions separately, from supervision only on their composition. Furthermore, the symbols required to create the rules are created internally and are associated to perception within a unique NN learning pipeline. In comparison to the SOTA NeSy methods, DSL is the first end-to-end NeSy system to resolve a non-trivial instance of both the symbol grounding problem and rule learning from perception.

3 Background

Notation. We denote sets with math calligraphic font and its elements with corresponding indexed lower case letter, e.g., $\mathcal{S} = \{s_i | \forall i \in \mathbb{N}, 0 < i \leq k\}$, where $k = |\mathcal{S}|$ is the cardinality of the set. Tensors are denoted with capital bold letters (e.g. $\mathbf{G}$) and the $[\dots]$ operator is used to index values in a tensor. For instance, given a matrix $\mathbf{G} \in \mathbb{R}^{10\times 10}$, the element $\mathbf{G}[1,2]$ corresponds to the entry in row 1 and column 2 of $\mathbf{G}$. Similar to python syntax, we introduce the colon symbol for indexing slices of a tensor. As an example, $\mathbf{G}[1,:]$ corresponds to the vector $\langle \mathbf{G}[1,1], \dots, \mathbf{G}[1,10] \rangle$. We use a bar on top of functions and elements of a set to denote a tuple of functions and elements, respectively. For instance, $\bar{s} = \bar{f}(\bar{x})$ is equivalent to:

$$(s_1, \dots, s_n) = (f_1(x_1), \dots, f_n(x_n))$$

Note that the length n of the tuple is omitted from the bar notation since it will always be clear from the context.

Fuzzy Logic. Fuzzy Logic is a multi-valued generalization of classical logic, where truth values are reals in the range $[0, 1]$. In this work, we will only be dealing with conjunctions, which in fuzzy logic are interpreted using *t-norms*. A t-norm $t : [0,1] \times [0,1] \to [0,1]$ is a function that, given the truth values t_1 and t_2 for two logical variables, computes the truth value of their conjunction. In this paper, we will exploit Gödel t-norm, which defines the truth value of a conjunction as the minimum of t_1 and t_2.

Problem Definition. Our approach to NeSy can be abstractly described as the problem of jointly learning a set of perception and symbolic functions providing supervision only on their composition. We define $\mathcal{X}$ to be the space of possible perception inputs. Given a finite set $\mathcal{S}$ of discrete symbols, a *perception functions* $f : \mathcal{X} \to \mathcal{S}$ maps from $\mathcal{X}$ to symbolic output in $\mathcal{S}$. We define a *symbolic function*, $g : \mathcal{S}^n \to \mathcal{S}$, that maps an n-tuple of symbols to a single output symbol. We will also consider g with a typed domain, i.e., given some sets of symbols $\mathcal{S}_1, ..., \mathcal{S}_n$ and $\mathcal{S}$, g could map from $\mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ to $\mathcal{S}$. Finally, we define a *NeSy functions* $\phi : \mathcal{X}^n \to \mathcal{S}$ as a composition of perception and symbolic functions. In this paper, we will provide supervision only on the NeSy-function through a training set Tr of the form $Tr = \left\{ \left(\bar{x}_i, y_i\right) \right\}_{i=1}^{m}$, where $y_i = \phi(\bar{x}_i)$ and m is the dimension of the training set. The goal is learning both the NeSy function and its components. NeSy-functions can constitute arbitrary compositions of symbolic and perception functions. In this paper we consider two such cases, namely *Direct NeSy function*, and *Recurrent NeSy function*.

Definition 1 (Direct NeSy function)**.** *Let $g : \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \to \mathcal{S}$ be a symbolic function and $f_i : \mathcal{X} \to \mathcal{S}_i$, for $i = 1, \dots, n$ be n perception functions. A* Direct NeSy-function *is defined as the composition of g with the f_i*

$$\phi(x_1, \dots, x_n) = g(f_1(x_1), \dots, f_n(x_n)) \tag{1}$$

Example 1 (Sum task)**.** *Let $\mathcal{S}_1$ and $\mathcal{S}$ be the following set of symbols: $\mathcal{S}_1$ are the integers from 0 to 9 and $\mathcal{S}$ is the set of integers from 0 to 18. Let us have a training set, consisting of tuples (x_1, x_2, y) where x_1 and x_2 are images of handwritten digits and y is the result of adding the digits x_1 and x_2. Our goal is to learn the Direct NeSy-function:*

$$\phi(x_1, x_2) = g(f(x_1), f(x_2))$$

where f is handwritten digit classifiers. Hence, our goal is to learn f and g with supervision provided only on $g(f(\bar{x}))$.

As a second type of composition we will consider Recurrent NeSy functions, i.e., NeSy functions defined recursively. In general, it is possible to define complex types of recurrent compositions involving multiple perception and symbolic functions. In this work, we focus on a few such possibilities.

Definition 2 (Simple Recurrent NeSy-function)**.** *Let $g : \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ be a symbolic function and $f : \mathcal{X} \to \mathcal{S}$ be a perception function. Moreover, it is given an ordered list of perceptions $X = \{x^{(k)}\}_{k=1}^{K}$, with $x^{(k)} \in \mathcal{X}$. We define $X^{(k)}$ as the sequence of first k elements of X. A* Simple Recurrent NeSy-function *ϕ is defined recursively as:*

$$\phi(X^{(k)}) = g(f(x^{(k)}), \phi(X^{(k-1)}))$$
$$\phi(X^{(0)}) = s^{(0)} \in \mathcal{S}$$

Example 2 (Visual Parity)**.** *Let $\mathcal{S} = \{s_0, s_1\}$ be a set composed of two symbols, representing binary values, and $\phi(X)$ the Simple Recurrent NeSy function which represents the parity function, i.e., the function that returns s_0 if the number of s_1 in the sequence is even, s_1 if it is odd. $\phi(X)$ can be expressed in terms of a perception function f and a symbolic function g using previous definition of Simple Recurrent NeSy function: the f converts the perceptions in binary values, while the g represents the XOR operator, with $s^{(0)} = s_0$.*

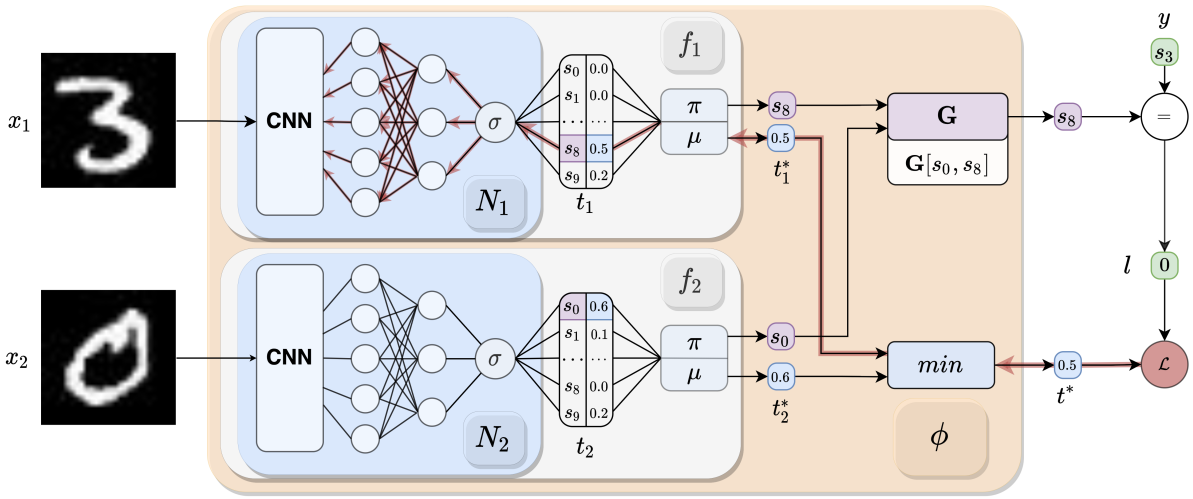

Figure 1: Architecture of Deep Symbolic Learning for the Sum task. Red arrows represent the backward signal during learning.

Example 3 (Multi-digit Sum)**.** *Let $\mathcal{S}$ be the set of symbols corresponding to the integers from 0 to 9, and $\mathcal{S}_c = \{s_0, s_1\}$ another set of symbols. We have a training set composed of pairs of multi-digit numbers and their sum as labels. Each number is represented by a list of MNIST digit images. The goal is to learn the NeSy function ϕ that computes the sum of the given numbers. Similarly to Example 2, the NeSy function can be defined recursively. However, in this case, there are two symbolic functions, which compute the single-digit summation modulo 10 and the carry, respectively.*

4 Method

Policy Functions. In this paper we will exploit the concept of *policy functions* inspired by Reinforcement Learning (RL). In RL, an agent has at its disposal a set of available actions, and at each time frame only one action can be performed. The goal is to select actions that maximize the expected reward. A strategy for choosing the actions, based on the current state of the system, is called a *policy*. In this work we consider two specific policies, namely the *greedy* and the *ϵ-greedy*, and we adapt them to the context of NeSy. In our setting, a *policy* selects a symbol instead of an action, and it is defined as a function $\pi : [0,1]^{|\mathcal{S}|} \to \mathcal{S}$ that, given a vector $t \in [0,1]^{|\mathcal{S}|}$, returns a symbol $s_i \in \mathcal{S}$. Intuitively, t is a vector of confidences returned by a neural network, which in our framework are interpreted as a vector of fuzzy truth values. Formally, t_i corresponds to the truth value of the proposition $(s_i = s^*)$, where s^* is the correct (unknown) symbol. Moreover, we define the function $\mu : [0,1]^{|\mathcal{S}|} \to [0,1]$ as the function that returns the truth value of the symbol chosen by the policy.

The *greedy policy* selects the symbol with highest truth value: $\pi(t) = argmax_i t_i$. The function μ returns the corresponding truth value: $\mu(t) = max_i t_i$. DSL exploits the differentiability of μ to indirectly influence the policy π, which is not differentiable. In the case of the greedy policy, by decreasing the highest confidence ($\mu(t)$), we reduce the chances for the current symbol to be selected again.

ϵ-greedy behaves like the greedy policy with probability $1-\epsilon$, while it chooses a random symbol with probability ϵ. The advantage of ϵ-greedy over greedy is a better ability to explore the solutions space. In our experiments, we use ϵ-greedy during training, and greedy policy at test time.

DSL for Direct NeSy-functions. For sake of presentation, we first assume symbolic functions to be given, and our goal is to learn the perception functions. We will then extend DSL to learn also the symbolic function.

We first define the representation of the perception functions $f_i : \mathcal{X} \to \mathcal{S}_i$ and the symbolic function g. W.l.o.g., we assume that symbols in any set $\mathcal{S}_i$ are represented by integers from 1 to $|\mathcal{S}_i|$. The symbolic function $g : \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \to \mathcal{S}$ is stored as a $|\mathcal{S}_1| \times \cdots \times |\mathcal{S}_n|$ tensor $\mathbf{G}$, where $\mathbf{G}[s_1, \ldots, s_n]$ contains the integer representing the symbolic output of $g(\bar{s})$. Every perception function $f_i : \mathcal{X} \to \mathcal{S}_i$ is modelled as $\pi(N_i)$, where $N_i : \mathcal{X} \to [0,1]^{|\mathcal{S}_i|}$ is a neural network (NN), and $\pi : [0,1]^{|\mathcal{S}_i|} \to \mathcal{S}_i$ is a *policy function*. For every $x \in \mathcal{X}$, $N_i(x)$ is an $|\mathcal{S}_i|$-dimensional vector $\bar{t}_i \in [0,1]^{|\mathcal{S}_i|}$ whose entries sum to 1. Intuitively, the l^{th} entry in $\bar{t}_i$ represents the predicted truth value associated with the l^{th} symbol being the output of $f_i(x)$. The policy function π makes a choice and picks a single symbol from $\mathcal{S}_i$ based on $\bar{t}_i$. In summary, our model is defined as:

$$\phi'(\bar{x}) = \mathbf{G}[\pi(N_1(x_1)), \ldots, \pi(N_n(x_n))]$$

where ϕ' is the learned approximation of target function ϕ.

Example 4 (Example 1 continued)**.** *We assume the same setup as Example 1, with an addition that $f_i(x_i)$ is $\pi(N_i(x_i))$ (with $i \in 1, 2$), as presented above. Let the prediction of f_1 and f_2 be the integers 3 and 5 respectively. In this context, $\mathbf{G}$ is a matrix that contains the sum of every possible pair of digits, so that $\mathbf{G}[i,j] = i + j$. Therefore, the prediction is: $\phi'(\bar{x}) = \mathbf{G}[3,5] = 8$.*

Learning the Perception Functions. In example 4, if one of the two internal predictions were wrong, then the final prediction 8 would be wrong as well. Hence, we define the confi-

dence of the final prediction to be the same as the confidence of having both internal symbols correct simultaneously. In other words, we could consider the output $\phi'(\bar{x})$ of the model to be correct if the following formula holds for all perception input x_i:

$$\big(\pi(N_1(x_1)) = s_1^*\big) \wedge \cdots \wedge \big(\pi(N_n(x_n)) = s_n^*\big) \qquad (2)$$

where s_i^* is the (unknown) ground truth symbol associated to perception x_i. We interpret formula in Equation 2 using Gödel semantics, where the conjunctions are interpreted by the min function. We use t_i^* to denote the truth value (or the confidence) given by the NN for the symbol selected by π, i.e., $t_i^* = \mu(N_i(x_i))$. Hence, the truth value t^* associated to the final prediction $\phi'(\bar{x})$ is given as:

$$t^* = min_i\ t_i^* = min_i\ \mu(N_i(x_i)) \qquad (3)$$

To train the model we use the binary cross entropy loss on the confidence t^* of the predicted symbol. If it is the right prediction, the confidence should be increased. In such a case, the ground truth label is set to one. If $\phi'(\bar{x})$ is the wrong prediction, the confidence should be reduced, and the label is set to zero. In summary, the entire architecture is trained with the following loss function:

$$\mathcal{L} = -\sum_{(\bar{x},y)\in Tr} l \cdot \log(t^*) + (1-l)\cdot\log(1-t^*)$$

where $l = \mathbb{1}(\phi'(\bar{x}) = y)$, and $\mathbb{1}$ is the indicator function. The architecture is summarized in Figure 1, where we show an instance of DSL in the context of Example 1.

DSL for Recurrent NeSy-functions. In DSL, a simple recurrent NeSy function is represented recursively as:

$$\phi'(X^{(k)}) = \mathbf{G}[\pi(N(x^{(k)})), \phi'(X^{(k-1)})]$$
$$\phi'(X^{(0)}) = \pi(\sigma(\mathbf{W}_0))$$

where $\mathbf{W}_0 \in \mathbb{R}^{|\mathcal{S}|}$ is the set of weights associated to the initial output symbol. Again, we define $t^* = min_i t_i^*$ as the minimum among the truth values of the internally selected symbols. The architecture is presented in Figure 2. It is worth noticing the similarity between the DSL model and the Equation 2. In general, a DSL model can be instantiated by following the same compositional structure of the NeSy function we want to learn, applying the policy when a value is expected to be symbolic. For instance, in the multi-digit task of Example 3, we can change the model by exploiting two distinct matrices (G_c and G_s) of size $[10, 10, 2]$, instead of one. G_c maps the two current images and the carry to two possible outputs (the next carry values), while G_s to 10 (the output digits). Differently from the visual parity case, here the output is a list of numbers, whose dimension is same as the inputs (e.g., [3,2]+[4,1]=[7,3]) or one digit longer (e.g., [9,2]+[4,1]=[1,3,3]). For this reason, we add a padding consisting of zeros at the beginning of the input lists, making their length the same as the output.

Learning Symbolic Functions. So far we have assumed the symbolic function g to be given. We now lift this assumption and define a strategy for learning the g. The idea

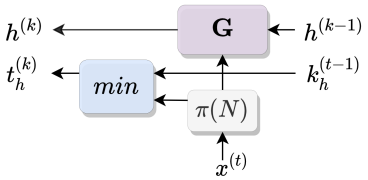

Figure 2: Architecture of Deep Symbolic Learning for the simple recurrent NeSy functions.

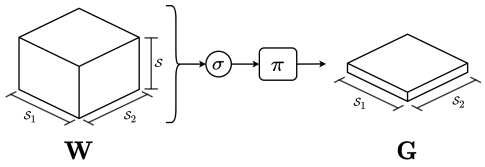

Figure 3: Tensor $\mathbf{W}$ is used by the policy to generate tensor $\mathbf{G}$. This is done by applying the policy on the output dimension (vertical axes in the image), selecting a single output element for each pair of symbols $(s_1, s_2) \in \mathcal{S}_1 \times \mathcal{S}_2$.

comes from a simple observation: for each tuple $\bar{s}$ there exists exactly one output symbol $g(\bar{s})$. Note that the mechanism introduced to select a unique symbol from the NN output can be also used for selecting *propositional symbols*, i.e. static symbols that do not depend on the current perceptions. We use the policy functions on learnable weights, allowing to learn the symbolic rules directly from the data.

Formally, we define a tensor $\mathbf{W} \in \mathbb{R}^{|\mathcal{S}_1|\times\cdots\times|\mathcal{S}_n|\times|\mathcal{S}|}$ as the weight tensor of $\mathbf{G}$. Note that the tensor shape is the same as $\mathbf{G}$, except for the additional final dimension, which is used to store the weights for all of the output symbols. The entry in $\mathbf{G}$ corresponding to tuple $\bar{s}$ is defined as:

$$\mathbf{G}[\bar{s}] = \pi(\sigma(\mathbf{W}[\bar{s},\ :]))$$

where the softmax function σ and the policy π are applied along the last dimension of $\mathbf{W}$. The method is summarized by Figure 3.

Since the tensor $\mathbf{G}$ is now learned, we need to consider the confidence associated with the choice of symbols in $\mathbf{G}$. The confidence of the final prediction is now defined as

$$t^* = min(t_G^*, min_i\ t_i^*) \qquad (4)$$

where t_G^* is the confidence of the output symbol for the current prediction:

$$t_G^* = \mu(\sigma(\mathbf{W}[\bar{s},\ :])$$

with $\bar{s} = \pi(\bar{N}(\bar{x}))$ corresponding to the tuple of predictions made by the perception functions.

Gradient Analysis for the Greedy Policy. We analyze the partial derivatives of the loss function with respect to the truth values t_i^* and t_G^*. However, notice that t^* in equation (3) and (4) are computed by selecting minimum over $\{t_i^*\}_{i=1}^{|S|}$ and $\{t_i^*\}_{i=1}^{|S|} \cup \{t_G^*\}$ respectively. Hence, for simplicity of notation, in this analysis we denote t_G^* by t_0^*. We consider only a single training sample, and assume that the policy is the

greedy one.

$$\frac{\partial \mathcal{L}}{\partial t_i^*} = -l\frac{\partial \log(t^*)}{\partial t_i^*} - (1-l)\frac{\partial \log(1-t^*)}{\partial t_i^*}$$
$$= -\frac{l}{t^*}\frac{\partial t^*}{\partial t_i^*} + \frac{1-l}{1-t_i^*}\frac{\partial t^*}{\partial t_i^*}$$

Now, since t^* is the minimum of all $\{t_i^*\}_{i=0}^{|S|}$, the term $\frac{\partial t^*}{\partial t_i^*}$ is 1 if t_i is the minimum value in $\{t_i^*\}_{i=0}^{|S|}$ and 0 otherwise, reducing the total gradient to the following equation :

$$\frac{\partial \mathcal{L}}{\partial t_i^*} = \begin{cases} -\frac{l}{t^*} + \frac{1-l}{1-t^*} & i = \text{argmin}_j\, t_j^* \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

For each sample, only one confidence value t_i^* has a non-zero gradient, meaning that only a single symbolic choice is supervised, i.e., the choice of a rule symbol (when $i = 0$ in equation (5)) or the choice of a perception symbol. Hence, depending on the performance on a given sample, DSL manages to modify either a perception or the symbolic function. This behaviour is shown in Figure 1 by using red arrows to represent the backward signal generated by the backpropagation algorithm. The signal moves from the loss to f_1, which corresponds to the symbol with lower confidence, and when it reaches the softmax function (σ), it is spread to the entire network. In DSL, we have not only interpretable predictions, but gradients are interpretable as well. Indeed, for each sample, there is a unique function f_i or g taking all the blame (or glory) for a bad (or good) prediction of the entire model ϕ'.

5 Experiments

We evaluate our approach on a set of tasks where a combination of perception and reasoning is essential. Our goal is to demonstrate that: i) DSL can learn the NeSy function while simultaneously learning the two components f and g, in an end-to-end fashion (MNIST sum); ii) The perception functions f_i learned on a given task are easily transferable to new problems, where the symbolic function g has to be learned from scratch, with only a few examples (MNIST Minus - One-Shot Transfer); iii) DSL can also be generalized to problems with a recurrent nature (MNIST visual parity), iv) when we provide a smaller representation for G DSL can solve harder tasks, like the multi-digits sum. Furthermore, it can easily generalize up to N-digits sum, with a very large N (MNIST Multi-Digit sum).

Evaluation. The standard metric used for this type of tasks is the accuracy applied directly to the predictions of the NeSy function ϕ. This allows to understand the general behaviour of the entire model. However, different from previous models, the symbolic function is also learned from the data. For this reason, we also considered the quality of the learned symbolic rules. Nevertheless, the symbols associated with perception inputs in DSL are internally generated and form a permutation-invariant representation. Any permutation of the symbols leads to the same behavior of the model, given that the same permutation is applied to the indices of the tensor $\mathbf{G}$. Hence, to evaluate the model on learning of gs, we need to select a permutation that best explains the model w.r.t the "human" interpretation of symbols for digits. The problem is highlighted in Figure 4(left), where the confusion matrix of the MNIST digit classifier is introduced. Note that for each row (digit), only one column (predicted symbol) has a high value. The same is true for the columns. The network can distinguish the various digits, but the internal symbols are randomly assigned. To obviate this problem, we calculate the permutation of columns of the confusion matrix which produces the highest diagonal values (Figure 4(right)). We then apply the same permutation on the confusion matrix and $\mathbf{G}$, allowing us to obtain a human interpretable set of rules. This procedure allows for measuring the performances of DSL on learning the rules. However, we will omit this metric in the next sections since in all of our experiments tensor $\mathbf{G}$ is perfectly learned, i.e., we have an accuracy of 100% on the learned rules.

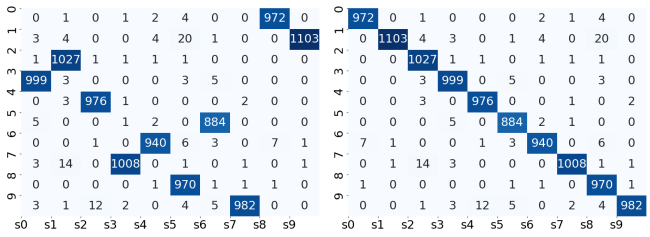

Figure 4: Confusion matrix for the MNIST digits: (left) before the permutation; (right) after permutation.

Implementation Details. All the experiments were conducted with a machine equipped with an NVIDIA GTX 3070 with 12GB RAM. For digit classification, we use the same CNN as [Manhaeve *et al.*, 2018]. We used MadGrad [Defazio and Jelassi, 2022] for optimization and optuna to select the best hyperparameters for every experiment. Results are averaged over 10 runs.

MNIST Sum. We first tackled the MNIST sum task presented in Example 1. A dataset consisting of triples (X, Y, Z) is given, where X and Y are two images of hand-written digits, while Z is the result of the sum of the two digits, e.g., ( ,  , 8). The goal is to learn an image classifier for the digits and the function g which maps digits to their sum. We implemented two different variants of our approach: **DSL** is the naive version of DSL, where the two digits are mapped to symbols by the same perception function, and the correct number of digits is given a priori; **DSL-NB** is a version of **DSL** where we removed the two aforementioned biases: we use two different neural networks, N_1 and N_2, to map perceptions to symbols, and the model is unaware of the right amount of latent symbols, with the neural network returning confidence on 20 symbols instead of 10. In table 1, we show that DSL variants have competitive performance w.r.t the state of the art [Yang *et al.*, 2020], [Winters *et al.*, 2022], [Manhaeve *et al.*, 2018]. Notice that all the SOTA methods receive a complete knowledge of the symbolic function g, while DSL needs to learn it, making the task much harder. Another important result is the accuracy of the DSL-NB method, which proves that DSL can work even with two perception networks and, most importantly, without knowing

	Accuracy (%)	TE/#E
NAP	$97.3_{\pm 0.3}$	$109s/1$
DPL	$97.2_{\pm 0.5}$	$367s/1$
DStL	$97.9_{\pm 0.1}$	$25.49s/2$
DSL	$98.8_{\pm 0.3}$	$0.95s/50$
DSL-NB	$97.9_{\pm 0.3}$	$0.99s/200$

Table 1: Results obtained on the MNIST sum task. TE is the time required for 1 epoch, and #E is the number of epochs of training. The SOTA methods are NeurASP (NAP), DeepProbLog (DPL), and DeepStochLog (DStL).

	Accuracy (%)			
	2	4	15	1000
NAP	$93.9_{\pm 0.7}$	T/O	T/O	T/O
DPL	$95.2_{\pm 1.7}$	T/O	T/O	T/O
DStL	$96.4_{\pm 0.1}$	$92.7_{\pm 0.6}$	T/O	T/O
DSL	$95.0_{\pm 0.7}$	$88.9_{\pm 0.5}$	$64.1_{\pm 1.5}$	$0.0_{\pm 0.0}$
	Fine-grained Accuracy (%)			
	2	4	15	1000
DSL	$97.9_{\pm 0.1}$	$97.3_{\pm 0.1}$	$96.7_{\pm 0.1}$	$96.5_{\pm 0.1}$

Table 2: Results obtained on the MNIST Multi-digit sum task. T/O stands for timeout.

the right amount of internal symbols.

MNIST Minus - One-Shot Transfer. One of the main advantages of NeSy frameworks is that the perception functions learned in the presence of a given knowledge (g in our framework) can be applied to different tasks without retraining, just by changing the knowledge. For instance, after learning to recognize digits from supervision on the addition task, methods like DeepProblog can be used to predict the difference between two numbers. However, it is required for a human to create different knowledge bases for the two tasks. In our framework, the g function is learnable, and the mapping from perception to symbols does not follow human intuition (see Evaluation Metrics section). Instead of writing a new knowledge for the Minus task, we replace the tensor $\mathbf{G}$ with a new one and learn it from scratch. In our experiment, we started from the perception function learned from the Sum task and used a single sample for each pair of digits to learn the new $\mathbf{G}$. We obtained an accuracy of 98.1 ± 0.5 after 300 epochs, each requiring $0.004s$. Note that we did not need to freeze the weights of the f. Since the perception functions already produce outputs with high confidence, DSL applies changes mainly on the tensor $\mathbf{G}$.

MNIST Visual Parity. We used the model in Figure 2 for the parity task using images of zeros and ones from the MNIST dataset, and the same CNN used for the MNIST sum task (with 2 output symbols instead of 10). Learning the parity function from sequences of bits is a hard problem for neural networks, which struggle to generalize to long sequences [Shalev-Shwartz *et al.*, 2017]. The parity function corresponds to the symbolic function g, and learning the perception function is an additional sub-task. We used sequences of 4 images during training and 20 on the test and only provided supervision on the final output. DSL reached an accuracy of 98.7 ± 0.4 in 1000 epochs, showing great generalization capabilities. As in other tasks, DSL learned the XOR function perfectly. Tnhe errors made by the model only depend on the perception functions. If the perceptions are correctly recognized, the model works regardless of the sequence length.

MNIST Multi-digit Sum. The previous experiment on the Visual Parity have demonstrated the ability of DSL to learn recursive NeSy functions. However, this experiment was conducted on a simple task where the number of allowed symbols was limited to two, and the symbolic function g could be directly stored in a 2x2 matrix.The multi-digit sum is more challenging since the hypothesis space becomes much larger, and we need to learn two symbolic functions (g_c and g_s) simultaneously. Thus, we decided to rely on Curriculum Learning [Bengio *et al.*, 2009], where initially we provide only samples composed of a single digit and no padding, reducing the problem to learning the digit sum modulo 10. We then provide another training set composed of two digits numbers and the padding, allowing the model to learn the missing rules. We trained our model on the 2-digits sum and we evaluate the learned model on sequences of varying length, showing the generalization capabilities of DSL.

Table 2 reports the results obtained by NeurASP, DeepProblog, DeepStochLog and DSL. We tested our model on N-digit sums, with N up to 1000. Also in this case, DSL learned perfect rules; thus, the accuracy degradation obtained by increasing the value of N is only due to errors made by the perception function (98.5% accuracy). For this reason, our performance follows a similar trend of $0.985^{(2N)}$. To better understand the true performance of DSL, we also measured a fine-grained accuracy that measures the mean ratio of correct digits in the final output. Furthermore, our approach took only 0.27 seconds to infer on the entire test set for $N = 1000$, while no other methods scale to more than 4 digits.

6 Conclusion and Future Work

We presented Deep Symbolic Learning, a NeSy framework for learning the composition of perception and symbolic functions. To the best of our knowledge, DSL is the first NeSy system that can create and map symbolic representations to perception while learning the symbolic rules simultaneously. A key technical contribution of DSL is the integration of discrete symbolic choices within an end-to-end differentiable neural architecture. For this, DSL exploits the notion of policy deriving from Reinforcement Learning. Furthermore, DSL can learn the perception and symbolic functions while performing comparably to SOTA NeSy systems, where complete supervision of the symbolic component is given. Moreover, in the multi-digit sum, DSL's inference scales linearly, allowing the evaluation of huge sequences. In the future, we aim to extend DSL to problems with a larger combinatorial search space. To this end, we aim to consider factorized matrix representations for the symbolic function g, and its weight matrix W. Furthermore, we aim to generalize DSL to more complex perception inputs involving text, audio, and vision.

Acknowledgments

TC and LS acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

References

[Aspis *et al.*, 2022] Yaniv Aspis, Krysia Broda, Jorge Lobo, and Alessandra Russo. Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2022.

[Badreddine *et al.*, 2022] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *Artificial Intelligence*, 2022.

[Bengio *et al.*, 2009] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

[Besold *et al.*, 2021] Tarek R. Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. In *Neuro-Symbolic Artificial Intelligence: The State of the Art*, 2021.

[Bruynooghe *et al.*, 2010] Maurice Bruynooghe, Theofrastos Mantadelis, Angelika Kimmig, Bernd Gutmann, Joost Vennekens, Gerda Janssens, and Luc De Raedt. Problog technology for inference in a probabilistic first order logic. In *European Conference on Artificial Intelligence (ECAI)*, 2010.

[Chang *et al.*, 2020] Oscar Chang, Lampros Flokas, Hod Lipson, and Michael Spranger. Assessing satnet's ability to solve the symbol grounding problem. In *Adv. Neural Inform. Process. Syst. (NIPS)*, 2020.

[Chen *et al.*, 2016] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Adv. Neural Inform. Process. Syst. (NIPS)*, 2016.

[Dai and Muggleton, 2021] Wang-Zhou Dai and Stephen Muggleton. Abductive knowledge induction from raw data. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[Dai *et al.*, 2019] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In *Adv. Neural Inform. Process. Syst. (NIPS)*, 2019.

[Darwiche, 2011] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[Defazio and Jelassi, 2022] Aaron Defazio and Samy Jelassi. Adaptivity without compromise: a momentumized, adaptive, dual averaged gradient method for stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 2022.

[Diligenti *et al.*, 2017] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 2017.

[Evans *et al.*, 2021] Richard Evans, Matko Bošnjak, Lars Buesing, Kevin Ellis, David Pfau, Pushmeet Kohli, and Marek Sergot. Making sense of raw input. *Artificial Intelligence*, 2021.

[Harnad, 1990] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 1990.

[Huang *et al.*, 2021] Yu-Xuan Huang, Wang-Zhou Dai, Le-Wen Cai, Stephen H Muggleton, and Yuan Jiang. Fast abductive learning by similarity-based consistency optimization. In *Adv. Neural Inform. Process. Syst. (NIPS)*, 2021.

[LeCun *et al.*, 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[Manhaeve *et al.*, 2018] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Adv. Neural Inform. Process. Syst. (NIPS)*, 2018.

[Riegel *et al.*, 2020] Ryan Riegel, Alexander Gray, Francois Luus, Naweed Khan, Ndivhuwo Makondo, Ismail Yunus Akhalwaya, Haifeng Qian, Ronald Fagin, Francisco Barahona, Udit Sharma, et al. Logical neural networks. *CoRR*, abs/2006.13155, 2020.

[Rocktäschel and Riedel, 2017] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Adv. Neural Inform. Process. Syst. (NIPS)*, 2017.

[Shalev-Shwartz *et al.*, 2017] Shai Shalev-Shwartz, Ohad Shamir, and Shaked Shammah. Failures of gradient-based deep learning. In *International Conference on Machine Learning (ICML)*, 2017.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. Reinforcement learning - an introduction. *MIT Press*, 1998.

[Topan *et al.*, 2021] Sever Topan, David Rolnick, and Xujie Si. Techniques for symbol grounding with satnet. In *Adv. Neural Inform. Process. Syst. (NIPS)*, 2021.

[Tsamoura *et al.*, 2021] Efthymia Tsamoura, Timothy Hospedales, and Loizos Michael. Neural-symbolic integration: A compositional perspective. In *AAAI*, 2021.

[Wang *et al.*, 2019] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning (ICML)*, 2019.

[Winters *et al.*, 2022] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. In *AAAI*, 2022.

[Xu *et al.*, 2018] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International Conference on Machine Learning (ICML)*, 2018.

[Yang *et al.*, 2020] Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2020.