Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning
Special Session on KR and Machine Learning

Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings

Yaniv Aspis[1] , **Krysia Broda**[1] , **Jorge Lobo**[2,3] , **Alessandra Russo**[1]
[1]Imperial College London
[2]ICREA
[3]Universitat Pompeu Fabra
{yaniv.aspis17, k.broda, a.russo}@imperial.ac.uk, jorge.lobo@upf.edu

Abstract

Neuro-symbolic reasoning approaches proposed in recent years combine a neural perception component with a symbolic reasoning component to solve a downstream task. By doing so, these approaches can provide neural networks with symbolic reasoning capabilities, improve their interpretability and enable generalization beyond the training task. However, this often comes at the cost of poor training time, with potential scalability issues. In this paper, we propose a scalable neuro-symbolic approach, called Embed2Sym. We complement a two-stage (perception and reasoning) neural network architecture designed to solve a downstream task end-to-end with a symbolic optimisation method for extracting learned latent concepts. Specifically, the trained perception network generates clusters in embedding space that are identified and labelled using symbolic knowledge and a symbolic solver. With the latent concepts identified, a neuro-symbolic model is constructed by combining the perception network with the symbolic knowledge of the downstream task, resulting in a model that is interpretable and transferable. Our evaluation shows that Embed2Sym outperforms state-of-the-art neuro-symbolic systems on benchmark tasks in terms of training time by several orders of magnitude while providing similar if not better accuracy.

1 Introduction

Research into methods of combining neural and symbolic approaches to AI has accelerated in recent years (Sarker et al. 2021). Neural networks have achieved high success over a large range of tasks involving perception, such as object recognition (Zhao et al. 2019) or acoustic event detection (Xia et al. 2019). However, tasks involving reasoning are generally considered challenging for a neural network. While an architecture can be devised to solve specific tasks (Palm, Paquet, and Winther 2018; Lample and Charton 2020), a trained network lacks the true generalization capabilities that a reasoning system should possess, while also not being interpretable. Neuro-symbolic methods are seen as a method of enhancing a neural network by providing it with some of the reasoning capabilities and interpretability that a symbolic system traditionally has.

Broadly speaking, many systems are built around differentiable neural architectures that learn to approximate symbolic computation from data (Minervini et al. 2018; Riegel et al. 2020). A different approach that has recently gained attention involves extending the capabilities of symbolic solvers by (1) allowing the system to depend on the output of a neural component, and (2) using inconsistent results arising from the neural output, reasoning process and data to guide the training of the network. These systems see neuro-symbolic reasoning on raw data as a combination of two tasks: a perception task, handled by a neural component, and a reasoning downstream task, handled by the symbolic solver. In this paper we refer to these systems as "neuro-symbolic reasoning systems". Prominent examples include DeepProbLog (Manhaeve et al. 2021) and NeurASP (Yang, Ishay, and Lee 2020).

In neuro-symbolic reasoning, the output of the neural component is associated with a symbolic concept (typically a predicate of a logic program). We refer to these concepts as "latent" as one does not typically provide labels for them. In fact, the ability of neuro-symbolic reasoning systems to train a neural network by only using the labels of the downstream task ("target labels") is one of their great advantages. Additional advantages include increased interpretability, and the ability to generalize to downstream tasks other than the one the system was trained on (by keeping the perception component fixed and applying different rules). However, experiments have shown that the training time for these systems can be significantly long even when the perception task is simple (Tsamoura, Hospedales, and Michael 2021; Yang, Ishay, and Lee 2020). The difficulty in training is inherent in current approaches: in order to train, the symbolic component must "work backwards" from a given target label to a set of possible latent labels, which can be exponential in size. This fact has so far hindered most neuro-symbolic reasoning systems from being applied to tasks involving a larger number of inputs and more complex perception tasks are usually not considered, with a few exceptions, e.g. (Huang et al. 2021).

In contrast, approaches such as Concept Bottleneck models (Koh et al. 2020) have demonstrated that performing the reasoning step with a second neural network can achieve competitive accuracy when latent concepts are effectively integrated into the neural architecture, without loss of scalability. But they require latent bottleneck concepts to be labelled to preserve interpretability and generalization. They are also not typically applied in a neuro-symbolic setting where multiple raw inputs must be considered together.

In this paper, we introduce Embed2Sym, a novel approach to neuro-symbolic reasoning that is scalable, able to produce neuro-symbolic models that are interpretable and generalize beyond the training task, and does not require latent concepts to be labelled. Embed2Sym works in three steps. First, a neural network composed of a perception component and a reasoning component is trained on the downstream task end-to-end. The architecture for this network has previously been shown to be effective in solving diagrammatic reasoning tasks (Wang, Jamnik, and Liò 2018). Here we consider its use within the context of a general neuro-symbolic approach. By training the perception component in this manner, rather than through evaluation of a logical formula, we avoid the combinatorial difficulty that other neuro-symbolic reasoning systems confront. Second, we search for latent concepts that have been learned by the perception component by analysing the clustered structure of datapoints in embedding space. Third, we identify the discovered latent concepts and assign them their symbolic meaning by means of an optimisation task solved using a symbolic system (Clingo) and background knowledge of the task. In this way, we are able to lift a purely neural model into a neuro-symbolic one that is interpretable and can generalize to tasks the neural network alone could not perform.

We evaluate Embed2Sym on several tasks previously introduced to benchmark neuro-symbolic systems. Results confirm Embed2Sym achieves similar or better accuracy to competing systems while training faster by orders of magnitude. The scalability of Embed2Sym allows it to train on tasks involving at least dozens of raw inputs, such as adding two 15-digit numbers given as MNIST images, which other systems find infeasible. We also confirm Embed2Sym can scale to tasks involving more challenging perception domains such as the CIFAR-10 dataset.

The paper is structured as follows: Section 2 discusses related work. Section 3 covers the various aspects of the Embed2Sym algorithm for neuro-symbolic training and inference. Section 4 presents the results of an evaluation of the system confirming its performance and scalability. Section 5 discusses the results and Section 6 concludes the paper.

2 Related Work

Research on combining neural networks and symbolic reasoning has a long history, and approaches vary greatly, see survey in (Besold et al. 2017). For example, Neural Theorem Prover (Minervini et al. 2018) uses soft unification to learn symbol embeddings to correctly satisfy queries. Logic Tensor Networks (Badreddine et al. 2022) represent First-Order Logic constructs as vectors and differentiable functions, and uses fuzzy logical operations. Logical Neural Networks (Riegel et al. 2020) implement conjunction and disjunction using a perceptron whose weights and bias are learned with a tailored activation function designed to preserve two-valued semantics as much as possible.

We consider our work to be most closely related to neuro-symbolic reasoning systems that extend a symbolic solver with neural capabilities. DeepProbLog (Manhaeve et al. 2021) is an extension of the ProbLog system (De Raedt, Kimmig, and Toivonen 2007). A new construct in the form of a neural Annotated Disjunction allows for the output of a network to be interpreted as the probabilities of atoms. A formula computing the probability of a query is compiled and transformed into an arithmetic circuit which can be evaluated and used to produce gradient information for the network. DeepStochLog (Winters et al. 2021) is an analogous framework that uses Stochastic Definite Clause Grammars rather than probabilistic logic programs.

NeurASP (Yang, Ishay, and Lee 2020) extends Answer Set Programming with neural predicates. The predicate is linked to a neural network and translated by NeurASP into a choice rule representing the various output of the network. The answer sets of the program are assigned probabilities based on the network predictions, and the network is then trained to optimise a semantic loss function (Xu et al. 2018).

Abductive Learning (Dai et al. 2019) considers the output of a neural network in the absence of latent labels as incomplete knowledge and therefore uses abductive reasoning to generate these labels. Neurolog (Tsamoura, Hospedales, and Michael 2021) generalizes this concept by producing all abductive solutions for a given goal, and then applying a semantic loss. More recently, abductive learning has been extended into ABL-sim that uses vector embedding of latent concepts to guide the training of the neural network (Huang et al. 2021), which is trained using an additional loss term.

Similarly to the neuro-symbolic reasoning systems mentioned above, we divide a task into perception and reasoning sub-tasks. We also enhance a symbolic paradigm, in our case Answer Set Programming, by providing it with a means to reason over unstructured data. Unlike these approaches, we do not use the symbolic component to guide the training of the network. Instead, we use it to extract and label the latent concepts discovered by the network through end-to-end training. Like ABL-Sim, we consider the representation of latent concepts in embedding space. However, in our approach, these embeddings are not used as tools for guiding the training of the perception component, and hence we do not require any additional loss terms. Instead, these embeddings arise naturally when training the neural network, and are used to extract symbolic concepts out of the trained network for downstream symbolic computations.

3 Method

We wish to learn functions of the form $f = h \circ g : \mathcal{X} \to \mathcal{Z} \to \mathcal{Y}$, where f can be decomposed into a perception function $g : \mathcal{X} \to \mathcal{Z}$ and a reasoning function $h : \mathcal{Z} \to \mathcal{Y}$. $\mathcal{X}$ is the space of inputs, $\mathcal{Y}$ the space of target labels and $\mathcal{Z}$ a space of latent concept values relevant for the task.

A latent concept is a tuple $C = (n_C, S_C)$ where n_C is the name of the latent concept and S_C is an associated set of values, which we assume to be finite. The task input X to the function f itself is composed of multiple inputs $X = x_1 \times x_2 \times ... \times x_m$ where each x_i is either a raw input (e.g. image, text) or a symbolic input. While there could be many latent concepts, we assume w.l.o.g. that each x_i is associated with a single one. To deal with multiple latent concepts per input, we can construct a composite latent concept. For example, if x_i should be associated with

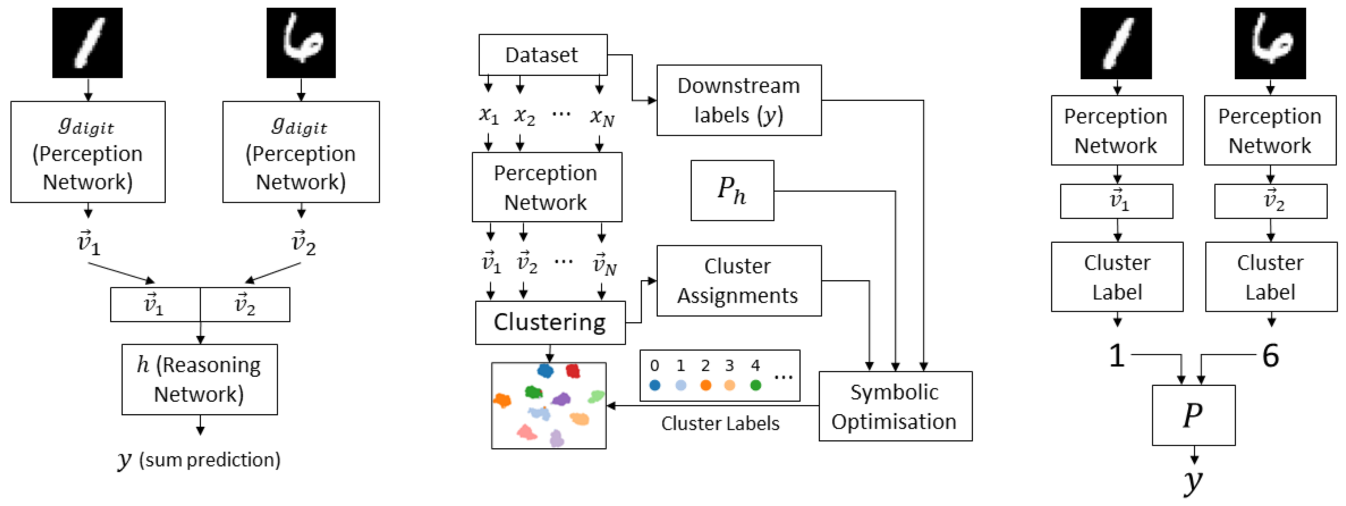

Figure 1: Embed2Sym architecture. Left: An example of a fully neural architecture for the MNIST Addition task. Middle: After training the fully neural network, clustering is applied on embedding vectors computed by the perception network. Symbolic optimisation is used to label the clusters together with the rules P_h. Right: An example inference on MNIST digits. The cluster labels are used as symbolic inputs to the program P (which may differ from P_h).

both $C_1 = (n_{C_1}, S_{C_1})$ and $C_2 = (n_{C_2}, S_{C_2})$ then we construct $C_{1,2} = \left(n_{C_{1,2}}, S_{C_1} \times S_{C_2}\right)$. $\mathcal{Y}$ is a set of target labels that represent the output of the full neuro-symbolic reasoning task. $\mathcal{C}$ is the set of latent concepts of the task.

To train our model, we are given a dataset of samples $\mathcal{D} = \{X, y\} \subset \mathcal{X} \times \mathcal{Y}$. Crucially $\mathcal{D}$ does not contain any information about the correct latent labels in $\mathcal{Z}$. To perform well at the task, the model would have to learn how to correctly map a raw input x_i to its latent concept value, which is the responsibility of the g function. g in general is in fact several functions, one for each latent concept. We denote the perception function associated with latent concept C as g_C. Each g_C is effectively a classifier for the latent concept C trained via a weakly supervised learning task using $\mathcal{D}$.

Example 1. *Consider the MNIST Addition task (Manhaeve et al. 2021). In this task, we are given two images of MNIST digits (LeCun, Cortes, and Burges 2010) and must output their sum. The task input X is a composition of the two 28x28 pixel images x_1 and x_2. Each image is associated with the latent concept $(\text{“digit”}, \{0, 1, ..., 9\})$. The target labels are the 19 possible sums $\mathcal{Y} = \{0, 1, ..., 18\}$. Samples in $\mathcal{D}$ are of the form (◼, ◼, 7). g is just a single function g_{digit} that maps an MNIST image to its corresponding digit. g_{digit} must be trained when the only given label is the sum.*

As is common in neuro-symbolic reasoning settings, we assume the reasoning component h can be represented by a given set of rules P_h that is known in advance and can be considered background knowledge for the task. It may have been provided by a human, online knowledge bases such as WordNet (Fellbaum 1998) and ConceptNet (Speer, Chin, and Havasi 2017), or an inductive rule learning system (Law, Russo, and Broda 2020). In settings where such knowledge is not known in advance, the rules would have to be learned in conjunction with the perception module g. We leave such cases for future work and for now assume we have been given this knowledge.

3.1 Fully Neural Model

Since h can be implemented correctly using the known rules P_h, training the perception component g remains the primary goal. However, since the rules are not differentiable, this can present some challenges. Some authors have taken the approach of replacing hard logical rules with probabilistic or continuous approximations (Manhaeve et al. 2021; Xu et al. 2018). We choose a different path. We begin by training a fully neural model. In this model, h is modeled as a neural network that is trained from the data as an approximate reasoner. Although this may look at first to be counter-intuitive, as we are interested in a neuro-symbolic model, this neural model is useful in three ways. First, it allows us to train the perception function g in a setting where neural networks excel: continuous space with dense representations. Second, we will see that even though g is not trained using domain knowledge, it still discovers structure in the data that is relevant to the domain and can be used to lift our differentiable model into a neuro-symbolic one. Third, it allows us to scale very well with the number of raw inputs the model receives.

For each latent concept C, we have a function $g_C : \mathcal{X}_C \to \mathbb{R}^k$ as a neural network. $\mathcal{X}_C$ is the space of raw inputs that can be mapped to values of C. In the MNIST Addition task, this will be 28x28 grayscale images. g_C maps these inputs to a vector space of *embeddings* of size k. k is a hyperparameter that can be set to different values for each C. The embeddings should form a dense representation of the latent concept values and a good g_C function would map raw inputs having the same concept values close together. The function h is also a neural network that operates on the set of embedding vectors produced from each input, and outputs the predicted label $y \in \mathcal{Y}$.

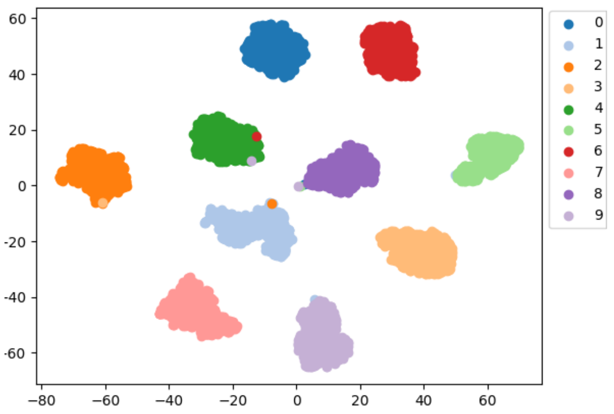

Figure 2: t-SNE visualization of g_{digit} embedding space post-training, clearly showing clusters representing 10 different digits.

More formally, for each input x_i in X let $C(i)$ be its associated latent concept. The fully neural model is composed as follows:

$$\forall i = 1...m \quad \vec{v}_i = g_{C(i)}\left(x_i\right) \tag{1}$$

$$y = h\left(\vec{v}_1, \vec{v}_2, ..., \vec{v}_m\right) \tag{2}$$

Figure 1 (left) illustrates this architecture for the MNIST Addition example when $m = 2$. g_{digit} in this case could be a CNN such as a LeNet, but modified to output an embedding vector ($\vec{v}_1/\vec{v}_2$) instead of a probability simplex vector, as commonly done by using a softmax operation. This change is important, as softmax outputs of the perception module do not make much sense in a purely neural setting. In addition, its use of exponents tends to cause one argument to dominate the others, sending a sparser signal to the reasoning component. For this task, h can simply be an MLP with an output layer of size 19, corresponding to the 19 possible sums, activated using softmax. The outputs of g_{digit} are concatenated before being fed into the h component.

3.2 Identifying Latent Concepts

The fully neural model described above can be trained to solve the task using $\mathcal{D}$. As the various g_C functions are trained, they tend to cluster datapoints corresponding to the same latent value together. So if $C = (n_C, \{v_1, v_2, v_3\})$, g_C will converge to a function that maps datapoints to three distinct clusters in embedding space. An example of this is shown in Figure 2. The plot depicts a t-SNE visualization of the embedding space of MNIST images after training the fully neural model depicted in Figure 1 (left) on the MNIST Addition task. Even though this model was not given any knowledge about relevant latent concepts for the task, 10 clear clusters emerged, each associated with a specific digit.

This observation is key to lifting the fully neural model into a neuro-symbolic one. To achieve neuro-symbolic reasoning, we need to transform each g_C function into a classifier for C. We can exploit the clustering behaviour to do so. After training g_C, we identify the various clusters in embedding space. Then, we label each cluster with the best latent concept value. The process for cluster discovery and labelling is depicted in Figure 1 (middle). When we wish to predict the latent concept value associated with a raw input x_i, we compute $g_C(x_i)$, check which cluster it belongs to and predict the label of that cluster.

```
latent_concept(n_c, s_i, i).

cluster_assignment(sample_id, n_c, x_i, j).

{cluster_mapping(n_c, X, Y) : Y=0..|S_C|} = 1 :-
    X=0..|S_C|.
:- cluster_mapping(n_c, X, Z),
    cluster_mapping(n_c, Y, Z), X != Y.

holds(Id, C, RawInput, Value) :-
    cluster_assignment(Id, C, RawInput, ClusterIdx),
    cluster_mapping(C, ClusterIdx, ValueIdx).
    latent_concept(C, Value, ValueIdx).

% Task-specific rules defining the label
% in terms of the ''holds'' predicate

:~ holds(sample_id, target, y). [-1, 0]
```

Listing 1: ASP encoding for the cluster labelling task.

Cluster Discovery To perform the clustering in embedding space, we employ the K-Means Clustering algorithm. K-Means has the advantages of being simple and fast, although it assumes clusters are isotropic in Euclidean distance. However, we believe this is sufficient to demonstrate our approach and have observed good results over the range of experiments we performed. Future work may be directed at investigating other clustering algorithms, such as DBScan (Ester et al. 1996) or Deep Clustering methods (Caron et al. 2018; Moradi Fard, Thonet, and Gaussier 2020).

Cluster Labelling So far we have identified the clusters in g_C's embedding space, but we have yet to discover which symbolic latent value of C is associated with each cluster. In other words, the clusters are unlabelled. For the labelling, we take advantage of the fact that the reasoning component h can be given an alternative representation using P_h.

An assignment of clusters to labels of C is simply a one-to-one mapping. We can evaluate such an assignment using the dataset $\mathcal{D}$. For each $(X, y) \in \mathcal{D}$, apply g to X to receive a sequence of embeddings. Each embedding is mapped to a cluster by K-Means, and then a latent concept value by the chosen assignment. We can then apply h in its symbolic rule form (P_h) to compute a prediction y'. The amount of datapoints misclassified is a measure of the assignment score. The smaller this value is, the better the assignment.

We can therefore define the task of labelling clusters as an optimisation problem, which we encode using an Answer Set Program. The syntax of Answer Set Programs supports an extensive range of programs allowing the expression of a large variety of neuro-symbolic problems. To solve the task, we use Clingo 5.5.1 (Gebser et al. 2017).

A summary of the ASP encoding of the optimisation task appears in Listing 1. For each latent concept $C = (n_C, \{s_0, s_2, ..., s_l\})$ we assign its various latent values indices. These are encoded as facts as can be seen in line 1. Next, we encode the various assignments of samples to clusters as provided by K-Means. For a sample with ID "sample_id", raw input x_i and its associated latent concept C, and the assigned cluster index j, we add a fact as in line 3.

For each latent concept C, we add a choice rule requiring each cluster to be mapped to a latent concept value (lines 5-6). We add an additional restriction that two clusters cannot be mapped to the same value (lines 7-9). For example, for MNIST Addition, we would have the following lines:

```
{cluster_mapping(digit, X, Y) : Y=0..9} = 1 :– X=0..9.
:– cluster_mapping(digit, X, Z),
      cluster_mapping(digit, Y, Z), X != Y.
```

These rules require that each cluster is mapped to exactly one of the digits 0-9 in an answer set. The next rule, seen in lines 10-13, defines the "holds" predicate that states that a raw input of a sample has been assigned a particular latent concept value in the answer set. A raw input x_i associated with latent concept C has been assigned a latent concept value $Value$ if the following three conditions hold. First, it has been assigned by K-Means to a cluster with index $ClusterIdx$ (indicated by the "cluster_assignment" predicate). Second, $ClusterIdx$ has been mapped to a latent concept value with index $ValueIdx$ (indicated by the "cluster_mapping" predicate). Third, the latent concept value with index $ValueIdx$ has value $Value$ (indicated by the "latent_concept" predicate).

With these rules in place, we can add the background knowledge related to the task (line 15). These are the rules that are given by P_h. The rules should be specified in terms of the "holds" predicate to establish a connection to the cluster assignment for each sample. For example, adding two MNIST digits together, we can add the following rule:

```
holds(Id, sum, Z) :– holds(Id, digit, $x_1$, X),
          holds(Id, digit, $x_2$, Y), Z = X+Y.
```

The "holds" atoms in the body will become true if their respective raw inputs have been mapped to a cluster that has been labelled by the variable X (for the first raw input x_1) or Y (for the second raw input x_2). The task's target output, in this case the sum, will also be indicated by a "holds" predicate, defined by the rule. The sample "Id" will have a target output "sum" whose value Z equals $X + Y$, as required.

Once the rules of h have been added, the optimisation process can produce predictions for every sample, once a cluster mapping has been chosen. To compute a score for each mapping, for each sample (X, y) with ID "sample_id", we add the weak constraint appearing in line 18, indicating its prediction for output "target" should be y. Each constraint has a penalty of -1, indicating that its body should be true. An optimal answer set has a minimal score, indicating the best coverage over the dataset. Using weak constraints in this way also allows us to deal with the fact that our data is noisy - the cluster assignments are not perfect. Once the task is solved, we extract the optimal cluster mapping out of the answer set, thus successfully labelling the clusters.

Algorithm 1 Embed2Sym Training

```
Input: $\mathcal{D}$, $\mathcal{C}$, $P_h$
Output: $g$, $K$, $\pi$
 1: Train fully neural model $f = h \circ g$ on $\mathcal{D}$
 2: Select $\mathcal{D}_2 \subseteq \mathcal{D}$
 3: $K \leftarrow \emptyset$
 4: for $c \in \mathcal{C}$ do
 5:     $V \leftarrow \emptyset$
 6:     for $(X, y) \in \mathcal{D}_2$ do
 7:         for $x_i \in X$ s.t. $c$ latent concept of $x_i$ do
 8:             Add $g_c(x_i)$ to $V$
 9:         end for
10:     end for
11:     Train $K_C$ on $V$ using K-Means Clustering.
12:     Add $K_C$ to $K$
13: end for
14: Encode task $O$ as in listing 1 using $\mathcal{D}_2$, $\mathcal{C}$, $K$, $P_H$
15: Solve $O$ and select optimal answer set $A$
16: $\pi \leftarrow \emptyset$
17: for $c \in \mathcal{C}$ do
18:     $\pi_c \leftarrow \emptyset$
19:     for $cluster\_mapping(n_C, i, j) \in A$ do
20:         Assign $\pi_c(i) = j$
21:     end for
22:     Add $\pi_c$ to $\pi$
23: end for
24: return $g$, $K$, $\pi$
```

3.3 Neuro-Symbolic Model

With the various components laid out, we can now present the full Embed2Sym training and inference algorithms. These are listed in Algorithm 1 and 2, respectively. An example inference process is also depicted in Figure 1 (right). In the algorithms we use some notations that we define here.

The perception component, g, is a cartesian product of perception networks applied to each raw input individually. If C_i is the latent concept corresponding to input x_i, then:

$$\begin{aligned} g(X) &= g(x_1 \times x_2 \times \cdots \times x_m) = \\ &= g_{C_1}(x_1) \times g_{C_2}(x_2) \times \cdots \times g_{C_m}(x_m) = \\ &= \vec{v}_1 \times \vec{v}_2 \times \cdots \times \vec{v}_m = \vec{v} \end{aligned} \tag{3}$$

K_C is a clustering function associated with latent concept C. It maps vectors from the embedding space associated with C to their cluster index. Formally, $K_C : \mathbb{R}^{k_C} \to \{1, ..., |S_C|\}$ where k_C is the dimensionality of the embeddings space $C = (n_C, S_C)$. Generally, K_C will be applied to the embeddings of several raw inputs. K denotes the cartesian product of the K_C's analogously to g above. We have:

$$\begin{aligned} K(\vec{v}) &= K_{C_1}(\vec{v}_1) \times K_{C_2}(\vec{v}_2) \times \cdots \times K_{C_m}(\vec{v}_m) = \\ &= (i_1, i_2, ..., i_m) = I \end{aligned} \tag{4}$$

π_C is the mapping of cluster indices to value indices for the latent concept C. π_C is always a permutation of $\{1, ..., |S_C|\}$. π is defined analogously to g and K:

$$\begin{aligned} \pi(I) &= \pi_{C_1}(i_1) \times \pi_{C_2}(i_2) \times \cdots \times \pi_{C_m}(i_m) = \\ &= (j_1, j_2, ..., j_m) = J \end{aligned} \tag{5}$$

Algorithm 2 Embed2Sym Inference

```
Input: $X, g, K, \pi, P, target$
Output: $y$
 1: $\vec{v} \leftarrow g(X)$
 2: $I \leftarrow K(\vec{v})$
 3: $J \leftarrow \pi(I)$
 4: $Facts \leftarrow \emptyset$
 5: for $j \in J$ do
 6:     Let $s$ be the latent concept value of $C$ corresponding to $j$
 7:     Let $x_i$ be the raw input name corresponding to $j$
 8:     Add $holds(0, n_C, x_i, s)$ to $Facts$
 9: end for
10: Let $A$ be an answer set of $P \cup Facts$
11: return $y$ s.t. $holds(0, target, y) \in A$
```

In line 2 of Algorithm 1 we make a selection of a subset of the dataset $\mathcal{D}$. This is motivated by the fact that both K-Means clustering and cluster labelling using Clingo are quite data efficient. These steps can therefore be sped up by selecting a much smaller subset of $\mathcal{D}$ than was required for training the perception module. Selection would typically be done by uniformly sampling from $\mathcal{D}$ without replacement.

An important aspect of Embed2Sym is that P_h of Algorithm 1 and P of Algorithm 2 need not be the same. This allows the rules of the reasoning component to be changed, as long as they are still concerned with the same latent concepts. For example, if the model was trained on the addition of MNIST images, it can perform multiplication as well at inference time. Hence, despite starting with a non-generalizing neural model, we achieved the same level of generalization other neuro-symbolic systems have, and the same level of interpretability.

4 Experiments

In this section, we present results following an evaluation of our Embed2Sym implementation in benchmark tasks involving both perception and reasoning. In designing these experiments, we aimed to answer the following questions:

1. Is Embed2Sym capable of solving tasks involving both perception and reasoning?
2. Is Embed2Sym able to scale to more challenging tasks involving a larger number of raw inputs?
3. Can Embed2Sym cope with challenging perception tasks?
4. How does Embed2Sym compare to other neuro-symbolic systems in terms of accuracy and training time?

To answer questions 1 & 2, we tested Embed2Sym on three benchmark tasks introduced before to evaluate neuro-symbolic systems, namely MNIST Addition, Member (Tsamoura, Hospedales, and Michael 2021) and Forth Sort (Bošnjak et al. 2017). We chose these tasks in particular as a means of testing the scalability of Embed2Sym when confronted with a large number of raw inputs, and increasing this number is straightforward for both. To answer question 3, we took the same approach previously introduced to benchmark the ABL-Sim algorithm (Huang et al. 2021) and created a CIFAR-10 Addition task as a more challenging version of the MNIST Addition task. To answer question 4, we also compare our models to two state-of-the-art neuro-symbolic systems, DeepProbLog and NeurASP, as they are both well known and have code available that allows for easy adaptation to the tasks we chose. When reporting results, we use the shorthand notations "E2S - FN", "E2S - NS", "DPLog" to refer to the fully neural, neuro-symbolic models and DeepProbLog, respectively. Our implementation of Embed2Sym was developed in Python3.8 using the TensorFlow framework, and can be found at the following link: https://github.com/YanivAspis/Embed2Sym.

N	1	2	3	4	15
E2S - FN	97.40	82.10	22.50	00.73	00.00
E2S - NS	97.73	94.31	93.90	91.61	66.40
DPLog	96.02	T/O	T/O	T/O	T/O
NeurASP	96.07	95.00	T/O	T/O	T/O

Table 1: Average test sum accuracy (%) on the MNIST Addition task for varying number of digits. T/O stands for "timeout".

Experiments were performed on a system running Ubuntu 18.04.6 with the following specifications: Intel i7-6700 CPU @ 3.40 GHz, 8 GB RAM and a single GeForce GTX 970 GPU. Hyperparameters for the experiments were determined by performance on a separate validation set separate from the test set. The K-Means and logical optimisation steps were performed using a subset of the dataset of 100 samples ($\mathcal{D}_{\in}$ of algorithm 1). We ran each experiment 5 times and report the average test accuracy and training time to account for the effects of random initialisation of the weights. If a system takes longer than 4 hours to train for any run, we stop training and declare a timeout.

4.1 MNIST Addition

We have used the MNIST Addition task as a running example throughout the paper to illustrate our approach. Here we consider the generalized task that allows for the addition of numbers with multiple digits. Specifically, each summand has N digits and so the number of raw inputs the model receives is $2N$. For the fully neural model, we can follow the architecture depicted in Figure 1 for small values of N. However, as N grows, the number of target labels (possible sums) grows exponentially, resulting in an output layer of exponential size. To resolve this, the reasoning component outputs each digit of the sum separately, so the number of output neurons grows linearly. Otherwise, the architecture is unchanged, where the g_{digit} component is applied to each of the $2N$ MNIST images, and their embeddings are concatenated before the reasoning component is applied. We generated for every N a training set of 30,000 samples, 500 validation samples and 5000 test samples. Results are summarised in Tables 1, 2 and 3. The results reported in each column are for training and testing on the number of digits appearing at the top.

Table 1 shows that Embed2Sym is able to produce models that solve this task with close or better accuracy than that

N	1	2	3	4	15
E2S - NS	98.86	98.54	98.96	98.92	98.64
DPLog	98.04	T/O	T/O	T/O	T/O
NeurASP	98.02	98.72	T/O	T/O	T/O

Table 2: Average test digit accuracy (%) on the MNIST Addition task for varying number of digits. T/O stands for "timeout".

N	1	2	3	4	15
E2S - FN	38.2	65.0	93.7	115.7	412.3
E2S - NS	40.9	69.0	99.6	122.6	434.1
DPLog	1033.4	T/O	T/O	T/O	T/O
NeurASP	919.7	3414.6	T/O	T/O	T/O

Table 3: Average training time (sec) on the MNIST Addition task for varying number of digits. Training time of "E2S - FN" includes the training time of "E2S - NS". T/O stands for "timeout". Results reported here for DeepProbLog are after only 1 epoch of training.

of competing systems. For $N = 1$ the fully neural model performs quite well, despite not having the advantage of being given any knowledge on the task. Note that in order to get the answer correct, the fully neural model must get each digit of the sum correct. Table 2 confirms that even though the fully neural scores poorly on larger values of N, digit accuracy remains high, meaning the perception network is still well trained. As N grows, the reasoning task becomes very difficult, as statistically speaking the network is likely to get at least one digit in the sum wrong. Therefore the fully neural model by itself is incapable of solving the task for large N in a satisfying manner.

However, all of this is solved once the model is lifted into a neuro-symbolic form. Despite not succeeding in the downstream task, clusters still form in the perception network's embedding space, and this can be exploited by Embed2Sym to create a neuro-symbolic model fully capable of adding MNIST numbers of large lengths. Note that for $N = 15$, a downstream average task accuracy of 66.40% is due to the statistical independence of predicting each individual input digit. An average accuracy of 98.64% in predicting each of 30 individual inputs correctly corresponds to a theoretical overall accuracy of 66.31% in predicting them all at once. Hence a result of 66.40% is in fact as high as one would expect for such a length.

The results suggest two important conclusions. First, even though the reasoning component did not achieve good accuracy, Embed2Sym is still able to extract the latent concepts from the perception component's embedding space. In this sense, the neuro-symbolic model can be used to improve upon the fully neural one in tasks where the reasoning steps may be more complex, by exploiting background knowledge. Second, this improvement can be achieved using only a limited portion of the dataset.

While other neuro-symbolic systems also provide a neural network with generalization capabilities, we see the advantage of Embed2Sym in its scalability in terms of training time. For small N, training is still feasible for NeurASP and DeepProbLog. NeurASP trains faster than DeepProbLog as it relies on the fast Clingo engine, being able to train on 2 digit addition within the time limit. However, as we move to 3 digits or more, training becomes infeasible to both NeurASP and DeepProbLog, while Embed2Sym can train up to $N = 15$ within minutes, and theoretically could go higher. Analysing the results deeper, we see that Embed2Sym spends most of its training time at the fully neural stage, with the process of lifting to a neuro-symbolic one only taking a few more seconds. In fact, our analysis shows that those few seconds are mostly spent computing the embeddings using the perception component (lines 6-10 of Algorithm 1), while K-Means clustering and Clingo each take less than a second on their part. This suggests training time can be further reduced by decreasing the number of epochs spent training the fully neural model, although at the possible cost of reduced accuracy.

	Task (%)	Image (%)	Training Time
E2S - FN	84.68	—	5883.6
E2S - NS	84.45	91.74	5908.6
DPLog	55.29	73.75	(T/O) 14400.0
NeurASP	6.52	12.75	(T/O) 14400.0

Table 4: Results for the CIFAR-10 Single Digit Addition experiment. Task % = Test accuracy in predicting the sum correctly. Image % = Test accuracy in predicting the digit (object) correctly. The fully neural model produces embeddings for the latent concepts and therefore cannot be assessed on Image Accuracy. Training time is in seconds.

4.2 CIFAR-10 Addition

CIFAR-10 (Krizhevsky and Hinton 2009) is a dataset of images depicting 10 classes of objects including animals and vehicles. In this task we consider each class to be a digit instead. We created a dataset for CIFAR-10 Addition where the MNIST images have been replaced with images from CIFAR-10 and the sum label is computed based on the digit each image label has been mapped to. Hence, the task involves the same level of reasoning as MNIST Addition but with a far more complex perception task. For the perception component, we use a ResNet56 architecture (He et al. 2016). If a system does not converge within 4 hours, we declare a timeout and report the accuracy achieved at that point.

Results are summarised in Table 4. The experiment confirms Embed2Sym is capable of learning even in this more difficult setting, completing training in less than 2 hours. DeepProbLog did not achieve satisfying accuracy within the time limit, although results could be higher if more time is given. NeurASP was unable to learn as it could only train for 1 epoch. We suspect that the limitation to a batch size of 1 is also affecting the training process negatively.

4.3 Member

In the Member task (Tsamoura, Hospedales, and Michael 2021), we are given a set of MNIST images, and a digit in symbolic form. We must determine if the digit appears in

N	3	4	5	20
E2S - FN	98.67	98.22	98.57	99.49
E2S - NS	96.82	96.51	97.53	93.46
DPLog	97.71	97.23	88.52	86.75
NeurASP	97.41	T/O	T/O	T/O

Table 5: Average test membership accuracy (%) on the Member task for a varying size of the set. T/O stands for "timeout".

N	3	4	5	20
E2S - NS	94.52	94.23	94.33	90.54
DPLog	96.94	95.94	78.85	79.46
NeurASP	95.36	T/O	T/O	T/O

Table 6: Average test digit accuracy (%) on the Member task for varying number of digits. T/O stands for "timeout".

one of the images in the set (In other words, if it is a member of the set). For sets of N elements, the task has the following specification:

- **Latent Concept:** ("$digit$", $\{0, 1, ..., 9\}$)
- **Raw inputs:** x_1, x_2, ..., x_N, each a 28x28 image.
- **Symbolic input:** t, one of the values '0' to '9'.
- **Target labels:** "True" and "False" (membership).

From a time complexity point of view, this task is interesting as the number of ways one can interpret the set of raw inputs increases exponentially with N. However, the label gives minimal information: If it is true, we know one of the images is labelled t, but we do not know which. If it is false, all images must not be labelled t, but we have no further information about their label. Therefore the label does not eliminate the exponential number of sets that can be interpreted. We generated for every N 10,000 training samples, 1000 validation samples and 2000 test samples.

For the fully neural model, we keep the perception component unchanged. The symbolic input t is fed directly into the reasoning component, using one-hot encoding to represent the 10 possible values of t. This one-hot vector is transformed using fully connected layers to an embedding vector, which is joined to the other embedding vectors being fed into the h component. For h, we use an attention-like mechanism where the embedding of t is compared to the embedding of each x_i, and the results are then passed through multiple layers before the true/false prediction is made. For all N, we train the fully neural model for 30 epochs.

Results of this experiment are listed in Tables 5, 6 and 7. In terms of accuracy, we see that Embed2Sym again achieves comparable or better accuracy to competing systems. In this case, the fully neural model is slightly better than the neuro-symbolic one, especially for $N = 20$. This is most likely due to its more relaxed representation. The fully neural model can retrieve the correct answer to the question of membership, without having to commit to the exact identities of the images. For example, if it is asked if the digit 4 is in the set, it can answer "yes" as long as it believes

N	3	4	5	20
E2S - FN	33.2	40.0	51.6	169.2
E2S - NS	35.6	42.6	54.9	177.2
DPLog	190.4	239.8	304.2	998.9
NeurASP	2044.4	T/O	T/O	T/O

Table 7: Average training time (sec) on the Member task for varying number of digits. Training time of "E2S - FN" includes the training time of "E2S - NS". T/O stands for "timeout".

there are several images that could be a 4. It need not decide between them. In contrast, the neuro-symbolic model must assign a digit to each image. While this does mean that the switch from the neural model to the neuro-symbolic one comes with some cost of accuracy, there is a gain in the form of sound reasoning and interpretability. The ability to correctly predict membership without correctly predicting all set members also explains why all systems demonstrate lower digit accuracy than one would expect based on their accuracy on the downstream task.

Once again, Embed2Sym is much faster to train than competing systems. NeurASP is unable to scale beyond small values, timing out already at $N = 4$. This is due to the time spent by NeurASP computing all answer sets that satisfy a label, as this number grows exponentially with N. DeepProbLog, on the other hand, relies only on the various proof paths to its query, which grows linearly with N. This allows DeepProbLog to scale well on the Member task, while other tasks such as MNIST Addition remain infeasible due to the exponential number of proofs. This is in contrast to the MNIST Addition task, where NeurASP performed better, showing that the scalability on tasks can vary significantly between systems. However, Embed2Sym is able of scaling well on both tasks, and is about 5-6 times faster than DeepProbLog. We also noticed that DeepProbLog had more difficulty training for higher values of N, at times failing to converge, which explains the lower average accuracy.

4.4 Forth Sort

The experiments so far have all been concerned with classification of images to latent labels. However Embed2Sym can be applied in a broader range of settings. To demonstrate this, we performed the Forth Sort experiment (Bošnjak et al. 2017) that is concerned with the induction of an operation, namely the decision to swap two elements in a list during sorting. The input to this task is a list of symbolic digits. Note the assumption that we do not know how to compare two digits, and therefore cannot use this as background knowledge. Instead, a "perception" network is employed to learn if two digits should be swapped or not. The latent concept "swap" therefore has two values: true and false. The perception network receives every pair of digits in the list and outputs an embedding. The reasoning network takes these embeddings and outputs an approximate permutation matrix to apply to the original list, and the sorted list is used as a label. The background knowledge for the task involves sorting based on the decision of swap/not swap for every

N	4	5	6	7	8
E2S - NS	13.6	31.4	50.2	74.6	277.4
DPLog (B)	70.2	271.4	2277.3	T/O	T/O
DPLog (Q)	27.7	213.5	791.3	1602.8	T/O

Table 8: Average training time (sec) for the Forth Sort task on various list lengths. Time to reach 100% accuracy on the test set is reported. T/O stands for "timeout". B/Q = Bubble/Quick Sort.

possible pair of digits.

Results are reported in Table 8. Following the DeepProbLog paper, we report the time it takes to reach 100% accuracy on a test set. In the DeepProbLog paper results for Bubble Sort alone are reported. As we are concerned with training time, we also report results for DeepProbLog using Quick Sort. The results not only confirm that Embed2Sym can be used to solve the Forth Sort task, but can do so much faster than DeepProbLog.

5 Discussion

With regards to the four questions raised at the beginning of section 4, we can now provide the following answers: 1) Embed2Sym successfully solves tasks involving perception and reasoning, as confirmed by the Addition, Member and Forth Sort experiments. 2) Embed2Sym is capable of scaling to tasks involving dozens of raw inputs and potentially more. In particular, we have shown Embed2Sym is capable of solving 15-digit MNIST addition and 20-digit Member. 3) Embed2Sym can learn even when the perception task is more complex, as confirmed by the CIFAR-10 Addition experiment. 4) Embed2Sym achieves similar or better test accuracy to competing systems on all the tested tasks, while training faster by several orders of magnitude and scaling to tasks infeasible to other systems.

Embed2Sym offers a more scalable approach to neuro-symbolic reasoning by embracing a different philosophy than other systems. While most approaches attempt to turn logical formulas into differentiable operations, Embed2Sym uses the neural and symbolic components as a means to complement each other. The difficulty in creating a scalable version of differentiable logic is highlighted by the combinatorial issues existing systems encounter. Backpropagation through a differentiable logic gate requires answering the question "Which input values would result in the correct output". Systems such as NeurASP and ABL link this question to logical abduction. In the case of ABL this is done only in the context of definite programs, whereas NeurASP uses the full expressivity of Answer Set Programming. The difficulty in using abduction is that these systems need to find the full set of solutions which in general can be exponential in size. DeepProbLog takes a proof-theoretical view to this question, constructing a (weighted) boolean formula that satisfies a given query. The time complexity of this construction can be $\#P$-complete (Fierens et al. 2015). We see these difficulties manifest in practice particularly in the MNIST Addition experiment, where even for small values of N, training becomes infeasible for both NeurASP and DeepProbLog.

Embed2Sym solves this issue by embracing the approximate nature of neural network inference. The training of the fully neural model creates a structure in embedding space that a symbolic system can later identify and exploit. This gives a symbolic reasoning system access to inference on raw data while circumventing the difficulties arising from differentiable logic operations. The logical optimisation step, which is the critical step in creating the neuro-symbolic model, scales well with an increased number of raw inputs because it builds upon the advances in solving an optimisation problem in Answer Set Programming. We see from experiment that the logical optimisation step is very fast (less than 1 second), and it is in fact the training of the neural network that has a significant effect on training time.

It should be noted that while Embed2Sym works by identifying latent concepts discovered by a neural network, it should not be understood as an explainability method. Embed2Sym makes no attempt at explaining why the perception network associates a given raw input with a latent concept value. It merely states that it did. Specifically, it is not designed to discover if this relationship is due to a statistical correlation rather than a causal relation. Note that other neuro-symbolic systems do not address this issue either.

Future work will be directed at further development of the Embed2Sym method and testing on more challenging domains. Embed2Sym may benefit from more sophisticated methods of discovering cluster information in embedding space (Guo, Lin, and Ye 2021). In the experiments in this paper, we used a simple MLP for the reasoning component during the neural training stage. But in general more sophisticated models could be required, as reasoning gets more complex. For example, reasoning involving differentiating and integrating mathematical functions has been shown to be possible using tree generating Seq2Seq models (Lample and Charton 2020). We also wish to explore the possibility of learning the logic program from data, rather than being provided with the rules.

6 Summary

We introduced Embed2Sym, a novel approach to neuro-symbolic reasoning that is capable of solving tasks involving both perception and reasoning, interpretable and can generalize to other downstream tasks. Most importantly, our system scales far better than competing systems, being capable of solving tasks involving a far larger amount of raw inputs and to more complex perception tasks. Experiments confirm Embed2Sym is much faster to train than competing systems and can scale to many raw inputs, being able to train on the addition of 15 digits numbers (at least), far larger than any competing system has so far achieved. Results on CIFAR-10 Addition confirm the system is capable of training on more complex perception tasks, while other systems time out. Future work will be directed toward experimentation in more challenging domains, experimenting with more sophisticated clustering and reasoning networks, and exploring the possibility of learning rules from data.

References

Badreddine, S.; d'Avila Garcez, A.; Serafini, L.; and Spranger, M. 2022. Logic tensor networks. *Artificial Intelligence* 303:103649.

Besold, T. R.; d'Avila Garcez, A.; Bader, S.; Bowman, H.; Domingos, P.; Hitzler, P.; Kuehnberger, K.-U.; Lamb, L. C.; Lowd, D.; Lima, P. M. V.; de Penning, L.; Pinkas, G.; Poon, H.; and Zaverucha, G. 2017. Neural-symbolic learning and reasoning: A survey and interpretation.

Bošnjak, M.; Rocktäschel, T.; Naradowsky, J.; and Riedel, S. 2017. Programming with a differentiable forth interpreter. In Precup, D., and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 547–556. PMLR.

Caron, M.; Bojanowski, P.; Joulin, A.; and Douze, M. 2018. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 132–149.

Dai, W.-Z.; Xu, Q.; Yu, Y.; and Zhou, Z.-H. 2019. Bridging machine learning and logical reasoning by abductive learning. *Advances in Neural Information Processing Systems* 32.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, volume 7, 2462–2467. Hyderabad.

Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X.; et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, 226–231.

Fellbaum, C., ed. 1998. *WordNet: An Electronic Lexical Database*. Language, Speech, and Communication. Cambridge, MA: MIT Press.

Fierens, D.; Van den Broeck, G.; Renkens, J.; Shterionov, D.; Gutmann, B.; Thon, I.; Janssens, G.; and De Raedt, L. 2015. Inference and learning in probabilistic logic programs using weighted boolean formulas. *Theory and Practice of Logic Programming* 15(3):358–401.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2017. Multi-shot ASP solving with clingo. *CoRR* abs/1705.09811.

Guo, W.; Lin, K.; and Ye, W. 2021. Deep embedded k-means clustering. In *2021 International Conference on Data Mining Workshops (ICDMW)*, 686–694. Los Alamitos, CA, USA: IEEE Computer Society.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Huang, Y.-X.; Dai, W.-Z.; Cai, L.-W.; Muggleton, S.; and Jiang, Y. 2021. Fast abductive learning by similarity-based consistency optimization. *Advances in Neural Information Processing Systems* 34.

Koh, P. W.; Nguyen, T.; Tang, Y. S.; Mussmann, S.; Pierson, E.; Kim, B.; and Liang, P. 2020. Concept bottleneck models. In *International Conference on Machine Learning*, 5338–5348. PMLR.

Krizhevsky, A., and Hinton, G. 2009. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario.

Lample, G., and Charton, F. 2020. Deep learning for symbolic mathematics. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Law, M.; Russo, A.; and Broda, K. 2020. The ilasp system for inductive learning of answer set programs. *arXiv preprint arXiv:2005.00904*.

LeCun, Y.; Cortes, C.; and Burges, C. 2010. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2.

Manhaeve, R.; Dumančić, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2021. Neural probabilistic logic programming in deepproblog. *Artificial Intelligence* 298:103504.

Minervini, P.; Bosnjak, M.; Rocktäschel, T.; and Riedel, S. 2018. Towards neural theorem proving at scale. *arXiv preprint arXiv:1807.08204*.

Moradi Fard, M.; Thonet, T.; and Gaussier, E. 2020. Deep k-means: Jointly clustering with k-means and learning representations. *Pattern Recognition Letters* 138:185–192.

Palm, R.; Paquet, U.; and Winther, O. 2018. Recurrent relational networks. *Advances in Neural Information Processing Systems* 31.

Riegel, R.; Gray, A.; Luus, F.; Khan, N.; Makondo, N.; Akhalwaya, I. Y.; Qian, H.; Fagin, R.; Barahona, F.; Sharma, U.; et al. 2020. Logical neural networks. *arXiv preprint arXiv:2006.13155*.

Sarker, M. K.; Zhou, L.; Eberhart, A.; and Hitzler, P. 2021. Neuro-symbolic artificial intelligence: Current trends. *arXiv preprint arXiv:2105.05330*.

Speer, R.; Chin, J.; and Havasi, C. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, 4444–4451. AAAI Press.

Tsamoura, E.; Hospedales, T.; and Michael, L. 2021. Neural-symbolic integration: A compositional perspective. *Proceedings of the AAAI Conference on Artificial Intelligence* 35(6):5051–5060.

Wang, D.; Jamnik, M.; and Liò, P. 2018. Investigating diagrammatic reasoning with deep neural networks. In *Diagrams*.

Winters, T.; Marra, G.; Manhaeve, R.; and De Raedt, L. 2021. Deepstochlog: Neural stochastic logic programming. *arXiv preprint arXiv:2106.12574*.

Xia, X.; Togneri, R.; Sohel, F.; Zhao, Y.; and Huang, D. 2019. A survey: neural network-based deep learning for acoustic event detection. *Circuits, Systems, and Signal Processing* 38(8):3433–3453.

Xu, J.; Zhang, Z.; Friedman, T.; Liang, Y.; and Van den Broeck, G. 2018. A semantic loss function for deep learning with symbolic knowledge. In Dy, J., and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 5502–5511. PMLR.

Yang, Z.; Ishay, A.; and Lee, J. 2020. Neurasp: Embracing neural networks into answer set programming. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 1755–1762. International Joint Conferences on Artificial Intelligence Organization. Main track.

Zhao, Z.-Q.; Zheng, P.; Xu, S.-T.; and Wu, X. 2019. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems* 30(11):3212–3232.