NeuPSL: Neural Probabilistic Soft Logic

Connor Pryor[1] , **Charles Dickens**[1] , **Eriq Augustine** [1] , **Alon Albalak** [2] ,
William Yang Wang [2] and **Lise Getoor** [1]
[1] UC Santa Cruz
[2] UC Santa Barbara
cfpryor@ucsc.edu, cadicken@ucsc.edu, eaugusti@ucsc.edu, alon_albalak@ucsb.edu,
william@cs.ucsb.edu, getoor@ucsc.edu

Abstract

In this paper, we introduce *Neural Probabilistic Soft Logic* (NeuPSL), a novel neuro-symbolic (NeSy) framework that unites state-of-the-art symbolic reasoning with the low-level perception of deep neural networks. To model the boundary between neural and symbolic representations, we propose a family of energy-based models, *NeSy Energy-Based Models*, and show that they are general enough to include NeuPSL and many other NeSy approaches. Using this framework, we show how to seamlessly integrate neural and symbolic parameter learning and inference in NeuPSL. Through an extensive empirical evaluation, we demonstrate the benefits of using NeSy methods, achieving upwards of 30% improvement over independent neural network models. On a well-established NeSy task, MNIST-Addition, NeuPSL demonstrates its joint reasoning capabilities by outperforming existing NeSy approaches by up to 10% in low-data settings. Furthermore, NeuPSL achieves a 5% boost in performance over state-of-the-art NeSy methods in a canonical citation network task with up to a 40 times speed up.

1 Introduction

The field of artificial intelligence (AI) has long sought a symbiotic union of neural and symbolic methods. Neural-based methods excel at low-level perception and learn from large training data sets but struggle with interpretability and generalizing in low-data settings. Meanwhile, symbolic methods can effectively use domain knowledge, context, and common sense to reason with limited data but have difficulty representing complex low-level patterns. Recently, neuro-symbolic computing (NeSy) [Besold *et al.*, 2017; d'Avila Garcez *et al.*, 2019; De Raedt *et al.*, 2020] has emerged as a promising new research area with the goal of developing systems that integrate neural and symbolic methods in a mutually beneficial manner.

A neural and symbolic union has the potential to yield two highly desirable capabilities - the ability to perform structured prediction (*joint inference*) across related examples that possess complex low-level features and the ability to jointly learn (*joint learning*) and adapt parameters over neural and symbolic models simultaneously. For instance, predicting the result of competitions between teams using historical performance statistics in a tournament bracket requires methods to perform joint inference to reason over low-level trends and avoid inconsistencies such as two first-place finishes. Unfortunately, joint inference problems quickly grow in complexity as the output space typically increases combinatorially. For example, in the tournament setting, as the number of entries increases, the number of potential solutions grows exponentially ($O(2^n)$). An open challenge in the NeSy community is scaling joint inference and reasoning.

This paper introduces *Neural Probabilistic Soft Logic* (NeuPSL), a novel NeSy method that integrates deep neural networks with a symbolic method designed for fast joint learning and inference. NeuPSL extends probabilistic soft logic (PSL) [Bach *et al.*, 2017], a state-of-the-art and scalable probabilistic programming framework that can reason statistically (using probabilistic inference) and logically (using soft rules). PSL has been shown to excel in a wide variety of tasks, including natural language processing [Beltagy *et al.*, 2014; Deng and Wiebe, 2015; Liu *et al.*, 2016; Rospocher, 2018], data mining [Alshukaili *et al.*, 2016; Kimmig *et al.*, 2019], recommender systems [Kouki *et al.*, 2015], knowledge graph discovery [Pujara *et al.*, 2013], fairness modeling [Farnadi *et al.*, 2019; Dickens *et al.*, 2020], and causal reasoning [Sridhar *et al.*, 2018]. The key innovation of NeuPSL is a new class of predicates that rely on neural network output for their values. This change fundamentally alters the learning and joint inference problems by requiring efficient integrated symbolic and neural parameter learning. The appeal of this extension is that it allows for the semantics and implementation of the symbolic language to remain the same as PSL, while also incorporating the added benefit of low-level neural perception. To gain a deeper understanding of optimizing the symbolic and neural parameters, we propose a versatile mathematical framework, Neuro-Symbolic Energy-Based Models (NeSy-EBMs), that enables many NeSy systems to utilize established Energy-Based Model learning losses and algorithms. Utilizing this theory and leveraging the unique relaxation properties of PSL, we show that a gradient over these neural predicates can be calculated and passed back to common back-propagation engines such as PyTorch or Tensorflow, allowing for scalable end-to-end gradient training.

Our key contributions include: 1) We define *Neuro-Symbolic Energy-Based Models* (NeSy-EBMs), a family of energy-based models, and show how they provide a foundation for describing, understanding and comparing NeSy systems. 2) We introduce NeuPSL, describe how it fits into the NeSy ecosystem and supports scalable joint inference, and show how it can be trained end-to-end using a joint energy-based learning loss. 3) We perform extensive evaluations over two image classification tasks and two citation network datasets. Our results show NeuPSL consistently outperforms existing approaches on joint inference tasks and can more efficiently leverage structure, particularly in low-data settings.

2 Related Work

Neuro-symbolic computing (NeSy) is an active area of research that aims to incorporate logic-based reasoning with neural networks [d'Avila Garcez *et al.*, 2002; Bader and Hitzler, 2005; d'Avila Garcez *et al.*, 2009; Serafini and d'Avila Garcez, 2016; Besold *et al.*, 2017; Donadello *et al.*, 2017; Yang *et al.*, 2017; Evans and Grefenstette, 2018; Manhaeve *et al.*, 2021; d'Avila Garcez *et al.*, 2019; De Raedt *et al.*, 2020; Lamb *et al.*, 2020; Badreddine *et al.*, 2022]. The advantages of NeSy systems include interpretability, robustness, and the ability to integrate various sub-problem solutions (such as perception, reasoning, and decision-making). For a thorough introduction to NeSy literature, we refer the reader to the excellent surveys by Besold *et al.* (2017) and De Raedt *et al.* (2020). In this section, we identify key NeSy research categories and provide a brief description of each.

Differentiable frameworks of logical reasoning: Methods in this category use neural networks' universal function approximation properties to emulate logical reasoning inside networks. Examples include: Rocktäschel and Riedel (2017), Bošnjak *et al.* (2017), Evans and Grefenstette (2018), and Cohen *et al.* (2020).

Constrained Output: These approaches enforce constraints or regularizations on the output of neural networks. Examples include: Hu *et al.* (2016), Diligenti *et al.* (2017), Donadello *et al.* (2017), Mehta *et al.* (2018), Xu *et al.* (2018), and Nandwani *et al.* (2019).

Executable logic programs: These approaches use neural models to build executable logical programs. Examples include Liang *et al.* (2017) and Mao *et al.* (2019). We highlight Logic Tensor Networks (LTNs) [Badreddine *et al.*, 2022], as we include this approach in our empirical evaluation. LTNs connect neural predictions into functions representing symbolic relations with real-valued or fuzzy logic semantics.

Neural networks as predicates: This line of work integrates neural networks and probabilistic reasoning by introducing neural networks as predicates in the logical formulae. This technique provides a very general and flexible framework for NeSy reasoning and allows for the use of multiple networks as well as the full incorporation of constraints and relational information. Examples include DASL [Sikka *et al.*, 2020], NeurASP [Yang *et al.*, 2020], Nuts&Bolts [Sachan *et al.*, 2018], DeepProbLog (DPL) [Manhaeve *et al.*, 2021], and our proposed method (Neural Probabilistic Soft Logic). DPL combines general-purpose neural networks with the probabilistic modeling of ProbLog [De Raedt *et al.*, 2007] in a way that allows for learning and inference over complex tasks, such as program induction. We include DPL in our empirical evaluation.

3 Neuro-Symbolic Energy-Based Models

With the success and growth of NeSy research, there is an increasing need for a common formalization of NeSy systems to accelerate the research and understanding of the field. We fill this need with a general mathematical framework, *Neuro-Symbolic Energy-Based Models* (NeSy-EBMs). NeSy-EBMs encompass previous approaches and establishes the foundation of our approach. Energy-Based Models (EBMs) [LeCun *et al.*, 2006] measure the compatibility of a collection of observed (or input) variables $\mathbf{x} \in \mathcal{X}$ and target (or output) variables $\mathbf{y} \in \mathcal{Y}$ with a scalar-valued *energy function*: $E : \mathcal{Y} \times \mathcal{X} \rightarrow \mathbb{R}$. Low energy states of the variables represent high compatibility. Prediction or *inference* in EBMs is performed by finding the lowest energy state of the variables $\mathbf{y}$ given $\mathbf{x}$. Energy functions are parameterized by variables $\mathbf{w} \in \mathcal{W}$, and *learning* is the task of finding a parameter setting that associates low energy to correct solutions.

Building on the well-known EBM framework, NeSy-EBMs are a family of EBMs that integrate neural architectures with explicit encodings of symbolic relations. The input variables are organized into neural, $\mathbf{x}_{nn} \in \mathcal{X}_{nn}$, and symbolic, $\mathbf{x}_{sy} \in \mathcal{X}_{sy}$, vectors. Furthermore, the parameters of the energy function, $\mathbf{w}$, are partitioned into neural weights, $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$, and symbolic weights, $\mathbf{w}_{sy} \in \mathcal{W}_{sy}$. Formally,

Definition 1 (NeSy-EBM)**.** *Let $\mathbf{y} \in \mathcal{Y}$ and $\mathbf{x}_{sy} \in \mathcal{X}_{sy}$ be vectors of variables with symbolic interpretations. Let $\mathbf{g}_{nn}$ be neural networks with **neural weights** $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$ and inputs $\mathbf{x}_{nn} \in \mathcal{X}_{nn}$. A **symbolic potential** is a function of $\mathbf{y}$, $\mathbf{x}_{sy}$, and $\mathbf{g}_{nn}(\cdot)$ parameterized by **symbolic weights** $\mathbf{w}_{sy} \in \mathcal{W}_{sy}$: $\psi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \in \mathbb{R}$. A **NeSy-EBM energy function** is a mapping of a vector of m symbolic potential outputs, $\Psi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) = [\psi_i(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))]_{i=1}^{m}$, to a real value: $E(\Psi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn})) \in \mathbb{R}$.*

NeSy-EBMs are differentiated from one another by the instantiation process, the form of the symbolic potentials, and the definition of the energy function. In appendix, we formally show how two NeSy systems DeepProbLog (DPL) [Manhaeve *et al.*, 2018] and Logic Tensor Networks (LTNs) [Badreddine *et al.*, 2022] fit into the NeSy-EBM framework. In summary, DPL uses neural network outputs to specify event probabilities that are used in logical formulae defining probabilistic dependencies. The definition of the DPL symbolic potentials and energy function are tied to the inference task; a different definition of the symbolic potential and energy function is used to implement marginal versus MAP inference. For marginal, the most common DPL inference, symbolic potentials are functions of marginal probabilities, and the energy function is a joint distribution that is the sum of the symbolic potentials. LTNs instantiate a model which forwards neural network predictions into functions representing symbolic relations with real-valued or fuzzy logic seman-

tics. The fuzzy logic functions are symbolic potentials that are aggregated to define the energy function. The following section will introduce how our approach, NeuPSL, is instantiated as a NeSy-EBM. Using this common framework, understanding and theoretical advances can be made across NeSy approaches.

3.1 Joint Reasoning in NeSy-EBMs

We highlight two important categories of NeSy-EBM energy functions: *joint* and *independent*. Formally, an energy function that is additively separable over the output variables $\mathbf{y}$ is an *independent energy function*, i.e., corresponding to each of the n_y components of the output variable $\mathbf{y}$ there exists functions n_y functions $E_1(\mathbf{y}[1], \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$, $\cdots$, $E_{n_y}(\mathbf{y}[n_y], \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ such that

$$E(\cdot) = \sum_{i=1}^{n_y} E_i(\mathbf{y}[i], \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})).$$

While a function that is not separable over output variables $\mathbf{y}$ is a *joint energy function*. This categorization allows for an important distinction during inference and learning. Independent energy functions simplify inference and learning as finding an energy minimizer, $\mathbf{y}^*$, can be distributed across the independent functions E_i. In other words, the predicted value for a variable $\mathbf{y}[i]$ has no influence over that of $\mathbf{y}[j]$ where $j \neq i$ and can therefore be predicted separately, i.e., independently. However, independent energy functions cannot leverage some joint information that may be used to improve predictions. See appendix for further details.

4 Neural Probabilistic Soft Logic

Having laid the NeSy-EBM groundwork, we now introduce *Neural Probabilistic Soft Logic* (NeuPSL), a novel NeSy-EBM framework that extends the probabilistic soft logic (PSL) framework [Bach *et al.*, 2017]. At its core, NeuPSL leverages the power of neural networks' low-level perception by seamlessly integrating their outputs with a collection of symbolic potentials generated through a PSL program. Figure 1 provides a graphical representation of this process. The symbolic potentials and neural networks together define a *deep hinge-loss Markov random field* (*Deep-HL-MRF*), a tractable probabilistic graphical model that supports scalable convex joint inference. This section provides a comprehensive description of how NeuPSL instantiates its symbolic potentials and how the symbolic potentials are combined to define an energy function, while the following section details NeuPSL's end-to-end neural-symbolic inference, learning, and joint reasoning processes.

NeuPSL instantiates the symbolic potentials of its energy function using the PSL language where dependencies between relations and attributes of entities in a domain, defined as *atoms*, are encoded with weighted first-order logical clauses and linear arithmetic inequalities referred to as *rules*. To illustrate, consider a setting in which a neural network is used to classify the species of an animal in an image. Further, suppose there exists external information suggesting when two images may contain the same entity. The information linking the images may come from various sources, such as the images' caption or metadata indicating the images were captured by the same device within a short period of time. NeuPSL represents the neural network's animal classification of an image ($\texttt{Image}_1$) as a species ($\texttt{Species}$) with the atom $\textsc{Neural}(\texttt{Image}_1, \texttt{Species})$ and the probability that two images ($\texttt{Image}_1$ and $\texttt{Image}_2$) contain the same entity with the atom $\textsc{SameEntity}(\texttt{Image}_1, \texttt{Image}_2)$. Additionally, we represent NeuPSL's classification of $\texttt{Image}_2$ with $\textsc{Class}(\texttt{Image}_2, \texttt{Species})$. The following weighted logical rule in NeuPSL represents the notion that two images identified as the same entity may also be of the same species:

$$\begin{aligned} w : \textsc{Neural}(\texttt{Image}_1, \texttt{Species}) \\ \wedge\ \textsc{SameEntity}(\texttt{Image}_1, \texttt{Image}_2) \\ \rightarrow\ \textsc{Class}(\texttt{Image}_2, \texttt{Species}) \end{aligned} \tag{1}$$

The parameter w is the weight of the rule, and it quantifies its relative importance in the model. Note these rules can either be hard or soft constraints. Atoms and weighted rules are templates for creating symbolic potentials or soft constraints. To create these symbolic potentials, atoms and rules are instantiated with observed data and neural predictions. Atoms instantiated with elements from the data are referred to as *ground atoms*. Then, valid combinations of ground atoms substituted in the rules create *ground rules*. To illustrate, suppose that there are two images $\{Id1, Id2\}$ and three species classes $\{Cat, Dog, Frog\}$. Using the above data for cats would result in the following ground rules (analogous ground rules would be created for dogs and frogs):

$$\begin{aligned} w : \textsc{Neural}(Id1, Cat) \wedge \textsc{SameEntity}(Id1, Id2) \\ \rightarrow\ \textsc{Class}(Id2, Cat) \\ w : \textsc{Neural}(Id2, Cat) \wedge \textsc{SameEntity}(Id2, Id1) \\ \rightarrow\ \textsc{Class}(Id1, Cat) \end{aligned}$$

Ground atoms are mapped to either an observed variable, $x_{sy,i}$, target variable, y_i, or a neural function with inputs $\mathbf{x}_{nn}$ and parameters $\mathbf{w}_{nn,i}$: $g_{nn,i}(\mathbf{x}_{nn}, \mathbf{w}_{nn,i})$. Then, variables are aggregated into the vectors $\mathbf{x}_{sy} = [x_{sy_i}]_{i=1}^{n_x}$ and $\mathbf{y} = [y_i]_{i=1}^{n_y}$ and neural outputs are aggregated into the vector $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$. Ground rules are either logical (e.g., Equation 1) or arithmetic defined over $\mathbf{x}_{sy}$, $\mathbf{y}$, and $\mathbf{g}_{nn}$. These ground rules create one or more potentials $\phi(\cdot) \in \mathcal{R}$, where logical rules are relaxed using Łukasiewicz continuous valued logical semantics [Klir and Yuan, 1995]. Each potential $\phi(\cdot)$ is associated with a weight w_{psl} inherited from its instantiating rule. The potentials and weights from the instantiation process are used to define a member of a tractable class of graphical models, *deep hinge-loss Markov random fields* (Deep-HL-MRF):

Definition 2 (Deep Hinge-Loss Markov Random Field)**.** *Let $\mathbf{y} \in [0,1]^{n_y}$ and $\mathbf{x}_{sy} \in [0,1]^{n_x}$ be vectors of $[0,1]$ valued variables. Let $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$ be functions with corresponding parameters $\mathbf{w}_{nn} = [\mathbf{w}_{nn,i}]_{i=1}^{n_g}$ and inputs $\mathbf{x}_{nn}$. A **deep hinge-loss potential** is a function of the form*

$$\phi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) = \max(l(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})), 0)^{\alpha} \tag{2}$$

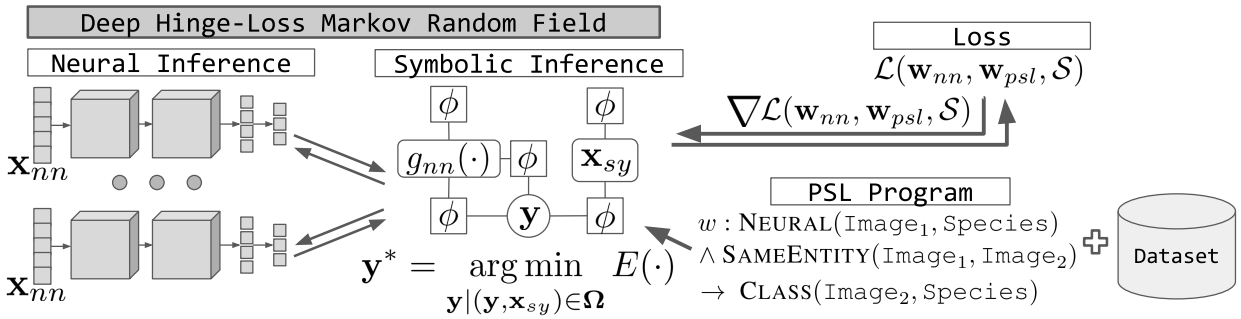


Figure 1: NeuPSL inference and learning pipeline.

*where $l(\cdot)$ is a linear function and $\alpha \in \{1, 2\}$. Let $\mathcal{T} = [t_i]_{i=1}^r$ denote an ordered partition of a set of m deep hinge-loss potentials: $\{\phi_1, \cdots, \phi_m\}$. For each partition t_i define $\Phi_i(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) := \sum_{j\in t_i} \phi_i(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn})$ and let $\mathbf{\Phi}(\mathbf{y}, \mathbf{x_{sy}}, \mathbf{x_{nn}}, \mathbf{w_{nn}}) := [\Phi_i(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn})]_{i=1}^r$. Further, let $\mathbf{w}_{psl} = [w_{psl,i}]_{i=1}^r$ be a vector of non-negative weights corresponding to the partition $\mathcal{T}$. Then, a **deep hinge-loss energy function** is*

$$E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = \mathbf{w}_{psl}^T \mathbf{\Phi}(\mathbf{y}, \mathbf{x_{sy}}, \mathbf{x_{nn}}, \mathbf{w_{nn}}) \tag{3}$$

*Further, let $\mathbf{c} = [c_i]_{i=1}^q$ be a vector of q linear constraints in standard form, defining the feasible set $\mathbf{\Omega} = \{\mathbf{y}, \mathbf{x}_{sy} \,|\, c_i(\mathbf{y}, \mathbf{x}_{sy}) \leq 0, \forall i \in \{0, \cdots, q\}\}$. Then a **deep hinge-loss Markov random field**, $\mathcal{P}$, with random variables $\mathbf{y}$ conditioned on $\mathbf{x}_{sy}$ and $\mathbf{x}_{nn}$ is a probability density of the form*

$$P(\mathbf{y}|\mathbf{x}_{sy}, \mathbf{x}_{nn}) = \begin{cases} \frac{\exp(-E(\cdot))}{\int_{\mathbf{y}|\mathbf{y},\mathbf{x}_{sy}\in\mathbf{\Omega}} \exp(-E(\cdot))d\mathbf{y}} & (\mathbf{y}, \mathbf{x}_{sy}) \in \mathbf{\Omega} \\ 0 & o.w. \end{cases}$$

Deep-HL-MRFs naturally fit into the NeSy-EBM framework. The symbolic potentials of deep-HL-MRFs are the aggregated and scaled deep hinge-loss potentials:

$$\begin{aligned} &\psi_{NeuPSL}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{psl}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\ &= \mathbf{w}_{psl}\Phi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) \end{aligned} \tag{4}$$

Then the energy function is the sum of symbolic potentials:

$$\begin{aligned} &E_{NeuPSL}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) \\ &= \sum_{i=1}^{r} \psi_{NeuPSL,i}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{psl}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \end{aligned} \tag{5}$$

5 NeuPSL Inference and Learning

There is a clear connection between neural and symbolic inference in NeuPSL that allows any neural architecture to interact with symbolic reasoning in a simple and expressive manner. The NeuPSL neural-symbolic interface and inference pipeline is shown in Figure 1. *Neural inference* is computing the output of the neural networks given the input $\mathbf{x}_{nn}$, i.e., computing $g_{nn,i}(\mathbf{x}_{nn}, \mathbf{w}_{nn,i})$ for all i. NeuPSL *symbolic inference* minimizes the energy function over $\mathbf{y}$:

$$\mathbf{y}^* = \underset{\mathbf{y}|(\mathbf{y},\mathbf{x}_{sy})\in\mathbf{\Omega}}{\arg\min} E(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) \tag{6}$$

Note that the hinge-loss potentials are convex in $\mathbf{y}$ and hence, with the common constraint enforcing symbolic parameters to be non-negative, i.e., $\mathbf{w}_{psl} > 0$, the energy function is convex in $\mathbf{y}$. Any scalable convex optimizer can be applied to solve (6). NeuPSL uses the alternating direction method of multipliers [Boyd *et al.*, 2010].

NeuPSL learning is the task of finding both neural and symbolic parameters, i.e., rule weights, that assign low energy to correct values of the output variables and higher energies to incorrect values. Learning objectives are functionals mapping an energy function and a set of training examples $\mathcal{S} = \{(\mathbf{y}_i, \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}) : i = 1, \cdots, P\}$ to a real-valued loss. As the energy function for NeuPSL is parameterized by the neural weights $\mathbf{w}_{nn}$ and symbolic weights $\mathbf{w}_{psl}$, we express the learning objective as a function of $\mathbf{w}_{nn}$, $\mathbf{w}_{psl}$, and $\mathcal{S}$: $\mathcal{L}(\mathcal{S}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$. Learning objectives follow the standard empirical risk minimization framework and are therefore separable over the training examples in $\mathcal{S}$ as a sum of per-sample loss functions $L_i(\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{nn,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$. Concisely, NeuPSL learning is the following minimization:

$$\begin{aligned} &\underset{\mathbf{w}_{nn},\mathbf{w}_{psl}}{\arg\min}\, \mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{psl}, \mathcal{S}) \\ &= \underset{\mathbf{w}_{nn},\mathbf{w}_{psl}}{\arg\min} \sum_{i=1}^{P} L_i(\mathbf{y}_i, \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) \end{aligned}$$

In the learning setting, variables $\mathbf{y}_i$ from the training set S are partitioned into vectors $\mathbf{y}_{i,t}$ and $\mathbf{z}_i$. The variables $\mathbf{y}_{i,t}$ represent variables for which there is a corresponding truth value, while $\mathbf{z}_i$ represent latent variables. Without loss of generality, we write $\mathbf{y}_i = (\mathbf{y}_{i,t}, \mathbf{z}_i)$.

There are multiple losses that one could motivate for optimizing the parameters of an EBM. Common losses, including the loss we present in this work, use the following terms:

$$\mathbf{z}_i^* = \underset{\mathbf{z}|((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x}_{sy,i})\in\mathbf{\Omega}}{\arg\min} E((\mathbf{y}_{i,t}, \mathbf{z}), \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$$

$$\mathbf{y}_i^* = \underset{\mathbf{y}|(\mathbf{y},\mathbf{x}_{sy,i})\in\mathbf{\Omega}}{\arg\min} E(\mathbf{y}, \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$$

In words, $\mathbf{z}_i^*$ and $\mathbf{y}_i^*$ are the lowest energy states given $(\mathbf{y}_{i,t}, \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i})$ and $(\mathbf{x}_{sy,i}, \mathbf{x}_{nn,i})$, respectively. A special case of learning is when the per-sample losses are not functions of $\mathbf{z}_i^*$ and $\mathbf{y}_i^*$, and more specifically, the losses do not require any subproblem optimization. We refer to this situation as *constraint learning*. Constraint learning reduces the time required per iteration at the cost of expressivity.

All interesting learning losses for NeuPSL are a composition of the energy function. Thus, a gradient-based learning algorithm will require the following partial derivatives: [1]

$$\frac{\partial E(\cdot)}{\partial \mathbf{w}_{psl}[i]} = \Phi_i(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn})$$
$$\frac{\partial E(\cdot)}{\partial \mathbf{w}_{nn}[i]} = \mathbf{w}_{psl}^T \nabla_{\mathbf{w}_{nn}[i]} \Phi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn})$$

Continuing with the derivative chain rule and noting the potential can be squared ($\alpha = 2$) or linear ($\alpha = 1$), the potential partial derivative with respect to $\mathbf{w}_{nn}[i]$ is the piece-wise defined function:[1]

$$\frac{\partial \phi(\cdot)}{\partial \mathbf{w}_{nn}[i]} = \begin{cases} \frac{\partial}{\partial \mathbf{g}_{nn}[i]} \phi(\cdot) \cdot \frac{\partial}{\partial \mathbf{w}_{nn}[i]} \mathbf{g}_{nn}[i](\cdot) & \alpha = 1 \\ 2 \cdot \phi(\cdot) \cdot \frac{\partial}{\partial \mathbf{g}_{nn}[i]} \phi(\cdot) \cdot \frac{\partial}{\partial \mathbf{w}_{nn}[i]} \mathbf{g}_{nn}[i](\cdot) & \alpha = 2 \end{cases}$$

$$\frac{\partial \phi(\cdot)}{\partial \mathbf{g}_{nn}[i]} = \begin{cases} 0 & \phi(\cdot) = 0 \\ \frac{\partial}{\partial g_{nn}[i]} l(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) & \phi(\cdot) > 0 \end{cases}$$

Since $l(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ is a linear function, the partial gradient with respect to $\mathbf{g}_{nn}[i]$ is trivial. With the partial derivatives presented here, standard backpropagation-based algorithms for computing gradients can be applied for both neural and symbolic parameter learning.

Energy Loss: A variety of differentiable loss functions can be chosen for $\mathcal{L}$. For simplicity, in this work, we present the *energy loss*. The energy loss parameter learning scheme directly minimizes the energy of the training samples, i.e., the per-sample losses are:

$$L_i(\mathbf{y}_i, \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = E((\mathbf{y}_{i,t}, \mathbf{z}_i^*), \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$$

Notice that inference over the latent variables is necessary for gradient and objective value computations. However, a complete prediction from NeuPSL, i.e., inference over all components of $\mathbf{y}$, is unnecessary. Therefore the parameter learning problem is as follows:

$$\arg\min_{\mathbf{w}_{nn}, \mathbf{w}_{psl}} \sum_{i=1}^{P} \min_{\mathbf{z} \in \mathbf{\Omega}} \mathbf{w}_{psl}^T \Phi((\mathbf{y}_{i,t}, \mathbf{z}), \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn})$$

With L2 regularization, the NeuPSL energy function is strongly convex in all components of $\mathbf{y}_i$. Thus, by Danskin (1966), the gradient of the energy loss, $L_i(\cdot)$, with respect to $\mathbf{w}_{psl}$ at $\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{nn,i}\mathbf{w}_{nn}$ is:

$$\nabla_{\mathbf{w}_{psl}} L_i(\mathbf{y}_i, \mathbf{x}_{sy,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = \Phi((\mathbf{y}_{i,t}, \mathbf{z}_i^*), \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn})$$

Then the per-sample energy loss partial derivative with respect to $\mathbf{w}_{nn}[j]$ at $\mathbf{y}_i, \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{psl}$ is:

$$\frac{\partial L_i(\mathbf{y}_i, \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl})}{\partial \mathbf{w}_{nn}[j]} = \sum_{r=1}^{R} \mathbf{w}_{psl}[r] \sum_{q \in \tau_r} \frac{\partial \phi_q((\mathbf{y}_{i,t}, \mathbf{z}_i^*), \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn})}{\partial \mathbf{w}_{nn}[j]}$$

Details on the learning algorithms and accounting for degenerate solutions of the energy loss are included in supplementary materials.

6 Experimental Evaluation

We evaluate NeuPSL's prediction performance and inference time on three tasks to demonstrate the significance of joint symbolic inference and learning. NeuPSL, implemented using the open-source PSL software package, can be integrated with any neural network library (here, we used TensorFlow).[2] Our investigation addresses the following questions: Q1) Can neuro-symbolic methods provide a boost over conventional purely data-driven neural models? Q2) Can we effectively leverage structural relationships across training examples through joint reasoning? Q3) How does NeuPSL compare with other neuro-symbolic methods in terms of time efficiency on large scale problems?

6.1 MNIST Addition

The first set of experiments are conducted on a variation of MNIST Addition, a widely used NeSy evaluation task [Manhaeve *et al.*, 2018]. The task involves determining the sum of two lists of MNIST images. For example, a **MNIST-Add1** addition is ($[\,3\,] + [\,5\,] = 8$), and a **MNIST-Add2** addition is ($[\,0, 4\,] + [\,3, 7\,] = 41$). The challenge stems from the lack of labels for the MNIST images in the addition equation. Only the final sum of the equation is given, leaving the task of identifying the individual digits and determining their values up to the model being used.

While NeuPSL proves to be successful in the original **MNIST-Add** setting (appendix for further details), here we are interested in exploring the power of joint inference and learning capabilities in NeSy systems. We introduce a variant of the **MNIST-Add** task in which digits are reused across multiple addition examples, i.e., we introduce *overlap*. Figure 2 demonstrates the process of introducing overlap and how joint models narrow the space of possible labels when MNIST images are re-used. For instance, in the scenario presented in Figure 2, the same MNIST image of a zero is utilized in two separate additions. To comply with both addition constraints, the potential label space is restricted and can no longer include options such as two or three, as they would violate one of the addition rules. In contrast, a model performing independent reasoning would have no way of enforcing this constraint across examples.

In the overlap variant of **MNIST-Add**, we focus on low-data settings to understand whether NeSy systems' joint reasoning can effectively leverage additional structure to overcome a lack of data. To create overlap, we begin with a set of n unique MNIST images from which we re-sample to create $(n+m)/2$ **MNIST-Add1** and $(n+m)/4$ **MNIST-Add2** additions. We vary the amount of overlap with $m \in \{0, n/2, n\}$ and compare performance with $n \in \{40, 60, 80\}$. Results are

[1] Note arguments of the energy function and symbolic potentials are dropped for simplicity, i.e., $E(\cdot) = E(\mathbf{y}, \mathbf{x}_{sy,i}, \mathbf{x}_{nn,i}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$, $\phi(\cdot) = \phi(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{x}_{nn}, \mathbf{w}_{nn})$, and $\mathbf{g}_{nn}[i](\cdot) = \mathbf{g}_{nn}[i](\mathbf{x}_{nn}, \mathbf{w}_{nn})$.

[2] Implementation details, hyperparameters, network architectures, hardware, and NeuPSL models, are described in the Appendix.
Code and Data: https://github.com/linqs/neupsl-ijcai23
Appendix: https://arxiv.org/abs/2205.14268

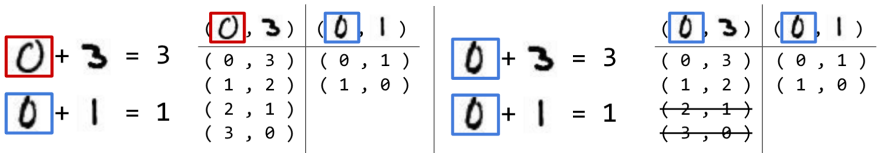


Figure 2: Example of overlapping MNIST images in **MNIST-Add1**. On the left, distinct images are used for each zero. On the right, the same image is used for both zeros.

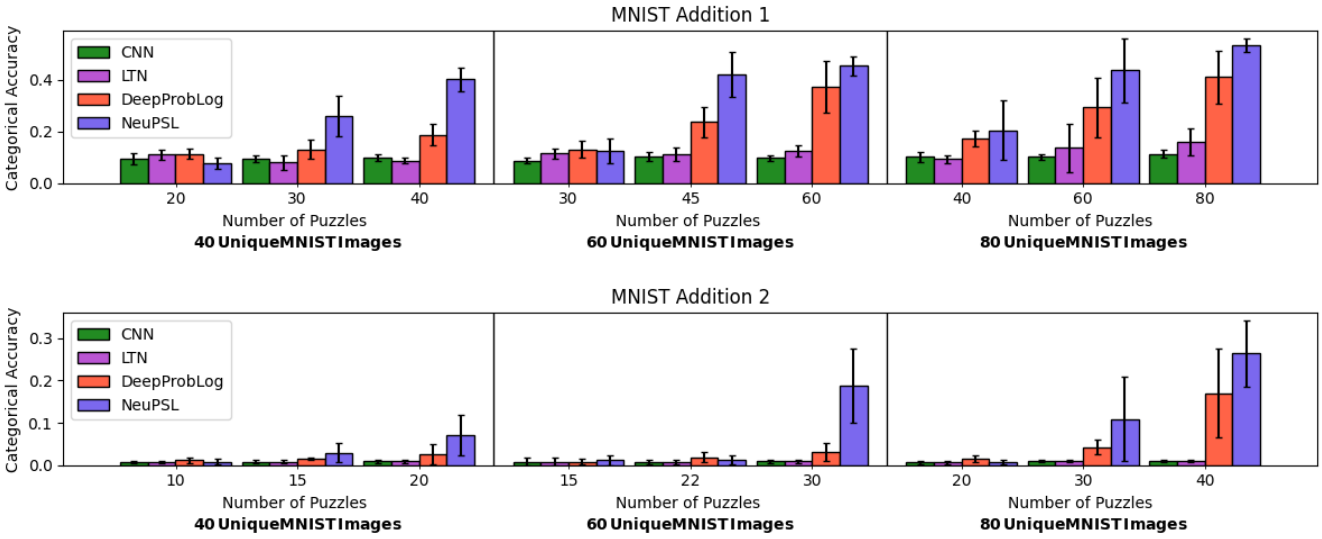


Figure 3: Average test set accuracy and standard deviation on **MNIST-Add** datasets with varying amounts of overlap.

reported over ten test sets of $1,000$ MNIST images with overlap proportional to the respective train set.

Figure 3 summarizes average performance for varying overlap settings. Each panel varies the number of additions for a set number of unique MNIST images. For example, the upper left panel presents the results obtained for MNIST-Add1 with 40 unique images used to generate 20, 30, and 40 additions. Initially, there is not enough structure from the additions with no overlap for symbolic inference to discern the correct digit labels for training the neural models. Then, despite the number of unique MNIST images remaining the same, as the number of additions increases, DPL and NeuPSL improve their prediction performance by leveraging the added joint information (Q2). In all cases, NeuPSL performs best and uses the added structure most efficiently. LTNs and the CNN baseline benefit the least from joint information, a consequence of both learning and inference being performed independently across batches of additions (Q1).

6.2 Visual Sudoku Classification

Inspired by the Visual Sudoku problem proposed by Wang *et al.* (2019), Augustine *et al.* (2022) introduced a novel NeSy task, **Visual-Sudoku-Classification**. In this task, 4x4 Sudoku puzzles are constructed using unlabeled MNIST images. The model must identify whether a puzzle is correct, i.e., no duplicate digits in any row, column, or square. Therefore this task does not require learning the underlying label for images but rather whether an entire puzzle is valid. For instance, $\boxed{3}$ does not need to belong to a "3" class, instead $\boxed{3}$ and $\boxed{4}$ need to be labeled as different symbols. Similar to MNIST-Add we explore an overlap variant in low-data settings, with overlapping MNIST images across puzzles.

We compare NeuPSL with two baselines, CNN-Visual and CNN-Digit. The first, CNN-Visual, takes the pixels for a Sudoku puzzle as input and outputs the probability the puzzle is valid. The second, CNN-Digit, is provided the (unfair) advantage of all sixteen image labels as input. We use this to verify whether a neural model can learn Sudoku rules. Scalably developing LTN and DPL models in this new setting is not straightforward due to the large dimensionality of the output space. A non-expert implementation of a visual sudoku model in DPL and LTN may result in suboptimal reports on model performance and are therefore not included.

Figure 4 shows the accuracy of NeuPSL and CNN models on **Visual-Sudoku-Classification** with varying amounts of overlap. CNN-Visual and CNN-Digit struggle to leverage the problem structure and fail to generalize even the highest data and overlap setting with 256 MNIST images across 64 puzzles. However, NeuPSL achieves 70% accuracy using roughly 64 MNIST images across 16 puzzles, again showing it efficiently leverages joint information across training examples (Q1 and Q2). This is a particularly impressive result as the neural network in the NeuPSL model was trained to be a 93% 4-digit distinguisher without digit labels.

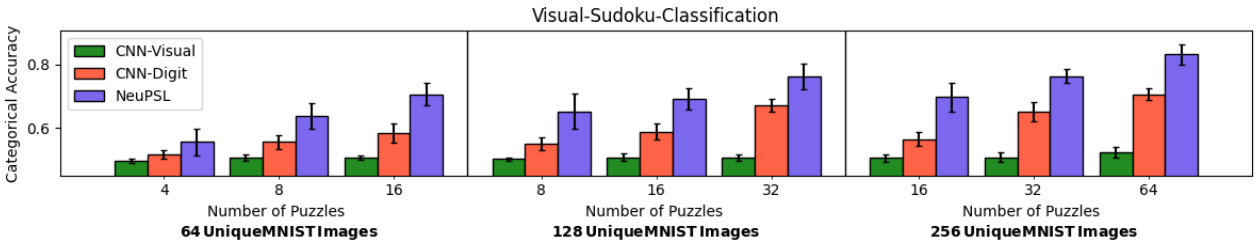


Figure 4: Average test set accuracy and standard deviation on **Visual-Sudoku-Classification** with varying amounts of overlap.

Method	Citeseer		Cora	
	(Accuracy)	(Seconds)	(Accuracy)	(Seconds)
Neural$_{PSL}$	57.76 ± 1.71	-	57.12 ± 2.13	-
LP$_{PSL}$	50.88 ± 1.18	-	73.32 ± 2.39	-
DeepProbLog	timeout	timeout	timeout	timeout
DeepStochLog	61.30 ± 1.44	34.42 ± 0.87	69.96 ± 1.47	165.28 ± 4.49
GCN	67.50 ± 0.57	3.10 ± 0.04	79.52 ± 1.13	1.31 ± 0.01
NeuPSL$_{LP}$	67.34 ± 1.17	3.98 ± 0.05	76.80 ± 2.27	4.00 ± 0.31
NeuPSL$_{LP+FP}$	**68.48 ± 1.22**	4.23 ± 0.05	**81.22 ± 0.79**	4.07 ± 0.14

Table 1: Test set accuracy and inference runtime in seconds on two citation network datasets.

6.3 Citation Network Node Classification

In our final experiment, we evaluate the performance of NeuPSL on two widely studied citation network node classification datasets: Citeseer and Cora [Sen *et al.*, 2008]. In these datasets, symbolic models have the potential to improve predictions by leveraging the homophilic structure of the citation network, i.e., two papers connected in the network are more likely to have the same label. This setting differs from **Visual-Sudoku-Classification** and **MNIST-Add** as the symbolic relations are not always true. Moreover, the symbolic relations can be defined over a general and potentially large number of nodes in the network, i.e., a node can be connected to any number of neighbors.

We propose two NeuPSL models for citation network node classification. Both models integrate a neural network that uses a paper's features to provide an initial classification, which is then adjusted via symbolic reasoning. The first model, NeuPSL$_{LP}$ (Label Propagation), directly uses the bag-of-words feature vector, while the second model, NeuPSL$_{LP+FP}$ (Label + Feature Propagation), first performs the feature construction procedure as described in Wu *et al.* (2019) to obtain a richer representation to provide to the neural model. We examine the runtime and model performance of NeSy methods NeuPSL$_{LP}$, NeuPSL$_{LP+FP}$, DPL and its scalable extension, DeepStochLog [Winters *et al.*, 2022], and a Graph Convolutional Network (GCN) [Kipf and Welling, 2017]. Additionally, we include the performance of two baselines, LP$_{PSL}$ and Neural$_{PSL}$. These baselines represent the distinct symbolic and neural components used in the NeuPSL$_{LP}$ model but perform only neural or symbolic reasoning, not both. We averaged the results over ten randomly sampled splits using 5% of the nodes for training, 5% of the nodes for validation, and 1000 nodes for testing.

Table 1 shows DeepStochLog, GCN, and NeuPSL all outperform the independent baselines (Q1), with NeuPSL$_{LP+FP}$ performing the best. These results demonstrate the power of using NeSy systems to effectively leverage structure to improve prediction performance. Additionally, NeuPSL is capable of scaling its joint inference process to larger structures, achieving higher accuracy with an 8 and 40 times speed up over DeepStochLog in Citeseer and Cora, respectively (Q3). Surprisingly, NeuPSL also achieves a higher prediction performance than even a GCN model while using significantly fewer trainable parameters.

7 Conclusion

In this paper, we introduced NeuPSL, a novel NeSy framework that integrates neural architectures and a tractable class of graphical models for jointly reasoning over symbolic relations and showed its utility across a range of neuro-symbolic tasks. There are many avenues for future work, including exploring different learning objectives, such as ones that balance traditional neural and energy-based losses and new application domains. Each of these is likely to provide new challenges and insights.

Acknowledgments

This work was partially supported by the National Science Foundation grant CCF-2023495 and a Google Faculty Research Award.

Contribution Statement

Connor Pryor and Charles Dickens contributed equally to this work.

References

Duhai Alshukaili, Alvarao Fernandees, and Norman Paton. Structuring linked data search results using probabilistic soft logic. In *ISWC*, 2016.

Eriq Augustine, Connor Pryor, Charles Dickens, Jay Pujara, William Yang Wang, and Lise Getoor. Visual sudoku puzzle classification: A suite of collective neuro-symbolic tasks. In *International Workshop on Neural-Symbolic Learning and Reasoning (NeSy)*, 2022.

Stephen Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. Hinge-loss Markov random fields and probabilistic soft logic. *JMLR*, 18(1):1–67, 2017.

Sebastian Bader and Pascal Hitzler. Dimensions of neural-symbolic integration - A structured survey. *arXiv preprint cs/0511042*, 2005.

Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini, and Michael Spranger. Logic tensor networks. *AI*, 303(4):103649, 2022.

Islam Beltagy, Katrin Erk, and Raymond Mooney. Probabilistic soft logic for semantic textual similarity. In *ACL*, 2014.

Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Daniel Lowd, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*, 2017.

Matko Bošnjak, Tim Rocktäschel, Jason Naradowsky, and Sebastian Riedel. Programming with a differentiable forth interpreter. In *ICML*, 2017.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2010.

William W. Cohen, Fan Yang, and Kathryn Mazaitis. Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *JAIR*, 67:285–325, 2020.

John Danskin. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966.

Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications*. Springer, 2002.

Artur S. d'Avila Garcez, Luís C. Lamb, and Dov M. Gabbay. *Neural-Symbolic Cognitive Reasoning*. Springer, 2009.

Artur d'Avila Garcez, Marco Gori, Luís C. Lamb, Luciano Serafini, Michael Spranger, and Son N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *Journal of Applied Logics*, 6(4):611–632, 2019.

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, 2007.

Luc De Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. In *IJCAI*, 2020.

Lingjia Deng and Janyce Wiebe. Joint prediction for Entity/Event-LEvel sentiment analysis using probabilistic soft logic models. In *EMNLP*, 2015.

Charles Dickens, Rishika Singh, and Lise Getoor. Hyperfair: A soft approach to integrating fairness criteria. In *FAcc-TRec*, 2020.

Charles Dickens, Connor Pryor, Eriq Augustine, Alon Albalak, and Lise Getoor. Efficient learning losses for deep hinge-loss markov random fields. In *Workshop on Tractable Probabilistic Modeling (TPM)*, Eindhoven, Netherlands, 2022.

Michelangelo Diligenti, Soumali Roychowdhury, and Marco Gori. Integrating prior knowledge into deep learning. In *ICMLA*, 2017.

Ivan Donadello, Luciano Serafini, and Artur S. d'Avila Garcez. Logic tensor networks for semantic image interpretation. In *IJCAI*, 2017.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *JAIR*, 61:1–64, 2018.

Golnoosh Farnadi, Behrouz Babaki, and Lise Getoor. A declarative approach to fairness in relational domains. *IEEE Data Engineering Bulletin*, 42(3):36–48, 2019.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *ACL*, 2016.

Angelika Kimmig, Alex Memory, Renée J. Miller, and Lise Getoor. A collective, probabilistic approach to schema mapping using diverse noisy evidence. *TKDE*, 31(8):1426–1439, 2019.

Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

George J. Klir and Bo Yuan. *Fuzzy Sets and Fuzzy Logic - Theory and Applications*. Prentice Hall, 1995.

Pigi Kouki, Shobeir Fakhraei, James R. Foulds, Magdalini Eirinaki, and Lise Getoor. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *RecSys*, 2015.

Luís C. Lamb, Artur d'Avila Garcez, Marco Gori, Marcelo O. R. Prates, Pedro H. C. Avelar, and Moshe Y. Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. In *IJCAI*, 2020.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. A tutorial on energy-based learning. *Predicting Structured Data*, 1(0), 2006.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *ACL*, 2017.

Shulin Liu, Kang Liu, Shizhu He, and Jun Zhao. A probabilistic soft logic based approach to exploiting latent and global information in event classification. In *AAAI*, 2016.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. In *NeurIPS*, 2018.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *AI*, 298:103504, 2021.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *ICLR*, 2019.

Sanket Vaibhav Mehta, Jay Yoon Lee, and Jaime Carbonell. Towards semi-supervised learning for deep semantic role labeling. In *EMNLP*, 2018.

Bogdan Moldovan, Ingo Thon, Jesse Davis, and Luc de Raedt. Mcmc estimation of conditional probabilities in probabilistic programming languages. In *ECSQARU*, 2015.

Yatin Nandwani, Abhishek Pathak, and Parag Singla. A primal dual formulation for deep learning with constraints. In *NeurIPS*, 2019.

Jay Pujara, Hui Miao, Lise Getoor, and William W. Cohen. Knowledge graph identification. In *ISWC*, 2013.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NeurIPS*, 2017.

Marco Rospocher. An ontology-driven probabilistic soft logic approach to improve nlp entity annotation. In *ISWC*, 2018.

Mrinmaya Sachan, Kumar Avinava Dubey, Tom M Mitchell, Dan Roth, and Eric P Xing. Learning pipelines with limited data and domain knowledge: A study in parsing physics problems. In *NeurIPS*, 2018.

Prithviraj Sen, Galileo Mark Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

Luciano Serafini and Artur S. d'Avila Garcez. Learning and reasoning with logic tensor networks. In *AI*IA*, 2016.

Shai Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning (FTML)*, 4(2):107–194, 2012.

Karan Sikka, Andrew Silberfarb, John Byrnes, Indranil Sur, Ed Chow, Ajay Divakaran, and Richard Rohwer. Deep adaptive semantic logic (dasl): Compiling declarative knowledge into deep neural networks. Technical report, SRI International, 2020.

Dhanya Sridhar, Jay Pujara, and Lise Getoor. Scalable probabilistic causal structure discovery. In *IJCAI*, 2018.

Sriram Srinivasan, Charles Dickens, Eriq Augustine, Golnoosh Farnadi, and Lise Getoor. A taxonomy of weight learning methods for statistical relational learning. *Machine Learning*, 2021.

Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, 2019.

Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. DeepStochLog: Neural stochastic logic programming. In *AAAI*, 2022.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *ICML*, 2018.

Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*, 2017.

Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In *IJCAI*, 2020.

A Appendix

The appendix includes the following sections: Limitations, Formulating Existing NeSy Frameworks as NeSy-EBMs, Joint Reasoning in NeSy-EBMs, NeuPSL Parameter Learning, Dataset Details, NeuPSL Models, Baseline Models, Extended Evaluation Details, and Computational Hardware Details.

B Limitations

Practitioners applying NeuPSL should consider the following three limitations. First, NeuPSL operates on real-valued logic, which improves scalability but is a relaxation of the original problem. This relaxation may overlook nuances (e.g., integer constraints) of the original task. Second, while NeuPSL demonstrates excellent performance in solving joint symbolic inference tasks, it comes at the expense of a higher inference runtime than a purely neural model. The computational demands of NeuPSL may limit its applicability in scenarios where real-time processing is necessary. Lastly, NeuPSL is trained in this work with the energy learning loss. Using this loss reduces the energy of the truth data but does not necessarily align with a downstream evaluation metric, and we have identified some degenerate solutions (Appendix E.1). Exploring the adaptation of NeuPSL to support different learning losses is an interesting avenue for future research.

C Formulating Existing NeSy Frameworks as NeSy-EBMs

This section shows how to formulate two popular NeSy frameworks, DeepProbLog (DPL) [Manhaeve *et al.*, 2018] and LTNs (LTNs) [Badreddine *et al.*, 2022], as NeSy-EBMs.

C.1 DeepProbLog

DeepProbLog (DPL) extends the probabilistic programming language ProbLog [De Raedt *et al.*, 2007]. A ProbLog program consists of (i) a set of probabilistic facts $\mathcal{F}$ of the form $p :: f$ where p is a probability and f is a $\{0, 1\}$ valued symbolic variable and (ii) a set $\mathcal{R}$ of symbolic statements or rules. The following ProbLog program is a common example that models the likelihood of a burglary or an earthquake, given an alarm was sounded and is also presented in Manhaeve *et al.* [2021]

Probabilistic Facts
0.1 :: burglary. 0.2 :: earthquake.
0.5 :: hearsAlarm(mary). 0.4 :: hearsAlarm(john).
Rules
alarm : − earthquake.
alarm : − burglary.
calls(X) : − alarm, hearsAlarm(X).

A subset of the probabilistic facts $F \subseteq \mathcal{F}$ defines a possible instantiation, or world:

$$t_F := F \cup \{f \,|\, \mathcal{R} \cup F \vDash f\}.$$

For the example, $t_{\{\text{burglary, hearsAlarm(mary)}\}} = \{\text{burglary, hearsAlarm(mary), alarm, calls(mary)}\}$. Then, the probability of a world, $P(t_F)$, is the product of the probabilities of the probabilistic facts in the world:

$$P(t_F) := \Pi_{\mathbf{f}[i] \in F} \mathbf{p}[i] \Pi_{\mathbf{f}[i] \in \mathcal{F} \setminus F} (1 - \mathbf{p}[i]).$$

For the running example, $P(t_{\{\text{burglary, hearsAlarm(mary)}\}}) = 0.1 \cdot 0.5 \cdot (1 - 0.2) \cdot (1 - 0.4)$. Finally, the probability of a query atom, q, is defined as the sum of the probabilities of the worlds containing q:

$$P(q) := \sum_{F \in \mathcal{F} \,|\, q \in t_F} P(t_F).$$

ProbLog inference, specifically as it is applied in the deep extension proposed by Manhaeve *et al.* (2018), is a marginal inference problem. Specifically, the inference task is computing the marginal probability of a single query atom as shown above. This is equivalent to finding the weighted model count (WMC) of the worlds where the query atom is true. Thus, the exact marginal inference problem in ProbLog is #P-complete, i.e., it is at least NP-hard. This means that computing the exact probability of a query in a ProbLog program is a computationally challenging problem that requires exponential time in the worst case. Therefore, exact marginal inference in ProbLog is generally only feasible for small or moderately sized problems. For larger problems with more variables, approximate inference techniques are used to obtain approximate probabilities more efficiently De Raedt *et al.* [2007]; Moldovan *et al.* [2015].

DPL introduces syntax and semantics to ProbLog to support specifying probabilities of events with neural networks Manhaeve *et al.* [2018, 2021]. Specifically, a set of neural annotated disjunctions (nADs) are specified by a user and take the form:

$$\begin{aligned} &\text{nn}(id, \mathbf{v}, u_1) :: \text{h}(\mathbf{v}, u_1)\,; \\ &\quad \cdots ; \text{nn}(id, \mathbf{v}, u_n) :: \text{h}(\mathbf{v}, u_n)\,; \vDash b_1, \cdots, b_m, \end{aligned}$$

where the b_i are atoms, $\mathbf{v}$ is a vector of features that the neural component, identified by id, has access to. Moreover, the output of the neural component, $\text{nn}(id, \mathbf{v}, u_i)$, is interpreted as the probability that the atom h_i is true and the sum of the outputs of the neural model must sum to 1: $\sum_{i=1}^{n} \text{nn}(id, \mathbf{v}, u_1) = 1$. The interpretation of an annotated disjunction is that whenever all of the atoms $b_1, \cdots, b_m$ are true, then each h_i will be true with probability $\text{nn}(id, \mathbf{v}, u_i)$.

Inference in DPL is exactly the same as ProbLog marginal inference with a single query atom, except a forward pass is made with the neural network to compute the probabilities of the nADs. Learning the parameters of the DPL model is the task of finding the setting of the trainable parameters, denoted by $\mathbf{x}$, that minimizes a sum of losses, $L()$. Each loss measures the distance between a vector of n desired probabilities $\mathbf{p}_{true}$ and $[P(q_1), \cdots, P(q_n)]$, the marginal inference values predicted by DPL:

$$\arg\min_{\mathbf{x}} \frac{1}{n} \sum_{i=1}^{n} L(P(q_i), \mathbf{p}_{true}[i]).$$

Though instantiating the marginal probability function is non-trivial and computationally expensive, marginal inference ultimately reduces to a series of differentiable algebraic operations and is therefore differentiable. DPL uses stochastic gradient descent to find parameters minimizing the training objective.

DPL is a NeSy-EBM. The fact probabilities, p, are partitioned into the observed NeSy-EBM symbolic variables $\mathbf{x}_{sy}$, the vector of symbolic parameters, $\mathbf{w}_{sy}$, and neural network outputs, $\mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$. Without loss of generality, suppose

$$\mathbf{p} = \begin{bmatrix} \mathbf{x}_{sy} \\ \mathbf{w}_{sy} \\ \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) \end{bmatrix}.$$

The query atoms, i.e., the atoms present in the DPL model that are not specified in the set of probabilistic facts, correspond to the symbolic variables $\mathbf{y}$.

The definition of the DPL symbolic potentials and energy function are tied to the inference task; a different definition of the symbolic potential and energy function is used to implement marginal versus MAP inference. As previously mentioned, DPL predictions are most commonly obtained by performing marginal inference for a single query atom. Moreover, a consequence of the DPL semantics is that the marginal inference problem reduces to an analytical expression composed of only product and sum operations. Thus, from the NeSy EBM perspective, to implement marginal inference DPL interprets a program with a set of probabilistic facts and data to define a symbolic potential for every marginal probability and then the energy function is simply the sum of the symbolic potentials. On the other hand, for MAP inference, DPL creates a symbolic potential for every possible world and the energy function is equivalent to the negative of the joint probability distribution implied by the DPL program. We will only formally cover the marginal inference case.

The probability of a world t_F, defined by the subset of probabilistic facts $F \in \mathcal{F}$ is a function of the DPL fact probabilities, $\mathbf{p}$, and hence is a function of $\mathbf{x}_{sy}$, $\mathbf{w}_{sy}$, and $\mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$:

$$\begin{aligned}
&P_{\mathbf{t}_F}(\mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\
&\quad := \left(\Pi_{\mathbf{x}_{sy}[j] \in F} \mathbf{x}_{sy}[j]\right) \cdot \left(\Pi_{\mathbf{x}_{sy}[j] \in \mathcal{F} \setminus F} (1 - \mathbf{x}_{sy}[j])\right) \\
&\qquad \cdot \left(\Pi_{\mathbf{w}_{sy}[j] \in F} \mathbf{w}_{sy}[j]\right) \cdot \left(\Pi_{\mathbf{w}_{sy}[j] \in \mathcal{F} \setminus F} (1 - \mathbf{w}_{sy}[j])\right) \\
&\qquad \cdot \left(\Pi_{\mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})[j] \in F} \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})[j]\right) \\
&\qquad \cdot \left(\Pi_{\mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})[j] \in \mathcal{F} \setminus F} (1 - \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})[j])\right).
\end{aligned}$$

Then, as in ProbLog, the marginal probability of a query atom is a function of the probabilities of the worlds. For the world t_F, defined by the subset of probabilistic facts $F \subseteq \mathcal{F}$, let $\chi_{t_F}[\cdot]$ be the indicator function identifying if a setting of the variables $\mathbf{y}$ matches the world t_F:

$$\chi_{t_F}[\hat{\mathbf{y}}] := \begin{cases} 1 & ((\hat{\mathbf{y}}[i] = 1) \implies \mathbf{y}[i] \in t_F) \, \forall i \in \{1, \cdots, n_{\mathbf{y}}\} \\ 0 & \text{o.w.} \end{cases}.$$

With $\chi_{t_F}[\mathbf{y}]$, it is also possible to write the marginal probability of a variable as function of $\mathbf{x}_{sy}$, $\mathbf{w}_{sy}$, and $\mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})$:

$$\begin{aligned}
&P_{\mathbf{y}[i]}(\mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\
&:= \sum_{\hat{\mathbf{y}} \in \{0,1\}^{n_{\mathbf{y}}}} \hat{\mathbf{y}}[i] \left(\sum_{F \in \mathcal{P}(\mathcal{F})} \chi_{t_F}[\hat{\mathbf{y}}] P_{\mathbf{t}_F}(\mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \right).
\end{aligned}$$

Let $d : [0,1] \times [0,1] \to \mathbb{R}$ be a metric quantifying the distance between its two arguments. For each variable $\mathbf{y}[i]$ for $i \in \{1, \cdots, n_{\mathbf{y}}\}$ define a symbolic potential:

$$\begin{aligned}
&\psi_{DPL,i}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \\
&\quad := d(\mathbf{y}[i], P_{\mathbf{y}[i]}(\mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))).
\end{aligned}$$

Let $\Psi_{DPL}(\cdot) := \left[\psi_{DPL,t_{F_i}}(\cdot)\right]_{i=1}^{n_{\mathbf{y}}}$ be the vector of all $n_{\mathbf{y}}$ symbolic potentials. The energy function to produce marginal inference DPL predictions is then the summation of all the symbolic potentials:

$$\begin{aligned}
&E_{DPL}(\Psi_{DPL}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) \\
&:= \sum_{i=1}^{n_{\mathbf{y}}} \Psi_{DPL}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i].
\end{aligned}$$

Clearly, the optimal value of the energy function is 0 and is achieved at the unique setting of the variables matching their corresponding marginal probability. Thus inference is equivalent to evaluating the marginal probabilities for each variable.

C.2 Logic Tensor Networks

Logic Tensor Networks (LTNs) forwards deep neural network predictions into functions representing symbolic relations with real-valued or fuzzy logic semantics Badreddine *et al.* [2022]. The fuzzy logic functions are combined using a *formula aggregator* to define a satisfaction level. Badreddine *et al.* (2022) suggest using the product real logical semantics to translate logical statements, i.e., given two truth values a and b in $[0, 1]$:

$$\begin{aligned}
\neg(a) &:= 1 - a \\
\wedge(a, b) &:= a \cdot b \\
\vee(a, b) &:= a + b - a \cdot b \\
\implies (a, b) &:= a + b - a \cdot b
\end{aligned}$$

Additionally, generalized mean semantics for existential and universal quantifiers are used for collections of truth values $\mathbf{a} = [a]_{i=1}^{n}$:

$$\begin{aligned}
\exists(\mathbf{a}) &:= \left(\frac{1}{n} \sum_{i=1}^{n} a_i^p\right)^{\frac{1}{p}} \\
\forall(\mathbf{a}) &:= 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - a_i)^p\right)^{\frac{1}{p}},
\end{aligned}$$

where $p \in \mathbb{R}_+$ is a hyperparameter. For example, consider the logical statement

$$\exists v \in \mathcal{V} \left(P(u, v) \wedge Q(v)\right).$$

LTNs instantiate predicate arguments with features. Let $\mathbf{X}_{\mathcal{U}}$ and $\mathbf{X}_{\mathcal{V}}$ be collections of variable feature vectors such that

$\mathbf{X}[u]$ and $\mathbf{X}[v]$ are the feature vectors corresponding to the entities u and v, respectively. Furthermore, the predicate values are either provided by a deep neural network output or are values representing observations or a potential prediction. For instance, the predicate $P(u, v)$ in the example can be instantiated as the output of a deep neural network parameterized by its weights $\mathbf{w}_{nn}$ and represented by the function: $nn(\mathbf{X}[u], \mathbf{X}[v]; \mathbf{w})$ which takes the two feature vectors, corresponding to the arguments u and v, respectively, to a value in $[0, 1]$. Then, $Q(v)$ could be a constant from $[0, 1]$. Let $\mathbf{x}_Q$ be a vector of scalars from $[0, 1]$ such that $\mathbf{x}_Q[v]$ represents the predicate value for $Q(v)$. Then, the logical statement in the example is a composition of the specified real-logic operators and quantifiers. For a provided instance of the argument u the real-valued logic function for the example is:

$$h_u(\mathbf{X}_\mathcal{U}, \mathbf{X}_\mathcal{V}, \mathbf{x}_\mathcal{Q}; \mathbf{w}) := \left(\frac{1}{\|\mathcal{V}\|} \sum_{v \in \mathcal{V}} \left(nn(\mathbf{X}[u], \mathbf{X}[v]; \mathbf{w}) \cdot \mathbf{x}_Q[v] \right)^p \right)^{\frac{1}{p}}.$$

Using the generalized mean semantics for the universal quantifier as the formula aggregator, the satisfaction level of the LTNs model prediction is:

$$G(\mathbf{w}) := 1 - \left(\frac{1}{\|\mathcal{U}\|} \sum_{u \in \mathcal{U}} \left(1 - h_u(\mathbf{X}_\mathcal{U}, \mathbf{X}_\mathcal{V}, \mathbf{x}_\mathcal{Q}; \mathbf{w}) \right)^p \right)^{\frac{1}{p}}.$$

There are many ways to instantiate an LTN depending on the modeler's choice of real-logic semantics, the formula aggregator, and the logical relations. The example above illustrates a common setting of the real-logic semantics and the formula aggregator for a specific composition of logical formula.

The parameters of the LTNs are the deep neural network weights. Learning is the task of finding a setting of the weights which maximize the satisfaction of an aggregated set of logical formula instantiated with observations and features:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} G(\mathbf{w}).$$

In other words, learning in LTNs can be understood as optimizing under first-order logic constraints relaxed into a loss function. There are a variety of real-valued logical semantics and formula aggregators that result in the satisfaction level function $G(\mathbf{w})$ being differentiable with respect to the weights. Given a trained set of parameters obtained by learning, $\mathbf{w}^*$, inference is presented as querying the truth value of an instantiated predicate or logical formula. A prediction in LTNs in a multi-class or joint output setting such is obtained by evaluating the truth-value of all possible outputs and returning the highest valued configuration, i.e., the state with maximum satisfaction.

Through the lens of NeSy-EBMs, the system's fuzzy logic semantics define the symbolic potentials and the formula aggregator is the energy function. More formally, the NeSy-EBM unobserved and observed symbolic variables and neural network outputs partition the instantiated predicates of the real-valued logic functions h_i. Each of the m real-valued logic functions can be written as a function of only the symbolic variables and the neural network outputs: $h_i(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$. The functions h_i are the symbolic potentials of the NeSy-EBM:

$$\psi_{LTN,i}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}) := h_i(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}).$$

Let $\Psi_{LTNs}(\cdot) := [\psi_{LTNs,i}(\cdot)]_{i=1}^m$ be the vector of all m symbolic potentials. Then, the formula aggregator defines the energy function. Using the generalized mean semantics for the universal quantifier, the NeSy-EBM energy function for LTNs is:

$$E_{LTN}(\Psi_{LTNs}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))) := \left(\frac{1}{m} \sum_{i=1}^{m} \left(1 - \Psi_{LTNs}(\mathbf{y}, \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))[i] \right)^p \right)^{\frac{1}{p}}.$$

The LTNs framework is general and the scalability and expressivity of the system are dependent on the modeler's choice of the domain of the unobserved variables: $\mathcal{Y}$, the real-valued logical semantics, and the formula aggregator. Furthermore, notice there is no explicit use of the symbolic parameters $\mathbf{w}_{sy}$ as the LTNs framework uses standard real-valued logics that typically do not have trainable parameters.

LTNs learning is finding the parameters with the highest satisfaction, i.e., learning with the energy loss in the NeSy-EBM framework. The NeSy-EBM framework connects LTNs to the EBM literature, which suggests principled alternative learning algorithms. Moreover, the NeSy-EBM framework sheds light on design choices for the various components of the LTNs to ensure the applicability of first-order methods for learning and desirable scalability and expressiveness properties of inference.

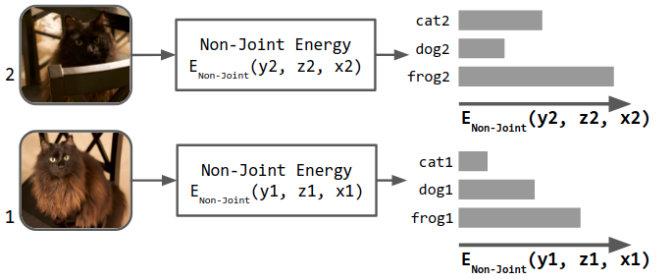
D Joint Reasoning in NeSy-EBMs

This section expands the discussion of joint reasoning in NeSy-EBMs. To reiterate, we highlight two important categories of NeSy-EBM energy functions: *joint* and *independent*. Formally, an energy function that is additively separable over the output variables $\mathbf{y}$ is an *independent energy function*, i.e., corresponding to each of the n_y components of the output variable $\mathbf{y}$ there exists functions n_y functions $E_1(\mathbf{y}[1], \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$, $\cdots$, $E_{n_y}(\mathbf{y}[n_y], \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ such that

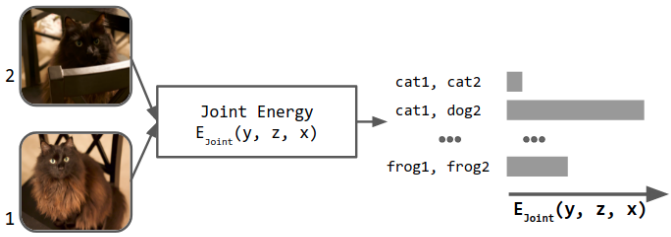
$$E(\cdot) = \sum_{i=1}^{n_y} E_i(\mathbf{y}[i], \mathbf{x}_{sy}, \mathbf{w}_{sy}, \mathbf{g}(\mathbf{x}_{nn}, \mathbf{w}_{nn})).$$

While a function that is not separable over output variables $\mathbf{y}$ is a *joint energy function*. This categorization allows for an important distinction during inference and learning. Independent energy functions simplify inference and learning as finding an energy minimizer, $\mathbf{y}^*$, can be distributed across the independent functions E_i. In other words, the predicted value for a variable $\mathbf{y}[i]$ has no influence over that of $\mathbf{y}[j]$ where $j \neq i$ and can therefore be predicted separately, i.e., independently. However, independent energy functions cannot leverage some joint information that may be used to improve predictions.

To illustrate, recall the example described in the *Neural Probabilistic Soft Logic* section where a neural network is used to classify the species of an animal in an image with external information. Figure 5 outlines the distinction between independent and joint prediction for this scenario. In Figure 5(a), the independent setting, the input is a single image, and the energy function is defined over the three possible



(a) Independent Energy



(b) Joint-Energy

Figure 5: Example of non-joint and joint energy functions.

classes: `dog`, `cat`, and `frog`. While in Figure 5(b), the joint setting, the input is a pair of images, and the energy function is defined for every possible combination of labels (e.g., (`dog`, `dog`), (`dog`, `cat`), etc.). The joint energy function of (b) leverages external information suggesting the images are of the same entity. Joint reasoning enables a model to make structured predictions that resolve contradictions an independent model could not detect.

For NeSy-EBMs, a joint energy function encodes dependencies between its output variables through its symbolic potentials. NeuPSL additionally benefits from scalable convex inference to speed up learning over a dependent set of output variables. As we see in the *Experimental Evaluation* section, utilizing joint inference and learning in NeSy-EBMs not only provides a boost in performance but produces results that non-joint methods cannot (even with five times the amount of data).

E NeuPSL Parameter Learning

This section details the NeuPSL parameter learning algorithm. We begin by discussing degenerate solutions to the energy loss problem and techniques for overcoming them. We then provide the precise parameter updates we use to efficiently fit NeuPSL model parameters while avoiding the discussed degenerate solutions.

E.1 Energy Loss Degenerate Solutions

In this section, we show two degenerate solutions of the energy loss learning problem for NeuPSL and methods for overcoming them. Recall that the NeuPSL energy loss learning problem is:

$$\begin{aligned}
&\arg\min_{(\mathbf{w}_{nn},\mathbf{w}_{psl})\in\mathbb{R}^{nn}\times\mathbb{R}^r_+} \mathcal{L}(\mathbf{w}_{nn},\mathbf{w}_{psl},\mathcal{S}) \\
&= \arg\min_{(\mathbf{w}_{nn},\mathbf{w}_{sy})\in\mathbb{R}^{nn}\times\mathcal{W}_{sy}} \sum_{i=1}^{P} E((\mathbf{y}_{i,t},\mathbf{z}^*_i),\mathbf{x}_{sy,i},\mathbf{x}_{nn,i},\mathbf{w}_{nn},\mathbf{w}_{sy}) \\
&= \arg\min_{(\mathbf{w}_{nn},\mathbf{w}_{psl})\in\mathbb{R}^{nn}\times\mathbb{R}^r_+} \\
&\sum_{i=1}^{P} \min_{\mathbf{z}|((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x})\in\mathbf{\Omega}} \mathbf{w}_{psl}^T\mathbf{\Phi}((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x}_{sy,i},\mathbf{x}_{nn,i},\mathbf{w}_{nn})
\end{aligned}$$

Note that the symbolic parameters are constrained to be non-negative real numbers. Furthermore, as every symbolic potential has the form:

$$\phi_i(\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{nn}) = \max(l_i(\mathbf{y},\mathbf{x}_{sy},\mathbf{g}_{nn}(\mathbf{x}_{nn},\mathbf{w}_{nn})),0)^\alpha$$

we have that $\phi_i(\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{nn}) \geq 0$ for all settings of the variables $\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{nn}$. Thus, $\Phi_i(\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{nn}) := \sum_{j\in t_i}\phi_i(\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{nn}) \geq 0$ and $\mathbf{\Phi}(\mathbf{y},\mathbf{x_{sy}},\mathbf{x_{nn}},\mathbf{w_{nn}}) := [\Phi_i(\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{nn})]_{i=1}^r \succeq \mathbf{0}$. Therefore, we have that

$$\begin{aligned}
&\mathcal{L}(\mathbf{w}_{nn},\mathbf{w}_{psl},\mathcal{S}) = \\
&\sum_{i=1}^{P} \min_{\mathbf{z}|((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x}_{sy})\in\mathbf{\Omega}} \mathbf{w}_{psl}^T\mathbf{\Phi}((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x}_{sy,i},\mathbf{x}_{nn,i},\mathbf{w}_{nn}) \geq 0
\end{aligned}$$

In fact, $\mathcal{L}(\mathbf{w}_{nn},\mathbf{w}_{psl},\mathcal{S}) = 0$ when $\mathbf{w}_{psl} = \mathbf{0}$. The $\mathbf{0}$ solution to the weight learning problem is degenerate and should be avoided. Precisely, $\mathbf{w}_{psl} = \mathbf{0}$ results in a *collapsed energy function*: a function that assigns all points $\mathbf{y} \in \mathcal{Y}$ to the same energy. Collapsed energy functions have no predictive power since inference, i.e., finding a lowest energy state of the variables is trivial and uninformative. To overcome this degenerate solution a simplex constraint on the symbolic parameters, $\mathbf{w}_{psl} \in \Delta^r := \{\mathbf{w} \in \mathbb{R}^r_+ \big| \|\mathbf{w}\|_1 = 1\}$, is added, making the degenerate solution $\mathbf{w}_{psl} = \mathbf{0}$ infeasible. This constraint also ensures the non-negativity of the parameters and does not inhibit the expressivity of NeuPSL when the deep HL-MRF is exclusively used to obtain MAP inference predictions. This property of (deep) HL-MRFs was shown by Srinivasan *et al.* (2021), where they proved and leveraged the fact that MAP inference in HL-MRFs is invariant to the scale of the weights. Formally, for all weight configurations $\mathbf{w}_{psl}$ and scalars $\tilde{c} \in \mathbb{R}_+$,

$$\begin{aligned}
&\arg\max_{\mathbf{y}|(\mathbf{y},\mathbf{x}_{sy})\in\mathbf{\Omega}} E(\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{nn},\mathbf{w}_{psl}) \\
&\qquad = \arg\max_{\mathbf{y}|(\mathbf{y},\mathbf{x}_{sy})\in\mathbf{\Omega}} E(\mathbf{y},\mathbf{x}_{sy},\mathbf{x}_{nn},\mathbf{w}_{nn},\tilde{c}\cdot\mathbf{w}_{psl})
\end{aligned}$$

The $\mathbf{w}_{psl} = \mathbf{0}$ is infeasible with the simplex constraint; however, an additional degenerate solution arises from its introduction. This is because the energy loss is concave in the symbolic parameters $\mathbf{w}_{psl}$ for fixed $\mathbf{w}_{nn}$ and $\mathcal{S}$, as is shown in following lemma and its corresponding proof. Consequently, a solution to the constrained energy loss learning problem must exist at corner points of the simplex.

Lemma 1. *The energy loss function*

$$\mathcal{L}(\mathbf{w}_{nn},\mathbf{w}_{psl},\mathcal{S}) = \sum_{i=1}^{P} \min_{\mathbf{z}|((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x})\in\mathbf{\Omega}} \mathbf{w}_{psl}^T \mathbf{\Phi}((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x}_{sy,i},\mathbf{x}_{nn,i},\mathbf{w}_{nn})$$

is concave in $\mathbf{w}_{psl}$.

Proof. For all i

$$E((\mathbf{y}_{i,t},\mathbf{z}_i^*),\mathbf{x}_{sy,i},\mathbf{x}_{nn,i},\mathbf{w}_{nn},\mathbf{w}_{psl}) = \inf_{\mathbf{z}|((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x}_{sy})\in\mathbf{\Omega}} \mathbf{w}_{psl}^T \Phi((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x}_{sy,i},\mathbf{x}_{nn,i},\mathbf{w}_{nn})$$

is a pointwise infimum of a set of affine, hence concave, functions of $\mathbf{w}_{psl}$ and is therefore concave [Boyd and Vandenberghe, 2004]. Therefore,

$$\mathcal{L}(\mathbf{w}_{nn},\mathbf{w}_{psl},\mathcal{S}) = \sum_{i=1}^{P} E((\mathbf{y}_{i,t},\mathbf{z}_i^*),\mathbf{x}_{sy,i},\mathbf{x}_{nn,i},\mathbf{w}_{nn},\mathbf{w}_{psl}) \tag{7}$$

is a sum of concave functions of $\mathbf{w}_{psl}$ and is concave. □

Additionally, note that the unit simplex, Δ^r, is a convex set, and, more precisely, a polyhedron. Following from its definition, a concave function is minimized over a polyhedron at one of the vertices. This solution is undesirable for the energy minimization problem because each symbolic relation corresponding to the parameters should have an influence over the model predictions. For this reason, we propose using a negative logarithm as a parameter regularizer, giving the simplex corner solutions infinitely high energy. With negative log regularization and simplex constraints, energy loss symbolic parameter learning is:

$$\min_{\mathbf{w}_{nn}\in\mathcal{W}_{nn},\mathbf{w}_{psl}\in\Delta^r} \mathcal{L}(\mathbf{w}_{nn},\mathbf{w}_{psl},\mathcal{S}) - \sum_{i=1}^{r} \log_b(\mathbf{w}_{psl}[i]) \tag{8}$$

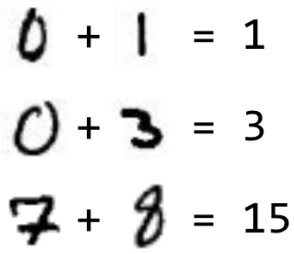
E.2 Exponentiated Gradient Descent

As suggested by Dickens *et al.* (2022), we minimize the energy loss with respect to the symbolic parameters constrained to the unit simplex via normalized exponentiated gradient descent [Kivinen and Warmuth, 1997; Shalev-Shwartz, 2012]. Then, minimization over neural parameters is performed with standard gradient descent. With an initial step size parameter $\eta > 0$, the parameter updates are

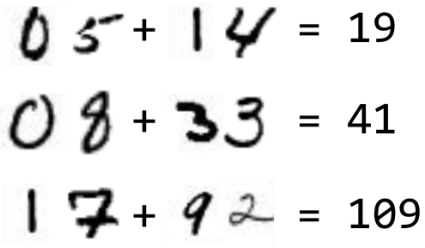
$$\mathbf{w}_{nn}^{k+1} = \mathbf{w}_{nn}^k + \eta \nabla_{w_{nn}} \mathcal{L}(\mathbf{w}_{nn}^k,\mathbf{w}_{psl}^k,S)$$

$$\mathbf{w}_{psl}^{k+1}[i] = \frac{\mathbf{w}_{psl}^k[i]\exp\{-\eta\frac{\partial\mathcal{L}(\mathbf{w}_{nn}^k,\mathbf{w}_{psl}^k,S)}{\partial\mathbf{w}_{psl}^k[i]}\}}{\sum_{j=1}^r \exp\{-\frac{\partial\mathcal{L}(\mathbf{w}_{nn}^k,\mathbf{w}_{psl}^k,S)}{\partial\mathbf{w}_{psl}^k[j]}\}}, \quad \forall i = 1,\cdots,r$$

With this update, the symbolic parameter $\mathbf{w}_{psl}$ is guaranteed to satisfy the simplex constraints.



(a) MNIST-Add1



(b) MNIST-Add2

Figure 6: Example of MNIST-Add1 and MNIST-Add2.

F Dataset Details

In this section, we provide additional information on the MNIST-Add and Visual-Sudoku-Classification datasets. Both datasets are generated from the original MNIST image classification dataset introduced by LeCun *et al.* (1998). Each MNIST image is a 28x28 matrix consisting of pixel grayscale values normalized to lie in the range $[0, 1]$.

F.1 MNIST-Add

The MNIST-Add task, originally proposed by Manhaeve *et al.* (2018), constructs addition equations using MNIST images with only their summation as a label. As shown in Figure 6, equations consist of two numbers each comprised of k MNIST images, i.e., MNIST-Add1 consists of two numbers with one image each ($k = 1$) and MNIST-Add2 consists of two numbers with two images each ($k = 2$). Given two numbers ($2 * k$ images), the classification task is to predict the sum.

Generation Addition examples are created by shuffling a list of MNIST images and then partitioning, in order, pairs of numbers. For example, let the corresponding list of MNIST images be [0, 3, 4, 5, 7, 8]. First this list is shuffled, [8, 3, 0, 7, 5, 4], and then partitioned into $2 * k$ tuples in order. In this scenario, MNIST-Add1 creates 3 addition examples, [[8, 3], [0, 7], [5, 4]].

Overlap The process for generating addition examples for overlap variations is the same, but the list of MNIST images contains duplicates. Specifically, a list of $n \in \{40, 60, 80\}$ unique MNIST images are randomly selected without replacement from the original MNIST train split. Then, a list of $m \in \{0, n/2, n\}$ images are randomly selected with replacement from these n images. These two lists are combined to create a final list of MNIST images ($n + m$ images). This list is used to generate MNIST-Add examples using the process

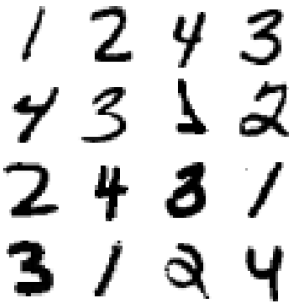


Figure 7: An example of a valid Visual-Sudoku-Classification puzzle.

described above. This process is then repeated to generate a validation set and then repeated again to generate the test set. The MNIST images in the test set are pulled from the original MNIST test split to avoid leaking data and $n = 1000$.

F.2 Visual-Sudoku-Classification

Inspired by the Visual Sudoku problem proposed by Wang *et al.* (2019), Augustine *et al.* (2022) introduced a novel NeSy task, **Visual-Sudoku-Classification**. In this task, 4x4 Sudoku puzzles are constructed using unlabeled MNIST images, e.g., Figure 7. The model must identify whether a puzzle is correct, i.e., no duplicate digits in any row, column, or square.

Generation Puzzles are created from a list of MNIST images, where this list has an equal representation of each class (e.g., zeroes, ones, twos, and threes). To create a "correct" puzzle, four images of each class are randomly selected without replacement from this list and arranged in a layout that adheres to the traditional sudoku puzzle rules. This layout is randomly chosen from all possible correct solutions. The first image represents the top-left corner, and the final image represents the bottom-right corner of the puzzle. For example, Figure 7 would be $\{1, 2, 4, 3, 4, 3, 1, 2, 2, 4, 3, 1, 3, 1, 2, 4\}$.

In addition to generating correctly solved Sudoku puzzles, incorrect puzzles are generated. Instead of randomly creating puzzles and checking if they are correct, we begin with the correct puzzles and corrupt them. In this way, we hope to create puzzles that are more subtle and closer to the incorrect puzzles that a human may create, as opposed to randomly generated puzzles that may be obviously incorrect.

The corruptions are done in one of two ways: *replacement* or *substitution*. A replacement corruption chooses a random cell and replaces it with a random image of another class. Replacement images are chosen uniformly from the same split. A substitution corruption randomly chooses two cells in the same puzzle and swaps them.

Each correct puzzle has one corrupted puzzle made from it, resulting in a balanced dataset. A fair coin is flipped for each puzzle to decide which corruption method will be used. After each corruption is made, a fair coin is flipped to see if the process continues. After the complete corruption process, the puzzle is checked to ensure it is not a valid Sudoku puzzle. If the puzzle is invalid, it is added to the split. Otherwise, the process is repeated using the same correct puzzle.

Overlap The process for generating puzzle examples for overlap variations is the same, but the list of MNIST images contains duplicates. Specifically, a list of $n \in \{64, 128, 256\}$ unique MNIST images are randomly selected without replacement from the original MNIST train split, with an equal representation of four classes (zeros, ones, twos, and threes). Then, a list of $m \in \{0, n, 3.0 \cdot n\}$ images are randomly selected with replacement from these n images, where there is an equal representation of the four class. These two lists are combined to create a final list of MNIST images ($n + m$ images). This list is used to generate puzzles using the process described above. This process is then repeated to generate a validation set and then repeated again to generate the test set. The MNIST images in the test set are pulled from the original MNIST test split to avoid leaking data and $n = 1000$.

Order	Layer	Parameter	Value
1	Convolutional	Kernel Size	5
		Output Channels	6
2	Max Pooling	Pooling Width	2
		Pooling Height	2
		Activation	ReLU
3	Convolutional	Kernel Size	5
		Output Channels	16
4	Max Pooling	Pooling Width	2
		Pooling Height	2
		Activation	ReLU
5	Fully Connected	Input Shape	256
		Output Shape	120
		Activation	ReLU
6	Fully Connected	Input Shape	120
		Output Shape	84
		Activation	ReLU
7	Fully Connected	Input Shape	84
		Output Shape	10
		Activation	Softmax

Table 2: Neural architecture used in NeuPSL for both MNIST-Add and Visual-Sudoku-Classification experiments.

G NeuPSL Models

This section provides an overview of the NeuPSL models used in the *Experimental Evaluation*. The subsequent subsections will examine the symbolic model, neural model, and hyperparameters employed for each setting.

G.1 MNIST-Add1

The NeuPSL model for the MNIST-Add1 experiment integrates the neural model summarized in Table 2 with the symbolic model depicted in Figure 8. The symbolic model contains the following predicates:

- **NEURAL**(**`Img`**, **`X`**) The NEURAL predicate is the class probability for each image as inferred by the neural network. `Img` is MNIST image identifier and `X` is a digit class that the image may represent.
- **DIGITSUM**(**`X`**, **`Y`**, **`Z`**) The DIGITSUM predicate determines if two digits (`X` and `Y`) sum to a number (`Z`). For

 # Digit Sums

w_1 : NEURAL(Img1, X) ∧ NEURAL(Img2, Y) ∧ DIGITSUM(X, Y, Z) → SUM(Img1, Img2, Z)

w_2 : ¬NEURAL(Img1, X) ∧ NEURAL(Img2, Y) ∧ DIGITSUM(X, Y, Z) → ¬SUM(Img1, Img2, Z)

w_3 : NEURAL(Img1, X) ∧ ¬NEURAL(Img2, Y) ∧ DIGITSUM(X, Y, Z) → ¬SUM(Img1, Img2, Z)

 # Digit Constraints

w_4 : NEURAL(Img1, +X) >= SUM(Img1, Img2, Z){X : POSSIBLEDIGITS(X, Z)}

w_5 : NEURAL(Img2, +X) >= SUM(Img1, Img2, Z){X : POSSIBLEDIGITS(X, Z)}

 # Simplex Constraints

SUM(Img1, Img2, +Z) = 1.

Figure 8: NeuPSL MNIST-Add1 Symbolic Model

Hyperparameter	Tuning Range	Final Value
Neural Learning Rate	{1e-2, 1e-3, 1e-4}	1e-3
ADMM Max Iterations	{50, 100, 500, 1000}	500

Table 3: NeuPSL hyperparameters for the MNIST-Add1 experiment.

example, DIGITSUM$(4, 5, 9)$ would return 1 as 4 added to 5 is 9. Conversely, DIGITSUM$(2, 2, 5)$ would return 0 as 2 added to 2 is not 5.

- **SUM**(**Img1**, **Img2**, **Z**) The SUM predicate is the probability that the digits represented in the images identified by arguments Img1 and Img2 add up to the number identified by the argument Z. This predicate instantiates decision variables, i.e., variables from this predicate are not fixed during inference and learning as described in the NeSy EBM, NeuPSL, and Inference and Learning sections.
- **POSSIBLEDIGITS**(**X**, **Z**) The POSSIBLEDIGITS predicate determines if a digit (X) can be included in a sum that equals a number (Z). For example, POSSIBLEDIGITS$(9, 0)$ would return 0 as no positive digit when added to 9 will equal 0. Conversely, POSSIBLEDIGITS$(9, 17)$ would return 1 as 8 added to 9 equals 17.

The *Digit Sums* rules represents the summation of the two images Img1 and Img2, i.e., if the neural model labels the image id Img1 as digit X and Img2 as Y and the digits X and Y sum to Z then the sum of the images must be Z.

The *Digit Constraints* rules restrict the possible values of the SUM predicate based on the neural model's prediction. For instance, if the neural model predicts that the digit label for image Img1 is 1, then the sum that Img1 is involved in cannot be any less than 1 or greater than 10.

Hyperparameters Table 3 presents the hyperparameter values and tuning ranges for the NeuPSL MNIST-Add1 models. The hyperparameter search was conducted on a single split generated from a list of 600 MNIST images, with the best parameters applied to all data settings. Any unspecified values were left at their default settings. The *ADMM Max Iterations* parameter refers to the number of ADMM iterations conducted between each step of gradient descent during the learning process. The *Neural Learning Rate* parameter refers to the learning rate of the neural model used to predict image labels.

G.2 MNIST-Add2

The NeuPSL model for the MNIST-Add2 experiment integrates the neural model summarized in Table 2 with the symbolic model depicted in Figure 9. The symbolic model contains the following predicates:

- **NEURAL**(**Img**, **X**) The NEURAL predicate is the class probability for each image as inferred by the neural network. Img is MNIST image identifier and X is a digit class that the image may represent.
- **DIGITSUM**(**X**, **Y**, **Z**) The DIGITSUM predicate determines if two digits (X and Y) sum to a number (Z). For example, DIGITSUM$(4, 5, 9)$ would return 1 as 4 added to 5 is 9. Conversely, DIGITSUM$(2, 2, 5)$ would return 0 as 2 added to 2 is not 5.
- **SUM**(**Img1**, **Img2**, **Img3**, **Img4**, **Z**) The SUM predicate is the probability that the numbers represented in the images identified by arguments (Img1, Img2) and (Img3, Img4) add up to the number identified by the argument Z. This predicate instantiates decision variables, i.e., variables from this predicate are not fixed during inference and learning as described in the NeSy EBM, NeuPSL, and Inference and Learning sections.
- **POSSIBLETENDIGITS**(**X**, **Z**) POSSIBLETENDIGITS takes a 0 or 1 value representing whether the digit identified by the argument X is possible when it is in the tens place of a number involved in a sum that totals to the number identified by the argument Z. For instance POSSIBLETENDIGITS$(9, 70) = 0$ as no positive number added to a number with a 9 in the tens place, e.g., 92, equals 70, while POSSIBLETENDIGITS$(9, 170) = 1$ as 78 added to 92 is 170.
- **POSSIBLEONESDIGITS**(**X**, **Z**)POSSIBLEONESDIGITS takes a 0 or 1 value representing whether the digit identified by the argument X is possible when it is in the ones place of a number involved in a sum that totals to the number identified by the argument Z. For instance

 # Tens Digit Sums

w_1 : NEURAL(Img1, X) ∧ NEURAL(Img3, Y) ∧ DIGITSUM(X, Y, Z) → IMAGEDIGITSUM(Img1, Img3, Z)

w_2 : ¬NEURAL(Img1, X) ∧ NEURAL(Img3, Y) ∧ DIGITSUM(X, Y, Z) → ¬IMAGEDIGITSUM(Img1, Img3, Z)

w_3 : NEURAL(Img1, X) ∧ ¬NEURAL(Img3, Y) ∧ DIGITSUM(X, Y, Z) → ¬IMAGEDIGITSUM(Img1, Img3, Z)

 # Ones Digit Sums

w_4 : NEURAL(Img2, X) ∧ NEURAL(Img4, Y) ∧ DIGITSUM(X, Y, Z) → IMAGEDIGITSUM(Img2, Img4, Z)

w_5 : ¬NEURAL(Img2, X) ∧ NEURAL(Img4, Y) ∧ DIGITSUM(X, Y, Z) → ¬IMAGEDIGITSUM(Img2, Img4, Z)

w_6 : NEURAL(Img2, X) ∧ ¬NEURAL(Img4, Y) ∧ DIGITSUM(X, Y, Z) → ¬IMAGEDIGITSUM(Img2, Img4, Z)

 # Place Digit Sums

IMAGEDIGITSUM(Img1, Img3, Z10) ∧ IMAGEDIGITSUM(Img2, Img4, Z1) ∧ PLACENUMBERSUM(Z10, Z1, Z)
→ SUM(Img1, Img2, Img3, Img4, Z)

¬IMAGEDIGITSUM(Img1, Img3, Z10) ∧ IMAGEDIGITSUM(Img2, Img4, Z1) ∧ PLACENUMBERSUM(Z10, Z1, Z)
→ ¬SUM(Img1, Img2, Img3, Img4, Z)

IMAGEDIGITSUM(Img1, Img3, Z10) ∧ ¬IMAGEDIGITSUM(Img2, Img4, Z1) ∧ PLACENUMBERSUM(Z10, Z1, Z)
→ ¬SUM(Img1, Img2, Img3, Img4, Z)

 # Tens Digit Constraints

w_7 : NEURAL(Img1, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLETENSDIGITS(X, Z)}

w_8 : NEURAL(Img3, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLETENSDIGITS(X, Z)}

 # Ones Digit Constraints

w_9 : NEURAL(Img2, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLEONESDIGITS(X, Z)}

w_{10} : NEURAL(Img4, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLEONESDIGITS(X, Z)}

 # Digit Sum Constraints

w_{11} : NEURAL(Img1, +X) >= IMAGEDIGITSUM(Img1, Img3, Z){X : POSSIBLEDIGITS(X, Z)}

w_{12} : NEURAL(Img3, +X) >= IMAGEDIGITSUM(Img1, Img3, Z){X : POSSIBLEDIGITS(X, Z)}

w_{13} : NEURAL(Img2, +X) >= IMAGEDIGITSUM(Img2, Img4, Z){X : POSSIBLEDIGITS(X, Z)}

w_{14} : NEURAL(Img4, +X) >= IMAGEDIGITSUM(Img2, Img4, Z){X : POSSIBLEDIGITS(X, Z)}

 # Number Sum Constraints

IMAGEDIGITSUM(Img1, Img3, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLETENSSUMS(X, Z)}

IMAGEDIGITSUM(Img2, Img4, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLEONESSUMS(X, Z)}

 # Simplex Constraints

SUM(Img1, Img2, Img3, Img4, +X) = 1.

IMAGEDIGITSUM(Img1, Img2, +X) = 1.

Figure 9: NeuPSL MNIST-Add2 Symbolic Model

 # Row Constraint

$\text{NEURAL}(\texttt{Puzzle}, +\texttt{X}, \texttt{Y}, \texttt{Number}) = 1.$

 # Column Constraint

$\text{NEURAL}(\texttt{Puzzle}, \texttt{X}, +\texttt{Y}, \texttt{Number}) = 1.$

 # Block Constraint

$$\begin{aligned}
&\text{NEURAL}(\texttt{Puzzle}, \text{“0”}, \text{“0”}, \texttt{Number}) + \text{NEURAL}(\texttt{Puzzle}, \text{“0”}, \text{“1”}, \texttt{Number}) \\
&\quad + \text{NEURAL}(\texttt{Puzzle}, \text{“1”}, \text{“0”}, \texttt{Number}) + \text{NEURAL}(\texttt{Puzzle}, \text{“1”}, \text{“1”}, \texttt{Number}) = 1. \\
&\text{NEURAL}(\texttt{Puzzle}, \text{“2”}, \text{“0”}, \texttt{Number}) + \text{NEURAL}(\texttt{Puzzle}, \text{“2”}, \text{“1”}, \texttt{Number}) \\
&\quad + \text{NEURAL}(\texttt{Puzzle}, \text{“3”}, \text{“0”}, \texttt{Number}) + \text{NEURAL}(\texttt{Puzzle}, \text{“3”}, \text{“1”}, \texttt{Number}) = 1. \\
&\text{NEURAL}(\texttt{Puzzle}, \text{“0”}, \text{“2”}, \texttt{Number}) + \text{NEURAL}(\texttt{Puzzle}, \text{“0”}, \text{“3”}, \texttt{Number}) \\
&\quad + \text{NEURAL}(\texttt{Puzzle}, \text{“1”}, \text{“2”}, \texttt{Number}) + \text{NEURAL}(\texttt{Puzzle}, \text{“1”}, \text{“3”}, \texttt{Number}) = 1. \\
&\text{NEURAL}(\texttt{Puzzle}, \text{“2”}, \text{“2”}, \texttt{Number}) + \text{NEURAL}(\texttt{Puzzle}, \text{“2”}, \text{“3”}, \texttt{Number}) \\
&\quad + \text{NEURAL}(\texttt{Puzzle}, \text{“3”}, \text{“2”}, \texttt{Number}) + \text{NEURAL}(\texttt{Puzzle}, \text{“3”}, \text{“3”}, \texttt{Number}) = 1.
\end{aligned}$$

 # Pin First Column

$w_2 : \text{FIRSTPUZZLE}(\texttt{Puzzle}, \texttt{X}, \texttt{Y}) - \text{NEURAL}(\texttt{Puzzle}, \texttt{X}, \texttt{Y}) = 0.0$

Figure 10: NeuPSL Visual-Sudoku-Classification Symbolic Model

$\text{POSSIBLEONESDIGITS}(9, 7) = 0$ as no positive number added to a number with a 9 in the ones place, e.g., 9, equals 7 while $\text{POSSIBLEONESDIGITS}(9, 170) = 1$ as 71 added to 99 is 170.

- **IMAGEDIGITSUM**(**Img1**, **Img2**, **Z**) The IMAGEDIGITSUM predicate is the probability that the digits represented in the images specified by `Img1` and `Img2` will sum up to the number indicated by the argument `Z`. These variables are considered latent in the NeuPSL model as there are no truth labels for sums of images in the ones or tens places.
- **PLACENUMBERSUM**(**Z10**, **Z1**, **Z**) The PLACENUMBERSUM predicate takes a 0 or 1 value representing whether the sum of the numbers `Z10` and `Z1`, where `Z10` is the sum of digits in the tens place and `Z1` is the sum of digits in the one place, adds up to the number `Z`. For instance $\text{PLACENUMBERSUM}(1, 15, 25)$ is 1 as $1 \cdot 10 + 15 = 25$.

The *Tens Digit Sums* and *Ones Digit Sums* rules compute the sum of two images in the same manner as the *Digit Sums* rules in the MNIST-Add1 model. The sum of the digits is captured by the latent variables instantiated by the predicate IMAGEDIGITSUM.

The *Place Digit Sums* rules use the value of the IMAGEDIGITSUM variables to infer the sum of the images. More specifically, if the IMAGEDIGITSUM of the images in the tens place, `Img1` and `Img3`), is $Z10$, and the IMAGEDIGITSUM of the images in the ones place, `Img2` and `Img4`) is $Z1$, and if according to PLACENUMBERSUM the sum of the numbers $Z10$ and $Z1$ is Z, then the SUM of the images must be Z. Notice that these rules are hard constraints as it is always possible and desirable to find values of the IMAGEDIGITSUM and SUM variables that satisfy these relations.

The *Tens Digit Constraint* rules restrict the possible values of the SUM predicate based on the neural model's prediction for the digit in the tens place of a number. For instance, if the neural model predicts that the digit label for the image `Img1` is 1 and `Img1` is in the tens place of a number, then the sum that `Img1` is involved in cannot be any less than 10 or greater than 118.

The *Ones Digit Constraint* rules restrict the possible values of the SUM predicate based on the neural model's prediction for the digit in the ones place of a number. For instance, if the neural model predicts that the digit label for the image `Img2` is 5 and `Img2` is in the one place of a number, then the sum that `Img2` is involved in cannot be any less than 5 or greater than 194.

The *Number Sum Constraint* rules limit the values that IMAGEDIGITSUM and SUM can take using constraints representing the possible sums in the tens and ones place. For instance, if the IMAGEDIGITSUM of two images, `Img1` and `Img3`, both in the tens place of two numbers being added, is 17, then the SUM cannot be less than 170 or greater than 188. Furthermore, if the IMAGEDIGITSUM of two images, `Img2` and `Img4`, both in the tens place of two numbers being added, is 17, then the SUM cannot be less than 17 or greater than 197, and must have a 7 in the ones place.

Hyperparameter	Tuning Range	Final Value
Neural Learning Rate	{1e-2, 1e-3, 1e-4}	1e-3
ADMM Max Iterations	{50, 100, 500, 1000}	100

Table 4: NeuPSL hyperparameters for the MNIST-Add2 experiments.

Hyperparameters Table 4 presents the hyperparameter values and tuning ranges for the NeuPSL MNIST-Add2 models. The hyperparameter search was conducted on a single split generated from a list of 600 MNIST images, with the best parameters applied to all data settings. Any unspecified values were left at their default settings. The *ADMM Max Iterations* parameter refers to the number of ADMM iterations

Number of Images	Number of Puzzles	Hyperparameter	Tuning Range	Final Value
~64	4	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.001
		ADMM Max Iterations	{50, 100, 1000}	100
	10	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.01
		ADMM Max Iterations	{50, 100, 1000}	50
	20	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.001
		ADMM Max Iterations	{50, 100, 1000}	1000
~160	10	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.001
		ADMM Max Iterations	{50, 100, 1000}	100
	20	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.001
		ADMM Max Iterations	{50, 100, 1000}	100
	40	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.001
		ADMM Max Iterations	{50, 100, 1000}	100
~320	20	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.001
		ADMM Max Iterations	{50, 100, 1000}	50
	40	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.001
		ADMM Max Iterations	{50, 100, 1000}	50
	80	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.0001
		ADMM Max Iterations	{50, 100, 1000}	100
~1600	100	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.0001
		ADMM Max Iterations	{50, 100, 1000}	100
~3200	200	Neural Learning Rate	{1e-2, 1e-3, 1e-4}	0.01
		ADMM Max Iterations	{50, 100, 1000}	100

Table 5: NeuPSL hyperparameters for the Visual-Sudoku-Classification experiments.

conducted between each step of gradient descent during the learning process. The *Neural Learning Rate* parameter refers to the learning rate of the neural model used to predict image labels.

G.3 Visual-Sudoku-Classification

The NeuPSL model for the Visual-Sudoku-Classification experiment integrates the neural model summarized in Table 2 with the symbolic model depicted in Figure 10. The symbolic model contains the following predicates:

- **NEURAL**(**`Puzzle`**, **`X`**, **`Y`**, **`Number`**) The NEURAL predicate contains the output class probability for each digit image inferred by the neural network. `Puzzle` is sudoku puzzle's identifier, `X` and `Y` represent the location of image in the puzzle, and `Number` is a digit that image may represent.
- **DIGIT**(**`Puzzle`**, **`X`**, **`Y`**, **`Number`**) The DIGIT predicate has the same arguments as the NEURAL predicate, representing PSL's digit prediction on the image.
- **FIRSTPUZZLE, X, Y**(**`Puzzle`**) The FIRSTPUZZLE predicate pins the values for the first row of the first puzzle to an arbitrary assignment. This is used to force the neural model to learn the correct label representation for easier evaluation.

The *Row Constraint*, *Column Constraint*, and *Block Constraint* rules encode the standard Sudoku rules into constraints. These constraints restrict multiple instances of a digit from appearing in a row, column, or block, respectively.

The *Pin First Column* rules are used to assign arbitrary classes to the first row of a Sudoku puzzle. The first row of the first correct puzzle from the training set is used to determine this arbitrary label assignment. By assigning the first row to arbitrary classes, the neural model is provided a starting point for differentiating between the different classes and makes the final evaluation easier.

Hyperparameters Table 5 presents the hyperparameter values and tuning ranges for the NeuPSL Visual-Sudoku-Classification models. A hyperparameter search was conducted for each data setting on the initial split, with the optimal hyperparameters applied to all subsequent splits. Any unspecified values were left at their default settings. The *ADMM Max Iterations* parameter refers to the number of ADMM iterations conducted between each step of gradient descent during the learning process. The *Neural Learning Rate* parameter refers to the learning rate of the neural model used to predict image labels.

G.4 Citation Network Node Classification

The NeuPSL model for the Citation Network Node Classification experiments integrates a single-layered neural model with the symbolic model depicted in Figure 11. The single-layer neural model connects the input to a dense-layered output containing a soft-max activation, kernel regularizer, and bias regularizer. The symbolic model contains the following predicates:

- **NEURAL**(**`Paper`**, **`Label`**) The NEURAL predicate contains the output class probability for each paper as

 # L2 Loss

$w_1 : \text{NEURAL}(\texttt{Paper}, \texttt{Label}) = \text{CATEGORY}(\texttt{Paper}, \texttt{Label})$

 # Label Propagation

$w_2 : \text{LINK}(\texttt{Paper1}, \texttt{Paper2}) \wedge \text{CATEGORY}(\texttt{Paper1}, \texttt{Label}) \rightarrow \text{CATEGORY}(\texttt{Paper2}, \texttt{Label})$

 # Simplex Constraints

$\text{CATEGORY}(\texttt{Paper}, +\texttt{Label}) = 1.$

Figure 11: NeuPSL Citation Network Symbolic Model

Dataset	Model	Neural/Symbolic	Hyperparameter	Tuning Range	Final Value
Citeseer	NeuPSL$_{LP}$	Neural	Hidden Layer Size	{None, 32, 64, 128}	None
			Learning Rate	{2.0e-0, 1.5e-0, 1.0e-0, 1.0e-1}	1.0e-0
			Weight Regularization	{5.0e-5, 1.0e-5, 5.0e-6, 1.0e-6, 5.0e-7}	1.0e-6
		Symbolic	ADMM Step Size	{0.1, 1.0}	1.0
			ADMM Max Iterations	{25, 100, 1000}	25
			Alpha	{0.0, 0.1}	0.0
			Gradient Steps	{5, 50, 100}	50
			Gradient Step Size	{1.0e-2, 1.0e-3, 1.0e-8}	1.0e-8
	NeuPSL$_{LP+FP}$	Neural	Hidden Layer Size	{None, 32, 64, 128}	None
			Learning Rate	{2.0e-0, 1.5e-0, 1.0e-0, 1.0e-1}	1.5e-0
			Weight Regularization	{5.0e-5, 1.0e-5, 5.0e-6, 1.0e-6, 5.0e-7}	1.0e-6
		Symbolic	ADMM Step Size	{0.01, 0.1, 1.0}	1.0
			ADMM Max Iterations	{25, 100, 1000}	1000
			Alpha	{0.0, 0.1}	0.0
			Gradient Steps	{5, 50, 100}	100
			Gradient Step Size	{1.0e-2, 1.0e-3, 1.0e-8}	1.0e-2
Cora	NeuPSL$_{LP}$	Neural	Hidden Layer Size	{None, 32, 64, 128}	None
			Learning Rate	{2.0e-0, 1.5e-0, 1.0e-0, 1.0e-1}	1.5e-0
			Weight Regularization	{5.0e-5, 1.0e-5, 5.0e-6, 1.0e-6, 5.0e-7}	5.0e-5
		Symbolic	ADMM Step Size	{0.01, 0.1, 1.0}	1.0
			ADMM Max Iterations	{25, 100, 1000}	25
			Alpha	{0.0, 0.1}	0.0
			Gradient Steps	{5, 50, 100}	50
			Gradient Step Size	{1.0e-2, 1.0e-3, 1.0e-8}	1.0e-8
	NeuPSL$_{LP+FP}$	Neural	Hidden Layer Size	{None, 32, 64, 128}	None
			Learning Rate	{2.0e-0, 1.5e-0, 1.0e-0, 1.0e-1}	1.5e-0
			Weight Regularization	{5.0e-5, 1.0e-5, 5.0e-6, 1.0e-6, 5.0e-7}	5.0e-7
		Symbolic	ADMM Step Size	{0.01, 0.1, 1.0}	1.0
			ADMM Max Iterations	{25, 100, 1000}	1000
			Alpha	{0.0, 0.1}	0.0
			Gradient Steps	{5, 50, 100}	100
			Gradient Step Size	{1.0e-2, 1.0e-3, 1.0e-8}	1.0e-3

Table 6: NeuPSL hyperparameters for the citation network node classification experiments.

inferred by the neural network. `Paper` is the identifier and `Label` is the category it can take.

- **CATEGORY**(**`Paper`**, **`Label`**) The CATEGORY predicate has the same arguments as the NEURAL predicate and represents PSL's label prediction on the paper.
- **LINK**(**`Paper1`**, **`Paper2`**) The LINK predicate denotes whether two papers share a citation link.

The *Label Propagation* rule propagates node labels to neighbors. In this sense, it encodes the idea that papers sharing a citation link are likely to have the same underlying label category.

Hyperparameters Table 6 presents the hyperparameter values and tuning ranges for the NeuPSL citation network node classification models. A hyperparameter search was conducted for each data setting on the initial split, with the optimal hyperparameters applied to all subsequent splits. The search process was divided into two distinct stages: a neural hyperparameter search and a symbolic hyperparameter search. The optimal hyperparameters identified during

Order	Layer	Parameter	Value
1	Convolutional	Input Shape	28×28
		Kernel Size	5
		Output Channels	6
		Activation	ELU
2	Max Pooling	Pooling Width	2
		Pooling Height	2
3	Convolutional	Kernel Size	5
		Output Channels	16
		Activation	ELU
4	Max Pooling	Pooling Width	2
		Pooling Height	2
5	Fully Connected	Input Shape	256
		Output Shape	100
		Activation	ELU
6	Concatenation	Input Shape	2×100
		Output Shape	200
		Activation	ELU
7	Fully Connected	Input Shape	200
		Output Shape	84
		Activation	ELU
8	Fully Connected	Input Shape	84
		Output Shape	19
		Activation	Softmax

Table 7: Neural architecture for the MNIST-Add2 CNN baseline [Badreddine *et al.*, 2022].

Order	Layer	Parameter	Value
1	Convolutional	Input Shape	28×28
		Kernel Size	5
		Output Channels	6
		Activation	ELU
2	Max Pooling	Pooling Width	2
		Pooling Height	2
3	Convolutional	Kernel Size	5
		Output Channels	16
		Activation	ELU
4	Max Pooling	Pooling Width	2
		Pooling Height	2
5	Fully Connected	Input Shape	256
		Output Shape	100
		Activation	ELU
6	Concatenation	Input Shape	4×100
		Output Shape	400
		Activation	ELU
7	Fully Connected	Input Shape	400
		Output Shape	128
		Activation	ELU
8	Fully Connected	Input Shape	128
		Output Shape	199
		Activation	Softmax

Table 8: Neural architecture for the MNIST-Add2 CNN baseline [Badreddine *et al.*, 2022].

the neural search were subsequently set during the symbolic search. All neural models were trained for 250 epochs utilizing early stopping on the validation set with a patience of 25. Final hyperparameter values for LP_{PSL} and Neural_{PSL} are the same as NeuPSL_{LP}. Any unspecified values were left at their default settings. The *Hidden Layer Size* parameter refers to the size of a single hidden layer, where "None" removes that hidden layer, resulting in a model with only input and output layers. The *Learning Rate* parameter refers to the learning rate of the neural model. The *Weight Regularization* parameter adds a kernel and bias regularizer to the hidden layer and output. The *ADMM Step Size* parameter refers to the initial step size of the ADMM reasoner. The *ADMM Max Iterations* parameter refers to the number of ADMM iterations conducted between each step of gradient descent during learning. The *Alpha* is a value that weights the importance of the structural gradient passed back from the symbolic potentials and the gradient with respect to the labels. The *Gradient Steps* parameter refers to the number of gradient steps taken for joint learning. The *Gradient Step Size* parameter refers to the step size used in learning the symbolic parameters.

H Baseline Models

This section provides additional details of the baseline models used in the *Experimental Evaluation*. The subsequent subsections will examine the architectural structure and hyperparameters employed for each setting.

H.1 MNIST-Add

The CNN baseline neural models for the MNIST-Add1 and MNIST-Add2 experiments are summarized in Table 7 and Table 8 respectively. These models take as input either two MNIST images (MNIST-Add1) or four MNIST images (MNIST-Add1) and output a probability distribution of the resulting sum. Both models were trained to minimize cross-entropy loss.

Model	Number of Additions	Hyperparameter	Tuning Range	Final
MNIST-Add1	300	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-3
		Batch Size	{16, 32, 64, 128}	32
	3,000	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-3
		Batch Size	{16, 32, 64, 128}	16
	25,000	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-3
		Batch Size	{16, 32, 64, 128}	32
MNIST-Add2	150	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-3
		Batch Size	{16, 32, 64, 128}	32
	1,500	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-3
		Batch Size	{16, 32, 64, 128}	32
	12,500	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-3
		Batch Size	{16, 32, 64, 128}	64

Table 9: CNN baseline hyperparameters for the MNIST-Add1 and MNIST-Add2 experiments.

Hyperparameters Table 9 presents the hyperparameter values and tuning ranges for the baseline MNIST-Add1 and MNIST-Add2 models. A hyperparameter search was conducted for three data sizes on the initial split, with the optimal results applied to all subsequent splits. All experiments involving overlap utilized the best hyperparameters identified from the MNIST-Add1 300 additions and MNIST-Add2 150 additions searches. Any unspecified values were left at their default settings. The *Batch Size* parameter refers to the number of addition examples per batch of training and evaluation. The *Learning Rate* parameter refers to the learning rate of the model used to predict.

Order	Layer	Parameter	Value
1	Convolutional	Input Shape	112×112
		Kernel Size	3
		Output Channels	16
		Activation	ReLU
2	Max Pooling	Pooling Width	2
		Pooling Height	2
3	Convolutional	Kernel Size	3
		Output Channels	16
		Activation	ReLU
4	Max Pooling	Pooling Width	2
		Pooling Height	2
5	Convolutional	Kernel Size	3
		Output Channels	16
		Activation	ReLU
6	Max Pooling	Pooling Width	2
		Pooling Height	2
7	Fully Connected	Input Shape	2304
		Output Shape	256
		Activation	ReLU
8	Fully Connected	Input Shape	256
		Output Shape	256
		Activation	ReLU
9	Fully Connected	Input Shape	256
		Output Shape	128
		Activation	ReLU
10	Fully Connected	Input Shape	128
		Output Shape	1
		Activation	Softmax

Table 10: Neural architecture for the Visual-Sudoku-Classification CNN-Visual baseline [Badreddine *et al.*, 2022].

Order	Layer	Parameter	Value
1	Fully Connected	Input Shape	16
		Output Shape	512
		Activation	ReLU
2	Fully Connected	Input Shape	512
		Output Shape	512
		Activation	ReLU
3	Fully Connected	Input Shape	512
		Output Shape	256
		Activation	ReLU
4	Fully Connected	Input Shape	256
		Output Shape	1
		Activation	ReLU

Table 11: Neural architecture for the Visual-Sudoku-Classification CNN-Digit baseline.

H.2 Visual-Sudoku-Classification

The CNN-Visual and CNN-Digit baseline neural models for the Visual-Sudoku-Classification experiments are summarized in Table 10 and Table 11 respectively. The input to the CNN-Visual baseline takes 16 MNIST images as input and produces a probability distribution indicating the likelihood that the images form a correct puzzle. The input to the CNN-Digit baseline takes 16 MNIST image ground truth labels as input and produces a probability distribution indicating the likelihood that the images' labels form a correct puzzle. Both models were trained to minimize cross-entropy loss.

Hyperparameters Table 12 presents the hyperparameter values and tuning ranges for the CNN-Visual and CNN-Digit baseline neural models. A hyperparameter search was conducted for each data setting on the initial split, with the optimal hyperparameters applied to all subsequent splits. Any unspecified values were left at their default settings. The *Learning Rate* parameter refers to the learning rate of the neural model.

Model	Number of Puzzles	Hyperparameter	Tuning Range	Final
CNN-Visual	10	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-4
	20	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-3
	100	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-2
	200	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-2
CNN-Digit	10	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-3
	20	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-2
	100	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-2
	200	Learning Rate	{1e-3, 1e-4, 1e-5}	1e-2

Table 12: CNN-Visual and CNN-Digit hyperparameters for the Visual-Sudoku-Classification experiment.

Order	Layer	Parameters	Value
1	Graph Conv Layer	Number of Parameters	237056
2	Graph conv Layer	Number of Parameters	390
		Activation	softmax

Table 13: Neural architecture for the citation network node classification GCN model.

Hyperparameter	Tuning Range	Final Value
Hidden Units	{16, 32, 64}	64
Learning Rate	{1e-2, 1e-3}	1e-3
Weight Regularizer	{1.0e-3, 5.0e-4}	1.0e-3

Table 14: GCN hyperparameters for the citation network node classification experiments.

H.3 Citation Network Node Classification

As described in the *Experimental Evaluation*, the LP_{PSL} and Neural_{PSL} baseline models represent the distinct symbolic and neural components used in NeuPSL_{LP}. Therefore, the LP_{PSL} model is depicted in Figure 11, and the Neural_{PSL} model is a single-layered neural model connecting the input to a dense-layered output containing a soft-max activation, kernel regularizer, and bias regularizer. Hyperparameters were set to the best values found for the NeuPSL_{LP} neural hyperparameter search (Table 6).

The GCN model follows the same architecture proposed by Kipf and Welling [2017] and is summarized in Table 13. The GCN takes a collection of node identifiers as input and outputs each node's class label.

Hyperparameters Table 14 presents the hyperparameter values and tuning ranges for the GCN model. Each GCN model was trained with 50 percent dropout, a batch size of 1024, and 1000 epochs (utilizing early stopping on the validation set with a patience of 250). A hyperparameter search was conducted for each data setting on the initial split, with the optimal hyperparameters applied to all subsequent splits. Any unspecified values were left at their default settings.

Method	MNIST-Add1			MNIST-Add2		
	Number of Additions					
	300	3,000	25,000	150	1,500	12,500
CNN	17.16 ± 00.62	78.99 ± 01.14	96.30 ± 00.30	01.31 ± 00.23	01.69 ± 00.27	23.88 ± 04.32
LTNs	69.23 ± 15.68	**93.90 ± 00.51**	80.54 ± 23.33	02.02 ± 00.97	71.79 ± 27.76	77.54 ± 35.55
DPL	**85.61 ± 01.28**	92.59 ± 01.40	-[2]	**71.37 ± 03.90**	**87.44 ± 02.15**	-[2]
NeuPSL	82.58 ± 02.56	**93.66 ± 00.32**	**97.34 ± 00.26**	56.94 ± 06.33	**87.05 ± 01.48**	**93.91 ± 00.37**

Table 15: Test set accuracy and standard deviation on **MNIST-Add**. Results reported here are run and averaged over the same ten splits.

I Extended Evaluation Details

This section provides NeSy model details and expands the *Experimental Evaluation* presented earlier on MNIST-Add and provides inference and learning times for all experiments.

I.1 NeSy Model Details

The NeSy methods used in this work, along with their respective publications and implementation codes, are listed below:

DeepProbLog (DPL): All DPL results use the DPL models presented in [Manhaeve *et al.*, 2021], using default hyperparameters. Code was obtained from github.com/ML-KULeuven/deepproblog.

DeepStochLog (DSL): All DeepStochLog results use the DeepStochLog models presented in [Winters *et al.*, 2022], using default hyperparameters. Code was obtained from github.com/ML-KULeuven/deepstochlog.

Logic Tensor Networks (LTNs): All LTNs results use the LTNs models presented in [Badreddine *et al.*, 2022], using default hyperparameters. Code was obtained from github.com/logictensornetworks/logictensornetworks.

Licenses for NeuPSL, DeepProbLog, DeepStochLog, are under Apache License 2.0 and Logic Tensor Networks are under MIT License.

I.2 MNIST-Add Extended Results

In this section, we conduct an extended analysis of the MNIST-Add experiment by comparing the performance of NeuPSL, DeepProbLog (DPL) [Manhaeve *et al.*, 2021], Logic Tensor Networks (LTNs) [Badreddine *et al.*, 2022] and neural baselines in non-overlap settings with commonly used split sizes in the research community [Manhaeve *et al.*, 2021]. Ten train splits are generated by randomly selecting, without replacement, $n \in \{600, 6000, 50000\}$ unique MNIST images from the original MNIST train split and converted to MNIST additions as described in the *Datasets* appendix. This process is then repeated to create validation and test splits, with the test splits being pulled from the original MNIST test split to prevent data leakage and $n = 10000$.

Table 15 shows the average accuracy and standard deviation for MNIST-Add1 and MNIST-Add2.[3] The best average accuracy and results within a standard deviation of the best are in bold. In all but two settings, NeuPSL is either the highest-performing model or within a standard deviation of the highest-performing model. Moreover, NeuPSL has a markedly lower variance for nearly all training examples in both MNIST-Add tasks.

[3]In the largest data setting, there appeared to be an error with DPL, and the results produced were random. Rather than present these potentially misleadingly low results, we indicate with '-'.

Setting	Method	Inference (sec)	Learning (sec)
Citeseer	NeuPSL$_{LP}$	3.98 ± 0.05	29.90 ± 0.82
	NeuPSL$_{LP+FP}$	4.23 ± 0.05	32.94 ± 0.36
Cora	NeuPSL$_{LP}$	4.00 ± 0.31	33.41 ± 1.23
	NeuPSL$_{LP+FP}$	4.07 ± 0.14	36.50 ± 0.53

Table 16: Inference and learning time for NeuPSL on Citation Network Node Classification experiments presented in Section 6.3.

Unique Digits	Puzzles	Inference (sec)	Learning (sec)
64	4	4.65 ± 0.16	43.18 ± 1.35
	8	6.47 ± 0.19	52.56 ± 1.08
	16	12.56 ± 0.66	68.64 ± 0.89
128	8	4.54 ± 0.07	52.45 ± 0.94
	16	6.48 ± 0.18	68.91 ± 1.01
	32	12.62 ± 0.52	102.60 ± 0.90
256	8	4.67 ± 0.20	68.62 ± 1.04
	16	6.53 ± 0.30	102.76 ± 2.05
	32	12.59 ± 0.53	170.66 ± 5.82

Table 17: Inference and learning time for NeuPSL on Visual Sudoku Puzzle Classification experiments presented in Section 6.2.

I.3 Inference and Learning Runtime

Table 16 summarizes the inference and learning time for NeuPSL on Citation Network Node Classification experiments presented in Section 6.3 and Table 17 summarizes the inference and learning time for NeuPSL on Visual Sudoku Puzzle Classification experiments presented in Section 6.2.

Figure 12 summarizes the inference and learning times associated with the **MNIST-Add** experiments described in Section 6.1. When evaluating the performance of the NeSy meth-

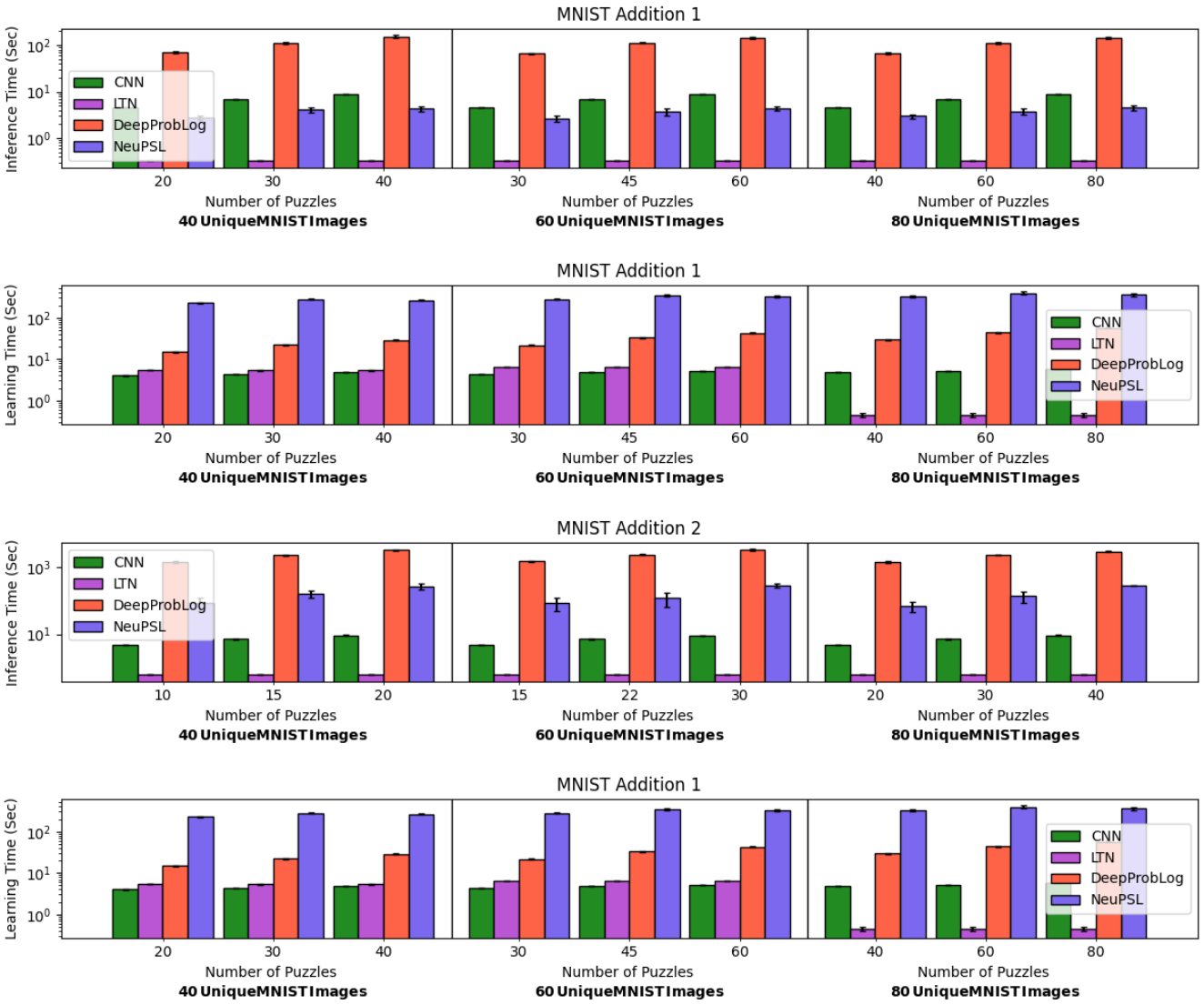


Figure 12: Inference and learning time for **MNIST-Add** experiments presented in Section 6.1.

ods that perform complex symbolic inference (DPL and NeuPSL), a trade-off is observed. NeuPSL inference runs an order of magnitude faster than DPL but, surprisingly, takes longer to train on roughly the same number of gradient steps. This timing difference derives from NeuPSL taking full gradient steps over the entire train dataset while DPL takes batched stochastic gradient steps. Symbolic inference is a subprocess of NeSy-EBM learning, and DPL performs inference over a single addition, while NeuPSL performs inference over every addition. An interesting direction for future work is to take batched gradient steps during NeuPSL learning, where the batches contain a set of overlapping additions.

Compared with the CNN and LTN models, DPL and NeuPSL run orders of magnitude slower. CNN and LTN inference is equivalent to making a feed-forward pass through a neural network. This will, therefore, be significantly faster than the complex symbolic inference done in DPL and NeuPSL, but comes with a decrease in predictive performance.

J Computational Hardware Details

All timing experiments were performed on an Ubuntu 22.04.1 Linux machine with Intel Xeon Processor E5-2630 v4 at 3.10GHz.