NeuPSL: Neural Probabilistic Soft Logic

Connor Pryor *
UC Santa Cruz
Santa Cruz, CA 95064
cfpryor@ucsc.edu

Charles Dickens *
UC Santa Cruz
Santa Cruz, CA 95064
cadicken@ucsc.edu

Eriq Augustine
UC Santa Cruz
Santa Cruz, CA 95064
eaugusti@ucsc.edu

Alon Albalak
UC Santa Barbara
Santa Barbara, CA 93106
alon_albalak@ucsb.edu

William Wang
UC Santa Barbara
Santa Barbara, CA 93106
william@cs.ucsb.edu

Lise Getoor
UC Santa Cruz
Santa Cruz, CA 95064
getoor@ucsc.edu

Abstract

We present *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 explicitly model the boundary between neural and symbolic representations, we introduce *NeSy Energy-Based Models*, a general family of energy-based models that combine neural and symbolic reasoning. Using this framework, we show how to seamlessly integrate neural and symbolic parameter learning and inference. We perform an extensive empirical evaluation and show that NeuPSL outperforms existing methods on joint inference and has significantly lower variance in almost all settings.

1 Introduction

The integration of deep learning and symbolic reasoning is one of the earliest, yet open, challenges in the machine learning community [d'Avila Garcez et al., 2002]. A promising area of research that incorporates neural perception into logic-based reasoning is neuro-symbolic computing (NeSy) [Besold et al., 2017, d'Avila Garcez et al., 2019, De Raedt et al., 2020]. This integration has the benefit of allowing for expressive models that are able to perform joint inference (structured prediction), i.e., jointly predicting multiple labels or tasks, and joint learning, i.e., parameter learning over a joint loss function. Joint inference and learning have been used in strictly deep neural architectures to great success; graph neural networks [Wu et al., 2021], conditional random fields [Zheng et al., 2015], hidden Markov models [Li et al., 2013], etc. Furthermore, models designed for tasks such as translation [Bahdanau et al., 2015] or image segmentation [Nowozin and Lampert, 2011] without joint inference are unable to accurately distinguish predictions. For example, in translation, understanding the difference between homonyms such as *fair* (equitable vs. beautiful) or in image segmentation, making locally consistent decisions about the foreground and background.

In this paper, we introduce a principled NeSy method that integrates deep learning with hard and soft logical constraints prioritizing joint learning and inference. Our approach, *Neural Probabilistic Soft Logic* (NeuPSL), extends probabilistic soft logic (PSL) [Bach et al., 2017], a state-of-the-art highly scalable probabilistic programming framework that is able to both reason statistically (using probabilistic inference) and logically (using soft rules). We choose PSL because it encodes an underlying graphical model that supports scalable and convex joint inference. Additionally, PSL has been shown to support a wide variety of joint inference and learning tasks including natural language processing [Beltagy et al., 2014, Deng and Wiebe, 2015, Liu et al., 2016, Rospocher, 2018], data

*These authors contributed equally to this work.

Preprint. Under review.

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].

Our key contributions are: 1) We define a family of energy-based models called Neuro-Symbolic Energy-Based Models (NeSy-EBMs), 2) We introduce NeuPSL and highlight how it fits into the NeSy ecosystem, paying particular attention to how it supports joint inference and learning, and 3) We perform an extensive evaluation, showing that NeuPSL consistently outperforms existing approaches on joint inference in low data settings while maintaining a significantly lower variance in most settings.

2 Related Work

Neuro-symbolic computing (NeSy) is a very active area of research that aims to incorporate logic-based reasoning with neural computation 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. [2021a], d'Avila Garcez et al. [2019], De Raedt et al. [2020], Lamb et al. [2020], Badreddine et al. [2022]. This integration allows for interpretability and reasoning through symbolic knowledge, while providing robust learning and efficient inference with neural networks. For an in-depth introduction, we refer the reader to these excellent surveys 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 the universal function approximation properties of neural networks 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 compare with them 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 neuro-symbolic 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], DeepProbLog (DPL) Manhaeve et al. [2021a], 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

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*: $E : \mathcal{Y} \times \mathcal{X} \to \mathbb{R}$. 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 this well-known formulation, we introduce *Neuro-Symbolic Energy-Based Models* (NeSy-EBMs). 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

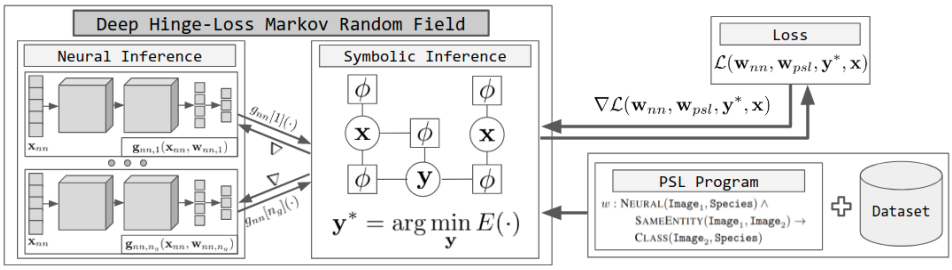


Figure 1: NeuPSL inference and learning pipeline.

neural parameters, $\mathbf{w}_{nn} \in \mathcal{W}_{nn}$, and symbolic parameters, $\mathbf{w}_{sy} \in \mathcal{W}_{sy}$. NeSy energy functions are written as $E : \mathcal{Y} \times \mathcal{X} \times \mathcal{X}_{nn} \times \mathcal{W}_{nn} \times \mathcal{W}_{sy} \to \mathbb{R}$. Formally,

Definition 1 (Neuro-Symbolic Energy-Based Models)**.** *Let* $\mathbf{y} = [y_i]_{i=1}^{n_y}$ *and* $\mathbf{x} = [x_i]_{i=1}^{n_x}$ *be vectors of real valued variables with symbolic interpretations. Let* $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$ *be neural networks with corresponding parameters* $\mathbf{w}_{nn} = [\mathbf{w}_{nn,i}]_{i=1}^{n_g}$ *and, without loss of generality, inputs* $\mathbf{x}_{nn}$*. A* ***symbolic potential*** *is a function:* $\psi(\mathbf{y}, \mathbf{x}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn})) \in \mathbb{R}$*. A* ***NeSy-EBM energy function****, parameterized by* ***symbolic parameters*** $\mathbf{w}_{sy} = [w_{sy,i}]_{i=1}^{n_{sy}}$*, is a mapping of a vector of symbolic potential outputs,* $\Psi(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) = [\psi_i(\mathbf{y}, \mathbf{x}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))]_{i=1}^{m}$*, to a real value:* $E(\Psi(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}), \mathbf{w}_{sy}) \in \mathbb{R}$*. For simplicity we also express a NeSy energy function as* $E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{sy}) \in \mathbb{R}$*.*

NeSy-EBM models are differentiated from one another by the instantiation process, the form of the symbolic potentials, and the method for combining the symbolic potentials to define the energy function. One example of a NeSy-EBM is DeepProbLog (DPL) [Manhaeve et al., 2018]. DPL uses the output of a neural network to specify probabilities of events. The events are then used in logical formulae defining probabilistic dependencies and act as symbolic potentials. The DPL energy function is a joint distribution and inference is finding the highest-probability, i.e., lowest energy, state of the variables. Logic Tensor Networks (LTNs) [Badreddine et al., 2022] are another NeSy method that belongs to the NeSy-EBM family, and instantiates a model which forwards neural network predictions into functions representing symbolic relations with real-valued or fuzzy logic semantics. From the NeSy-EBM point of view, the fuzzy logic functions are the symbolic potentials and their composition defines the energy function.

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) probabilistic framework [Bach et al., 2017]. As illustrated in Figure 1, NeuPSL leverages the low-level perception of neural networks by integrating their outputs into a set of symbolic potentials created by a PSL program. 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 a variety of 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:

$$w : \text{NEURAL}(\texttt{Image}_1, \texttt{Species}) \wedge \text{SAMEENTITY}(\texttt{Image}_1, \texttt{Image}_2) \rightarrow \text{CLASS}(\texttt{Image}_2, \texttt{Species}) \quad (1)$$

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 the 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):

$$w : \text{NEURAL}(Id1, Cat) \wedge \text{SAMEENTITY}(Id1, Id2) \rightarrow \text{CLASS}(Id2, Cat)$$
$$w : \text{NEURAL}(Id2, Cat) \wedge \text{SAMEENTITY}(Id2, Id1) \rightarrow \text{CLASS}(Id1, Cat)$$

Ground atoms are mapped to either an observed variable, x_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 $\mathbf{x} = [x_i]_{i=1}^{n_x}$ and $\mathbf{y} = [y_i]_{i=1}^{n_y}$ and neural outputs are aggregated into the vector $\mathbf{g}_{nn} = [g_{nn,i}]_{i=1}^{n_g}$. Each ground rule creates one or more potentials defined over $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{g}_{nn}$. Logical rules, such as the one given in example (1), are relaxed using Łukasiewicz continuous valued logical semantics Klir and Yuan [1995]. NeuPSL also supports arithmetic rules for defining penalty functions. Each potential ϕ_i is associated with a parameter $w_{psl,i}$ that is 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 $\mathbf{y} = [y_i]_{i=1}^{n_y}$ and $\mathbf{x} = [x_i]_{i=1}^{n_x}$ be vectors of real 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*

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

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

$$E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = \sum_{i=1}^{R} \mathbf{w}_{psl}[i] \Phi_i(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}) = \mathbf{w}_{psl}^{T} \mathbf{\Phi}(\mathbf{y}, \mathbf{x}, \mathbf{x_{nn}}, \mathbf{w_{nn}}) \quad (3)$$

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

$$P(\mathbf{y}|\mathbf{x}, \mathbf{x}_{nn}) = \begin{cases} \frac{1}{Z(\mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})} \exp(-E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})) & (\mathbf{y}, \mathbf{x}) \in \mathbf{\Omega} \\ 0 & o.w. \end{cases} \quad (4)$$

$$Z(\mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = \int_{\mathbf{y}|\mathbf{y}, \mathbf{x} \in \mathbf{\Omega}} \exp(-E(\mathbf{y}, \mathbf{x}, \mathbf{x}_{nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})) d\mathbf{y} \quad (5)$$

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 $\mathbf{y}$:

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

Note that the energy function and constraints are 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 parameters 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 $\mathcal{S} = \{(\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}) : i = 1, \cdots, P\}$ to a real-valued loss. As the energy function for NeuPSL is parameterized by the neural parameters $\mathbf{w}_{nn}$ and symbolic parameters $\mathbf{w}_{psl}$, 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 $L_i(\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl})$. Concisely, NeuPSL learning is the following minimization:

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

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. The most common losses, including the loss we present in this work, use the following terms:

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

In words, $\mathbf{z}_i^*$ and $\mathbf{y}_i^*$ are the lowest energy states given $(\mathbf{y}_{i,t}, \mathbf{x}_i, \mathbf{x}_{i,nn})$ and $(\mathbf{x}, \mathbf{x}_{i,nn})$, 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 necessarily require the following partials derivatives: [2]

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

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:[2]

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

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

Since $l(\mathbf{y}, \mathbf{x}, \mathbf{g}_{nn}(\mathbf{x}_{nn}, \mathbf{w}_{nn}))$ 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:

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

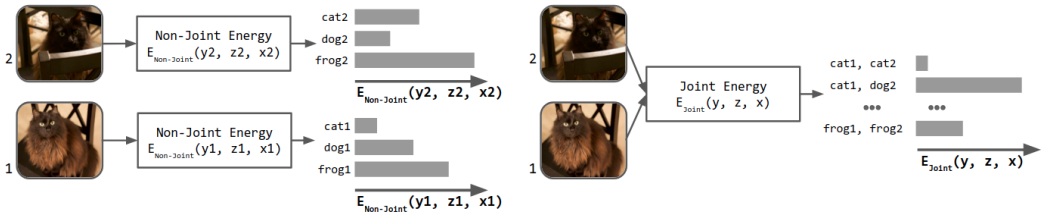
Notice that inference over the latent variables is necessary for gradient and objective value computations. However, a complete prediction from NeuPSL, i.e., inference over all components of $\mathbf{y}$, is not necessary. Therefore the parameter learning problem is as follows:

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

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 $\mathbf{y}_i, \mathbf{x}_i, \mathbf{x}_{i,nn} \mathbf{w}_{nn}$ is:

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

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



(a) Independent Energy

(b) Joint-Energy

Figure 2: Example of non-joint and joint energy functions.

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

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

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

6 Joint Reasoning in NeSy-EBMs

The EBM energy function encodes dependencies between potentially high dimensional input variables, $\mathbf{x}$, and output variables, $\mathbf{y}$. We highlight two important categories of energy functions: *joint* and *independent* energy functions. Formally, we refer to an energy function that is additively separable over the output variables $\mathbf{y}$ as an *independent energy function*, i.e., there exists functions $E_1(\mathbf{y}[1], \mathbf{x}), \cdots, E_{n_y}(\mathbf{y}[n_y], \mathbf{x})$ such that $E(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{n_y} E_i(\mathbf{y}[i], \mathbf{x})$. 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. In independent inference, 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. Independent energy functions speed up inference and learning, however, they cannot leverage joint information that may be used to improve predictions.

To illustrate, recall the example described in Section 4 where a neural network is used to classify the species of an animal in an image with external information. Figure 2 outlines the distinction between independent and joint prediction for this scenario. In Figure 2(a), the independent setting, the input is a single image, and the energy function is defined over the three possible classes: `dog`, `cat`, and `frog`. While in Figure 2(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 will see in the experiments 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).

7 Experimental Evaluation

We perform a series of experiments highlighting 1) the runtime of NeuPSL symbolic inference, 2) NeuPSL's ability to integrate neural and symbolic inference and learning, 3) the effectiveness of NeuPSL's joint inference and learning, and 4) NeuPSL's performance on Visual-Sudoku-Classification, a new NeSy task. NeuPSL is implemented using the open-source PSL software package and can be used with any neural network library (in this paper, we use TensorFlow). [3] In addition to NeuPSL,

[3] Implementation details including hyperparameters, network architectures, hardware, and NeuPSL symbolic models are described in the appendix materials.

we include results for DeepProbLog (DPL) Manhaeve et al. [2021a], Logic Tensor Networks (LTNs) [Badreddine et al., 2022], and convolutional neural network baselines (CNNs).

The first three experiments are performed over two canonical NeSy tasks: **MNIST-Add1** and **MNIST-Add2** [Manhaeve et al., 2018]. These tasks extend the classic MNIST image classification problem [LeCun et al., 1998] by constructing addition equations using MNIST images and requiring classification to be performed using only their sum as a label. For example, a **MNIST-Add1** addition is ($[3] + [5] = \mathbf{8}$), and a **MNIST-Add2** addition is ($[0, 4] + [3, 7] = \mathbf{41}$). Given n randomly selected MNIST images, we create $n/2$ **MNIST-Add1** or $n/4$ **MNIST-Add2** additions. We emphasize that individual MNIST images do not have labels, only the resulting sum. All results are averaged over ten splits constructed from a set of $10,000$ randomly sampled MNIST images.

7.1 Symbolic Inference and Runtime

We compare runtimes of a latent variable NeuPSL model, a constraint-based NeuPSL model, and a DPL model for both **MNIST-Add** tasks. Models are trained on additions constructed from $6,000$ MNIST images. Table 1 [4] shows the average inference times per addition and standard deviation across ten splits with the fastest approach bolded. We see the benefit of NeuPSL's convex symbolic inference as we consistently see a 2-3 times speedup for the fastest NeuPSL models. NeuPSL-Constraint requires significantly less runtime than NeuPSL-Latent for **MNIST-Add1**. This is because the instantiated latent model contains roughly twice the number of symbolic potentials and decision variables; hence, inference requires more time to converge. However, for **MNIST-Add2**, introducing additional latent variables results in a more concise set of symbolic potentials, and despite the latent model having more decision variables, the reduction in model size is enough to achieve significantly faster inference. For the remaining experiments, we report the results on NeuPSL-Constraint for **MNIST-Add1** and NeuPSL-Latent for **MNIST-Add2**. Note that inference in neural baselines and LTNs for **MNIST-Add** are equivalent to making a feed forward pass in a neural network, and therefore not comparable to the complex symbolic inference done in DPL and NeuPSL. This trivial inference is very fast, but comes with a decrease in predictive performance that we will examine next.

	DPL	NeuPSL-Constraint	NeuPSL-Latent
MNIST-Add1	1.34e-2 ± 1.67e-4	**4.00e-3 ± 1.00e-4**	2.98e-2 ± 2.33e-4
MNIST-Add2	1.65e+3 ± 4.80e+2	8.76e+3 ± —.—[4]	**7.38e+2 ± 3.36e+1**

Table 1: Inference times (milliseconds) on test sets constructed from $10,000$ MNIST images.

7.2 Deep Learning and Symbolic Reasoning

We evaluate the predictive performance of NeuPSL, DPL, LTNs, and neural baselines on **MNIST-Add** with varying amounts of training data. Table 2[5] shows the average accuracy and standard deviation across ten splits. 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 markedly lower variance for nearly all amounts of training examples in both **MNIST-Add** tasks.

7.3 Joint Inference and Learning

To quantify the benefit of joint learning and inference, we propose variants for **MNIST-Add** in which digits are re-used across multiple additions. Figure 3 illustrates the new information that joint models can leverage to narrow the space of possible labels when MNIST images are re-used. Since the same MNIST image of a zero appears in two addition samples, the value it takes must satisfy both, i.e., the possible label space can no longer include a two or three as it would violate the second addition.

[4] Timing results includes both model instantiation and inference. In the interest of time, NeuPSL-Constraint was only run once for **MNIST-Add2**.

[5] Exact inference results are reported for DPL in most settings as in Manhaeve et al. (2021a). However, exact inference in both $50,000$ **MNIST-Add** experiments produce random predictions (likely due to a NaN in DPL's loss). For these settings we report the results of DPL's approximate inference as described in Manhaeve et al. [2021b], which generally produces slightly lower accuracy with higher variance compared to exact inference.

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	74.08 ± 15.33[5]	**71.37 ± 03.90**	**87.44 ± 02.15**	92.23 ± 01.01[5]
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 2: Test set accuracy and standard deviation on **MNIST-Add**.

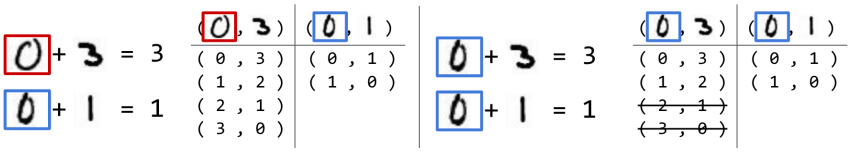


Figure 3: Example of overlapping MNIST images in **MNIST-Add1**. On the left, distinct images are used for each zero. On the right, the same image is used for both zeros.

To create overlap (reused MNIST images), we begin with a set of n unique MNIST images from which we re-sample to create a collection of $(n+m)/2$ **MNIST-Add1** and $(n+m)/4$ **MNIST-Add2** additions. We vary the amount of overlap with $m \in \{0, n/2, n, 2n\}$ and compare model performance in low data settings, $n \in \{40, 75, 150\}$. In low data settings there is not enough structure from the original additions for symbolic inference to discern the correct digit labels for training the neural models. Results are reported over test sets of $10,000$ MNIST images with overlap proportional to the respective train set.

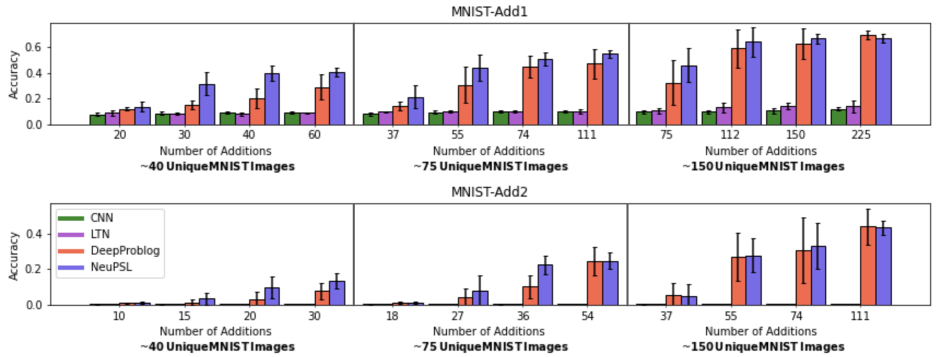


Figure 4: Average test set accuracy and standard deviation on **MNIST-Add** datasets with varying amounts of overlap.

Figure 4 shows the accuracy and standard deviation of all methods with varying amounts of overlap. We see that even though the number of unique MNIST images remains the same, as the number of additions increases, both DPL and NeuPSL are able to take advantage of the added joint information to improve their predictions. We see that in almost all cases, NeuPSL has the best performance. Furthermore, we again see NeuPSL has a lower standard deviation compared to DPL. LTNs and the CNN baseline benefit the least from joint information, likely this is a consequence of both learning and inference being performed independently across batches of additions.

7.4 Visual Sudoku Classification

Finally, we introduce **Visual-Sudoku-Classification**, a new NeSy task where 4x4 Sudoku puzzles are constructed using unlabeled MNIST images, and the task is to identify whether a puzzle is correct, i.e., has no duplicate digits in any row, column, or square. Unlike Wang et al. [2019], this task does not require learning the underlying label for images but rather whether an entire puzzle is valid. For instance, [3] does not need to belong to a "3" class, instead [3] and [4] need to be labeled as different. In this setting, n MNIST images creates $n/16$ **Visual-Sudoku-Classification** puzzles.

Method	Number of Puzzles			
	10	20	100	200
CNN-Visual	51.09 ± 15.78	51.36 ± 03.93	50.09 ± 02.17	51.23 ± 02.20
CNN-Digit	55.45 ± 05.22	54.55 ± 07.57	71.09 ± 04.99	82.22 ± 01.77
NeuPSL	**64.54 ± 12.93**	**72.27 ± 06.84**	**84.72 ± 03.61**	**85.05 ± 02.65**

Table 3: Test set accuracy and standard deviation on **Visual-Sudoku-Classification**.

We compare NeuPSL with two CNN baselines, CNN-Visual and CNN-Digit. CNN-Visual takes as input the pixels for a Sudoku puzzle and outputs the probability of the puzzle being valid. To verify whether the neural model is able to learn the Sudoku rules, we additionally introduce the CNN-Digit baseline that also outputs a probability of the puzzle being valid, but is provided all sixteen digit labels. Table 3 shows the test set accuracy and standard deviation of results with the best performing method in bold. We see that NeuPSL consistently outperforms both neural baselines. Furthermore, even when CNN-Digit is provided as input the sixteen digit labels, NeuPSL is able to outperform this baseline using half the amount of puzzles.

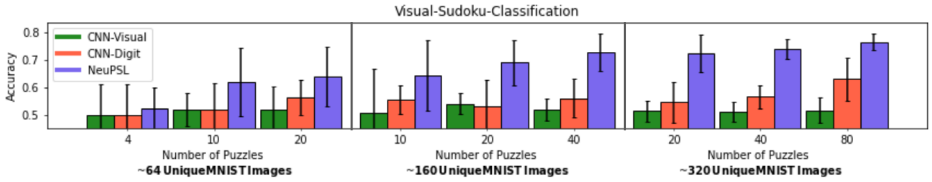


Figure 5: Average test set accuracy and standard deviation on **Visual-Sudoku-Classification** dataset with varying amounts of overlap.

Furthermore, we evaluate performance after adding relational structure in a similar fashion to the **MNIST-Add** variations. Figure 5 shows the accuracy and standard deviation of NeuPSL and CNN models on **Visual-Sudoku-Classification** with varying amounts of overlap. NeuPSL is efficient at leveraging the joint information across training examples, and is able to create a ~65% puzzle classifier using only about 64 MNIST images across 20 puzzles. This is a particularly impressive result as this performance implies the neural network in the NeuPSL model was trained to be a ~89% 4-digit distinguisher without any digit labels. Additionally, we observe that CNN-Visual is unable to generalize even in the highest settings (3200 MNIST images across 200 puzzles).

8 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 exploration of 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.

Acknowledgements

This work was partially supported by the National Science Foundation grants CCF-1740850 and CCF-2023495 and an unrestricted gift from Google.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent

Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

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

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. In *We Will Show Them! Essays in Honour of Dov Gabbay, Volume One*, 2005.

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

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*, 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 S. 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 *FAccTRec*, 2020.

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

Ivan Donadello, Luciano Serafini, and Artur 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. *IEEE Transactions on Knowledge and Data Engineering*, 31(8):1426–1439, 2019.

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 S. 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.

Longfei Li, Yong Zhao, Dongmei Jiang, Yanning Zhang, Fengna Wang, Isabel Gonzalez, Enescu Valentin, and Hichem Sahli. Hybrid deep neural network–hidden markov model (dnn-hmm) based speech emotion recognition. In *ACII*. IEEE, 2013.

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. *Artificial Intelligence*, 298:103504, 2021a.

Robin Manhaeve, Giuseppe Marra, and Luc De Raedt. Approximate inference for neural probabilistic logic programming. In *(KR)*, 2021b.

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.

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

Sebastian Nowozin and Christoph H Lampert. *Structured Learning and Prediction in Computer Vision*. Now publishers Inc, 2011.

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.

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

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. *arXiv*, abs/2106.12574, 2021.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

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.

Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015.

Appendix

The appendix include the following sections: Additional NeuPSL Optimization Details (Appendix A), Dataset Details (Appendix B), NeuPSL Models (Appendix C), Baseline Neural Models (Appendix D), Extended Evaluation Details (Appendix E), and Computational Hardware Details (Appendix F).

A Additional NeuPSL Optimization Details

This section expands the discussion of the learning algorithm (Section 5) with algorithmic details and discussion of degenerate solutions of NeuPSL neural and symbolic parameter learning with the energy loss. We begin by introducing a simplex constraint on the NeuPSL symbolic parameters. The simplex constraint is used to prevent the first degenerate solution we show exists, when the weights are all zero. We then show there are additional degenerate solutions of the simplex constrained problem at corners of the simplex. In order to address these degenerate solutions, we regularize the energy loss objective with a negative log function. Finally, we provide the learning algorithm used throughout all the experiments, projected gradient descent.

A.1 Simplex Constrained Symbolic Parameters

The NeuPSL learning problem constrains the symbolic parameters to the unit simplex. Srinivasan et al. (2021) showed that MAP inference in HL-MRFs is invariant to the scale of the weights. Formally, for all weight configurations $\mathbf{w}_{psl}$ and scalars $\tilde{c} \in \mathbb{R}_+$,

$$\underset{\mathbf{y}|(\mathbf{y},\mathbf{x})\in\mathbf{\Omega}}{\arg\max}\ E(\mathbf{y},\mathbf{x},\mathbf{x}_{nn},\mathbf{w}_{nn},\mathbf{w}_{psl}) = \underset{\mathbf{y}|(\mathbf{y},\mathbf{x})\in\mathbf{\Omega}}{\arg\max}\ E(\mathbf{y},\mathbf{x},\mathbf{x}_{nn},\mathbf{w}_{nn},\tilde{c}\cdot\mathbf{w}_{psl}) \tag{7}$$

For this reason, it is possible to constrain the search space of the symbolic parameters to the unit simplex, $\Delta^r = \{\mathbf{w} \in \mathbb{R}^r_+ \big| \|\mathbf{w}\|_1 = 1\}$, without inhibiting the expressivity of the model when the HL-MRF is exclusively used to obtain MAP inference predictions.

A.2 Energy Loss Degenerate Solutions

We now show two degenerate solutions of energy learning for NeuPSL and method for overcoming them. Recall that the energy loss learning problem presented in this work for end-to-end training of the symbolic and neural parameters of a NeuPSL model is:

$$\underset{(\mathbf{w}_{nn},\mathbf{w}_{psl})\in\mathbb{R}^{nn}\times\mathbb{R}^r_+}{\arg\min}\ \mathcal{L}(\mathbf{w}_{nn},\mathbf{w}_{psl},\mathcal{S}) \tag{8}$$

$$= \underset{(\mathbf{w}_{nn},\mathbf{w}_{psl})\in\mathbb{R}^{nn}\times\mathbb{R}^r_+}{\arg\min} \sum_{i=1}^{P} E((\mathbf{y}_{i,t},\mathbf{z}^*_i),\mathbf{x}_i,\mathbf{x}_{i,nn},\mathbf{w}_{nn},\mathbf{w}_{psl}) \tag{9}$$

$$= \underset{(\mathbf{w}_{nn},\mathbf{w}_{psl})\in\mathbb{R}^{nn}\times\mathbb{R}^r_+}{\arg\min} \sum_{i=1}^{P} \min_{\mathbf{z}|((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x})\in\mathbf{\Omega}} \mathbf{w}_{psl}^T\mathbf{\Phi}((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x}_i,\mathbf{x}_{i,nn},\mathbf{w}_{nn}) \tag{10}$$

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

$$\phi_i(\mathbf{y},\mathbf{x},\mathbf{x}_{nn},\mathbf{w}_{nn}) = \max(l_i(\mathbf{y},\mathbf{x},\mathbf{g}_{nn}(\mathbf{x}_{nn},\mathbf{w}_{nn})),0)^\alpha \tag{11}$$

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

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

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. Adding the simplex constraint discussed in the previous section, Section A.1, $\mathbf{w}_{psl} \in \Delta^r$, makes the degenerate solution $\mathbf{w}_{psl} = \mathbf{0}$ infeasible. This constraint also ensures non-negativity of the parameters, and does not inhibit the expressivity of NeuPSL as a MAP inference predictor.

An additional degenerate solution arises from the introduction of the simplex constraint in conjunction with the fact that the energy loss is concave in the symbolic parameters $\mathbf{w}_{psl}$ for fixed $\mathbf{w}_{nn}$ and $\mathcal{S}$. Precisely, a solution to the problem must exist at corner points of the simplex Δ^r.

Lemma 1. *The energy loss function*

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

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

Proof. For all i

$$E((\mathbf{y}_{i,t}, \mathbf{z}_i^*), \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) = \inf_{\mathbf{z}|((\mathbf{y}_{i,t},\mathbf{z}),\mathbf{x})\in\mathbf{\Omega}} \mathbf{w}_{psl}^T \Phi((\mathbf{y}_{i,t}, \mathbf{z}), \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}) \tag{12}$$

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,

$$\mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{psl}, \mathcal{S}) = \sum_{i=1}^{P} E((\mathbf{y}_{i,t}, \mathbf{z}_i^*), \mathbf{x}_i, \mathbf{x}_{i,nn}, \mathbf{w}_{nn}, \mathbf{w}_{psl}) \tag{13}$$

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

Further, Δ^r is a polyhedron described by the set.

$$\Delta^r = \{\mathbf{w}_{psl} \quad | \quad \mathbf{1}^T \mathbf{w}_{psl} = 1\} \tag{14}$$

The unit simplex, Δ^r, is therefore a convex set. A concave function is globally minimized over a polyhedron at one of the vertices, this can be shown by definition of concavity. In this case, we can find a solution to the energy minimization problem by comparing the objective value at each point of the simplex, i.e., setting one of the symbolic parameter components to 1 and the remaining parameters to 0. This solution is however undesirable; we want each of the symbolic relations corresponding to the parameters to be represented and have influence over the model predictions. For this reason we propose the use of the negative logarithm as a regularizer to break the concavity of the objective and give the simplex corner solutions infinitely high energy. With negative log regularization and simplex constraints, energy loss symbolic parameter learning is:

$$\min_{\mathbf{w}_{nn}\in\mathcal{W}_{nn}, \mathbf{w}_{psl}\in\Delta^r} \mathcal{L}(\mathbf{w}_{nn}, \mathbf{w}_{psl}, \mathcal{S}) - \sum_{i=1}^{r} \log_b(\mathbf{w}_{psl}[i]) \tag{15}$$

A.3 Projected Gradient Descent

Minimizing the regularized and constrained energy loss problem introduced in Section 5 is possible with a variety of algorithms. For simplicity we use projected gradient descent with a decaying learning rate, γ.

Algorithm 1: Projected Gradient Descent Algorithm

Input: $E(\cdot), \mathcal{S}, \gamma^0$

$k \leftarrow 0$;

repeat

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

  $\mathbf{w}_{psl}^{k+1} \leftarrow \Pi_{\Delta^r} \left[\mathbf{w}_{psl}^k + \gamma \nabla_{\mathbf{w}_{psl}} \mathcal{L}(\mathbf{w}_{nn}^k, \mathbf{w}_{psl}^k, \mathcal{S})\right]$

  $\gamma^{k+1} \leftarrow \gamma^k/(k+1)$;

  $k \leftarrow k+1$;

until *stopping criteria satisfied*;

B Datasets

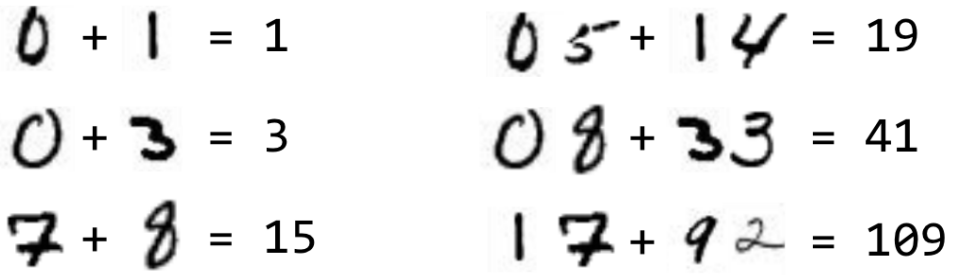
In this section we provide additional information on MNIST-Add Manhaeve et al. [2018] and Visual-Sudoku-Classification tasks. Both tasks make use of the MNIST image classification dataset introduced in LeCun et al. [1998]. Each MNIST image is a 28x28 matrix consisting of pixel gray scale values normalized to lie in the range $[0, 1]$.

B.1 MNIST-Add

The MNIST-Add dataset, 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.

The original dataset consists of three training sizes, $n = \{600, 6000, 50000\}$, and a single test size, $n = 10,000$, where n represents the number of unique MNIST images. These images are broken into $n/(2 * k)$ addition examples, e.g., the training sizes MNIST-Add1 creates $\{300, 3,000, 25,000\}$ additions and MNIST-Add2 creates $\{150, 1,500, 12,500\}$ additions. The addition examples are created by first shuffling the list of MNIST images, and then partitioning, in order, pairs of numbers. For example, let the training size $n = 6$ with the corresponding list of MNIST images, [ ,  ,  ,  ,  , ]. First this list is shuffled, [ ,  ,  ,  ,  , ], and then partitioned into $2 * k$ tuples in order. In this scenario, MNIST-Add1 creates 3 addition examples, $\Big[$[ , ], [, ], [ ,]$\Big]$.

We follow this same data generation process, and repeat it for 10 splits. We maintain the same train sizes $n = \{600, 6,000, 50,000\}$ and test sizes $n = 10,000$. Each train and test split is randomly sampled without replacement from the complete train partition of the MNIST dataset provided by TensorFlow Abadi et al. [2015].



(a) MNIST-Add1 (b) MNIST-Add2

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

B.1.1 Overlap

As discussed in Section 7.3, we create variations of the MNIST-Add datasets that introduces overlap across examples in the MNIST-Add datasets. For this experiment we focus on the train set sizes $n \in \{40, 75, 150\}$. In low data settings, leveraging overlap is especially important because there is not enough structure from the original additions for symbolic inference to discern the correct digit labels for training the neural models.

The overlap variation of the dataset augments the original list of MNIST images with duplicate images to create more addition examples. Precisely, as for the standard MNIST-Add task, for every split and train size, n images are randomly sampled without replacement. Then, $m \in \{0, n/2, n, 2n\}$ images are randomly sampled with replacement MNIST images from the n unique MNIST images. These two collections are joined to make MNIST image collections of $n + m$. Finally, these lists are shuffled and then used to generate MNIST-Add examples using the process described in the previous section.

B.2 Visual-Sudoku-Classification

Inspired by Wang et al. [2019], the Visual-Sudoku-Classification task uses MNIST images to create 4x4 Sudoku puzzles as shown in Figure 7. MNIST images, without labels, are used instead of digits to represent each cell in the puzzle. However, unlike Wang et al. [2019] in which MNIST zeros are used to represent blank squares, and the task is to solve the puzzle, the Visual-Sudoku-Classification task is to identify whether a puzzle is correct, i.e., there are no duplicate digits in any row, column, or square.

Similar to MNIST-Add, puzzles are created using $n \in \{160, 320, 1, 600, 3, 200\}$ MNIST images for the training set and $n \in \{320, 640, 3, 200, 6, 400\}$ MNIST images for the test set. These images are broken into $n/16$ puzzle examples. Precisely, MNIST images are first randomly sampled over each class in equal proportions. In other words, an equal number of zeros, ones, twos, and threes are randomly sampled from the original MNIST dataset. Then correct puzzles are created by sampling 4 unique images of each class and putting them into a valid puzzle configuration. A valid puzzle configuration is a single tuple where the first MNIST image represents the top left corner of the puzzle and the last is the bottom right corner. For example, Figure 7 would be $\{1, 2, 4, 3, 4, 3, 1, 2, 2, 4, 3, 1, 3, 1, 2, 4\}$.

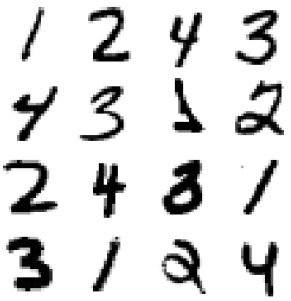


Figure 7: An example of a valid Visual-Sudoku-Classification puzzle.

B.2.1 Puzzle Corruption

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 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. For each puzzle, a fair coin is flipped 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 corruption process is complete, 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.

B.2.2 Overlap

As discussed in Section 7.4, we create variations that introduce overlap across examples in the Visual-Sudoku-Classification task. Overlap augments the original list of MNIST images by adding duplications. We decrease the train set sizes to be $n \in \{64, 160, 320\}$. As with MNIST-Add, overlap has a greater impact in lower data settings. For every split and train size, n MNIST images are uniformly sampled over each class, e.g., an equal number of zeros, ones, twos, and threes are

randomly sampled. Before the process of generating Visual-Sudoku-Classification examples, we add $m \in \{0, n, 3n\}$ randomly sampled with replacement MNIST images from the n unique MNIST images. These two sets are joined to make train and test MNIST image sizes of $n + m$. Finally, these lists are shuffled and then used to generate Visual-Sudoku-Classification examples as per the process described above.

C NeuPSL Models

In this section we provide details on the NeuPSL neural and symbolic models for both the MNIST-Add and Visual-Sudoku-Classification experiments.

C.1 NeuPSL Neural Model

All NeuPSL models use the following neural model to predict image labels for their respective tasks. This is the same neural architecture used for the digit label predictor in Manhaeve et al. [2018]. Additional details on the hyperparameter search for this model are given for each NeuPSL model.

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 4: The architecture of the CNN network used in NeuPSL for both MNIST-Add and Visual-Sudoku-Classification experiments.

C.2 MNIST-Add

There are two groups of MNIST-Add models we evaluate in this work: 1) *NeuPSL-Constraint* models enforce constraints directly on the output of the neural model with no latent variables in either inference or learning and 2) *NeuPSL-Latent* models enforce constraints directly on the output of the neural model with latent variables representing either a digit label for an image or the sum of two digits. Both versions of the model are presented in the following subsections and are different for the MNIST-Add1 and MNIST-Add2 tasks. The MNIST-Add1 and MNIST-Add2 NeuPSL models both contain the follow two 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 DIGIT predicate takes a 0 or 1 value representing whether the sum of two digits `X` and `Y` adds up to `Z`. This predicate only instantiates observations, i.e., variables from this predicate are fixed during inference and learning as described in Section 3, 4, and 5.

C.2.1 MNIST-Add1

In addition, both NeuPSL-Constraint and NeuPSL-Latent models for MNIST-Add1 contain the following two predicates:

SUM$(\texttt{Img1}, \texttt{Img2}, \texttt{Z})$ 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 Section 3, 4, and 5.

POSSIBLEDIGITS$(\texttt{X}, \texttt{Z})$ The POSSIBLEDIGITS predicate represents whether the digit identified, `X`, is possible when it is in a sum that totals to the number identified, `Z`. For instance POSSIBLEDIGITS$(9, 0) = 0$ because no positive digit added to 9 equals 0, while POSSIBLEDIGITS$(9, 17) = 1$ because 8 added to 9 is 17.

NeuPSL-Constraint
The NeuPSL-Constraint model expresses domain constraints over the NEURAL$(\texttt{Img}, \texttt{X})$ predicate.

 # Digit Sums
w_1 : NEURAL$(\texttt{Img1}, \texttt{X}) \wedge$ NEURAL$(\texttt{Img2}, \texttt{Y}) \wedge$ DIGITSUM$(\texttt{X}, \texttt{Y}, \texttt{Z}) \rightarrow$ SUM$(\texttt{Img1}, \texttt{Img2}, \texttt{Z})$
w_2 : $\neg$NEURAL$(\texttt{Img1}, \texttt{X}) \wedge$ NEURAL$(\texttt{Img2}, \texttt{Y}) \wedge$ DIGITSUM$(\texttt{X}, \texttt{Y}, \texttt{Z}) \rightarrow \neg$SUM$(\texttt{Img1}, \texttt{Img2}, \texttt{Z})$
w_3 : NEURAL$(\texttt{Img1}, \texttt{X}) \wedge \neg$NEURAL$(\texttt{Img2}, \texttt{Y}) \wedge$ DIGITSUM$(\texttt{X}, \texttt{Y}, \texttt{Z}) \rightarrow \neg$SUM$(\texttt{Img1}, \texttt{Img2}, \texttt{Z})$

 # Digit Constraints
w_4 : NEURAL$(\texttt{Img1}, +\texttt{X}) >=$ SUM$(\texttt{Img1}, \texttt{Img2}, \texttt{Z})\{\texttt{X} :$ POSSIBLEDIGITS$(\texttt{X}, \texttt{Z})\}$
w_5 : NEURAL$(\texttt{Img2}, +\texttt{X}) >=$ SUM$(\texttt{Img1}, \texttt{Img2}, \texttt{Z})\{\texttt{X} :$ POSSIBLEDIGITS$(\texttt{X}, \texttt{Z})\}$

 # Simplex Constraints
SUM$(\texttt{Img1}, \texttt{Img2}, +\texttt{Z}) = 1.$

Figure 8: NeuPSL-Constraint MNIST-Add1 Model

The group of rules labeled *Digit Sums* 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 group of rules labeled *Digit Constraints* 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.

NeuPSL-Latent
NeuPSL-Latent model for MNIST-Add1 includes the same domain constraints and relations as the NeuPSL-Constraint model for MNIST-Add1 but also expresses them over latent variables instantiated by the following predicate:

DIGIT$(\texttt{Img}, \texttt{X})$ The DIGIT predicate has the same arguments as the NEURAL predicate, and represents the probability the image identified by the argument `Img` has the digit label `X`.

 # Neural and Latent Variable Proximity
w_1 : NEURAL(Img, X) = DIGIT(Img, X)

 # Digit Sums
DIGIT(Img1, X) ∧ DIGIT(Img2, Y) ∧ DIGITSUM(X, Y, Z) → SUM(Img1, Img2, Z)
¬DIGIT(Img1, X) ∧ DIGIT(Img2, Y) ∧ DIGITSUM(X, Y, Z) → ¬SUM(Img1, Img2, Z)
DIGIT(Img1, X) ∧ ¬DIGIT(Img2, Y) ∧ DIGITSUM(X, Y, Z) → ¬SUM(Img1, Img2, Z)

 # Digit Constraints
DIGIT(Img1, +X) >= SUM(Img1, Img2, Z){X : POSSIBLEDIGITS(X, Z)}
DIGIT(Img2, +X) >= SUM(Img1, Img2, Z){X : POSSIBLEDIGITS(X, Z)}

 # Digit Sums
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)
w_4 : NEURAL(Img1, X) ∧ ¬NEURAL(Img2, Y) ∧ DIGITSUM(X, Y, Z) → ¬SUM(Img1, Img2, Z)

 # Digit Constraints
w_5 : NEURAL(Img1, +X) >= SUM(Img1, Img2, Z){X : POSSIBLEDIGITS(X, Z)}
w_6 : NEURAL(Img2, +X) >= SUM(Img1, Img2, Z){X : POSSIBLEDIGITS(X, Z)}

 # Simplex Constraints
SUM(Img1, Img2, +Z) = 1.
DIGIT(ImgId, +X) = 1.

Figure 9: NeuPSL-Latent MNIST-Add1 Model

The *Neural and Latent Variable Proximity* rule is introduced to increase the energy of solutions with conflicting NEURAL(Img, X) and DIGIT(Img, X) predictions.

The rule groups from the MNIST-Add1-Constraint model are added for the DIGIT(Img, X) predicates. However, they are not weighted, and un-weighted rules create hard constraints rather than potentials. Since DIGIT(Img, X) instantiates latent variables, a feasible solution satisfying all hard constraints is possible. A feasible DIGIT(Img, X) solution is critical during energy loss parameter learning as a subproblem is finding an energy minimizing state of the latent variables given the true values of the SUM(Img1, Img2, +Z) variables. In other words, a setting of the DIGIT(Img, X) variables is found that satisfies all of the constraints expressed in the model when the true values of SUM(Img1, Img2, +Z) are known. The low energy state of DIGIT(Img, X) interacts with neural parameter learning via the *Neural and Latent Variable Proximity* rule. The gradient the neural model backpropagates to fit its parameters during learning will push the neural predictions to be closer to the low energy and feasible DIGIT(Img, X) variables in order to satisfy the constraints.

The NeuPSL-Constraint model rules are duplicated for the latent DIGIT(Img, X) variables which results in the NeuPSL-Latent model having roughly twice as many instantiated potentials. Furthermore, the DIGIT(Img, X) variables are additional decision variables in the optimization problem for inference that the NeuPSL-Latent model must solve.

MNIST-Add1 Hyperparameters
Table 5 shows the hyperparameter values and the tuning ranges of the NeuPSL MNIST-Add1 models. The *Neural Learning Rate* is the learning rate of the neural model used to predict image labels and the *ADMM Max Iterations* parameter is the number of ADMM iterations performed between each step of gradient descent during learning. The hyperparameter search was performed on the MNIST-Add1 300 additions setting and the best parameters were used for the other data settings. All unspecified values were left at their default values.

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 5: Hyperparameters searched over and used for the MNIST-Add1 experiments

C.2.2 MNIST-Add2

The NeuPSL-Constraint and NeuPSL-Latent NeuPSL models for MNIST-Add2 contain the follow three predicates in common:

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 Section 3, 4, and 5. Note that this is similar, however, not the same predicate as the analogous SUM in the NeuPSL MNIST-Add1 models. In this predicate there are four Img arguments that represent the sum of two digit numbers. The Img1 and Img3 arguments identify the images for the digits in the tens place, and Img2 and Img4 identify images in the ones place.

POSSIBLETENDIGITS(X, Z) The POSSIBLETENDIGITS predicate takes 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 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) The POSSIBLEONESDIGITS predicate takes 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 the number identified by the argument Z. For instance 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 POSSIBLEONESDIGITS$(9, 170) = 1$ as 71 added to 99 is 170.

NeuPSL-Constraint
The NeuPSL-Constraint model uses a similar modeling pattern to the MNIST-Add1 NeuPSL-Constraint model. Specifically, the MNIST-Add2-Constraint model enforces domain constraints over the NEURAL(Img, X) predicate and the summation relation is expressed using the following predicate:

NUMBERSUM(X10, X1, Y10, Y1, Z) The NUMBERSUM predicate takes a 0 or 1 value representing whether the sum of the two numbers identified by arguments (X10, X1) and (Y10, Y1) adds up to the number identified by the argument Z.

The *Number Sums* rules are analogous to the *Digit Sums* rules in the MNIST-Add1-Constraint model and represent the summation of the two collections of images (Img1, Img2) and (Img3, Img4). Specifically, if the neural model labels the images Img1 as X10, Img2 as X1, Img3 as Y10, and Img4 as Y1 and the numbers (X10, X1) and (Y10, Y1) sum to Z, then the sum of the images must be Z.

The *Tens Digit Constraints* 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 Constraints* 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.

Notice that every instantiation of NUMBERSUM(X10, X1, Y10, Y1, Z) that is non-zero in the *Number Sums* rules will create a potential that is non-trivial, i.e., the potential can be dis-satisfied depending on the values of the other predicates in the rule. Thus, since there are 10 possible classes for each of X10, X1, Y10, Y1 we have 10^4 instantiations of each rule in the *Number Sums* group and each addition. This modelling pattern does not scale to larger MNIST-Add settings; in general, for MNIST-Add(n) the analogous NUMBERSUM predicate will instantiate $10^{2 \cdot n}$ potentials for each rule in the *Number Sums* group.

NeuPSL-Latent
The NeuPSL model for MNIST-Add2 includes similar constraints and relations described in the NeuPSL2-Constraint MNIST-Add model, but uses latent variables to capture dependencies and

 # Number Sums

w_1 : NEURAL(Img1, X10) ∧ NEURAL(Img2, X1) ∧ NEURAL(Img3, Y10) ∧ NEURAL(Img4, Y1)
∧ NUMBERSUM(X10, X1, Y10, Y1, Z) → SUM(Img1, Img2, Img3, Img4, Z)

w_2 : ¬NEURAL(Img1, X10) ∧ NEURAL(Img2, X1) ∧ NEURAL(Img3, Y10) ∧ NEURAL(Img4, Y1)
∧ NUMBERSUM(X10, X1, Y10, Y1, Z) → ¬SUM(Img1, Img2, Img3, Img4, Z)

w_3 : NEURAL(Img1, X10) ∧ ¬NEURAL(Img2, X1) ∧ NEURAL(Img3, Y10) ∧ NEURAL(Img4, Y1)
∧ NUMBERSUM(X10, X1, Y10, Y1, Z) → ¬SUM(Img1, Img2, Img3, Img4, Z)

w_4 : NEURAL(Img1, X10) ∧ NEURAL(Img2, X1) ∧ ¬NEURAL(Img3, Y10) ∧ NEURAL(Img4, Y1)
∧ NUMBERSUM(X10, X1, Y10, Y1, Z) → ¬SUM(Img1, Img2, Img3, Img4, Z)

w_5 : NEURAL(Img1, X10) ∧ NEURAL(Img2, X1) ∧ NEURAL(Img3, Y10) ∧ ¬NEURAL(Img4, Y1)
∧ NUMBERSUM(X10, X1, Y10, Y1, Z) → ¬SUM(Img1, Img2, Img3, Img4, Z)

 # Tens Digit Constraints

w_6 : NEURAL(Img1, +X) >= SUM(Img1, Img2, Img3, Img4, Z){X : POSSIBLETENSDIGITS(X, Z)}

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

 # Ones Digit Constraints

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

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

 # Simplex Constraints

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

Figure 10: NeuPSL-Constraint MNIST-Add2 Model

relations in a more concise manner. NeuPSL-Latent model for MNIST-Add introduces the following predicates:

IMAGEDIGITSUM(Img1, Img2, Z) The IMAGEDIGITSUM predicate is the probability that the digits represented in the images identified by arguments Img1 and Img2 adds up to the number identified by the argument Z. The variables instantiated by this predicate are latent in the NeuPSL model because in neither learning nor inference do we have the true value of the sum of the digits represented by the images in the ones or tens places of the two numbers in the addition.

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 PLACENUMBERSUM$(1, 15, 25)$ is 1 as $1 \cdot 10 + 15 = 25$.

 # Tens Digit Sums

w_1 : NEURAL(Img1, X) $\wedge$ NEURAL(Img3, Y) $\wedge$ DIGITSUM(X, Y, Z) $\rightarrow$ IMAGEDIGITSUM(Img1, Img3, Z)

w_2 : $\neg$NEURAL(Img1, X) $\wedge$ NEURAL(Img3, Y) $\wedge$ DIGITSUM(X, Y, Z) $\rightarrow$ $\neg$IMAGEDIGITSUM(Img1, Img3, Z)

w_3 : NEURAL(Img1, X) $\wedge$ $\neg$NEURAL(Img3, Y) $\wedge$ DIGITSUM(X, Y, Z) $\rightarrow$ $\neg$IMAGEDIGITSUM(Img1, Img3, Z)

 # Ones Digit Sums

w_4 : NEURAL(Img2, X) $\wedge$ NEURAL(Img4, Y) $\wedge$ DIGITSUM(X, Y, Z) $\rightarrow$ IMAGEDIGITSUM(Img2, Img4, Z)

w_5 : $\neg$NEURAL(Img2, X) $\wedge$ NEURAL(Img4, Y) $\wedge$ DIGITSUM(X, Y, Z) $\rightarrow$ $\neg$IMAGEDIGITSUM(Img2, Img4, Z)

w_6 : NEURAL(Img2, X) $\wedge$ $\neg$NEURAL(Img4, Y) $\wedge$ DIGITSUM(X, Y, Z) $\rightarrow$ $\neg$IMAGEDIGITSUM(Img2, Img4, Z)

 # Place Digit Sums

IMAGEDIGITSUM(Img1, Img3, Z10) $\wedge$ IMAGEDIGITSUM(Img2, Img4, Z1) $\wedge$ PLACENUMBERSUM(Z10, Z1, Z)
$\rightarrow$ SUM(Img1, Img2, Img3, Img4, Z)

$\neg$IMAGEDIGITSUM(Img1, Img3, Z10) $\wedge$ IMAGEDIGITSUM(Img2, Img4, Z1) $\wedge$ PLACENUMBERSUM(Z10, Z1, Z)
$\rightarrow$ $\neg$SUM(Img1, Img2, Img3, Img4, Z)

IMAGEDIGITSUM(Img1, Img3, Z10) $\wedge$ $\neg$IMAGEDIGITSUM(Img2, Img4, Z1) $\wedge$ PLACENUMBERSUM(Z10, Z1, Z)
$\rightarrow$ $\neg$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 11: NeuPSL-Latent MNIST-Add2 Model

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-Constraint 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 Constraints* and *Ones Digit Constraints* are the same as those in the MNIST-Add2-Constraint model.

The *Number Sum Constraints* 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.

The MNIST-Add2-Latent model does not suffer from the same scalability issue as the MNIST-Add2-Constraint model. The latent IMAGEDIGITSUM variables allow us to express the summation relations via DIGITSUM which has fewer arguments than NUMBERSUM predicates. This is possible because the summation of numbers is performed by breaking the summation into placed digit sums, for instance $07 + 32 = 10 \cdot (0 + 3) + 1 \cdot (7 + 2) = 39$.

MNIST-Add2 Hyperparameters
Table 6 shows the hyperparameter values and the tuning ranges of the NeuPSL MNIST-Add2 models. The *Neural Learning Rate* is the learning rate of the neural model used to predict image labels and the *ADMM Max Iterations* parameter is the number of ADMM iterations performed between each step of gradient descent during learning. The hyperparameter search was performed on the MNIST-Add2 150 additions setting and the best parameters were used for the other data settings. All unspecified values were left at their default values.

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 6: Hyperparameters searched over and used for the MNIST-Add1 experiments.

C.3 Visual-Sudoku-Classification

The Visual-Sudoku-Classification PSL model contains three predicates:

NEURAL(`Puzzle`, `X`, `Y`, `Number`) The NEURAL predicate contains the output class probability for each digit image as 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, and represents PSL's digit prediction on the image.

FIRSTPUZZLE(`Puzzle`) The FIRSTPUZZLE predicate denotes the first correct puzzle in the training set.

 # L2 Loss

w_1 : NEURAL(Puzzle, X, Y, Number) = DIGIT(Puzzle, X, Y, Number)

 # Row Constraint

NEURAL(Puzzle, +X, Y, Number) = 1.

 # Column Constraint

NEURAL(Puzzle, X, +Y, Number) = 1.

 # Block Constraint

NEURAL(Puzzle, "0", "0", Number) + NEURAL(Puzzle, "0", "1", Number)
+ NEURAL(Puzzle, "1", "0", Number) + NEURAL(Puzzle, "1", "1", Number) = 1.
NEURAL(Puzzle, "2", "0", Number) + NEURAL(Puzzle, "2", "1", Number)
+ NEURAL(Puzzle, "3", "0", Number) + NEURAL(Puzzle, "3", "1", Number) = 1.
NEURAL(Puzzle, "0", "2", Number) + NEURAL(Puzzle, "0", "3", Number)
+ NEURAL(Puzzle, "1", "2", Number) + NEURAL(Puzzle, "1", "3", Number) = 1.
NEURAL(Puzzle, "2", "2", Number) + NEURAL(Puzzle, "2", "3", Number)
+ NEURAL(Puzzle, "3", "2", Number) + NEURAL(Puzzle, "3", "3", Number) = 1.

 # Simplex Constraints

NEURAL(Puzzle, X, X, +Number) = 1.

 # Pin First Row

w_2 : FIRSTPUZZLE(Puzzle) + NEURAL(Puzzle, "0", "0", "1") = 1
w_3 : FIRSTPUZZLE(Puzzle) + NEURAL(Puzzle, "0", "1", "2") = 1
w_4 : FIRSTPUZZLE(Puzzle) + NEURAL(Puzzle, "0", "2", "3") = 1
w_5 : FIRSTPUZZLE(Puzzle) + NEURAL(Puzzle, "0", "3", "4") = 1

 # Hint Second Row

w_6 : FIRSTPUZZLE(Puzzle) + NEURAL(Puzzle, "1", "0", "3") + NEURAL(Puzzle, "1", "0", "4") = 1
w_7 : FIRSTPUZZLE(Puzzle) + NEURAL(Puzzle, "1", "1", "3") + NEURAL(Puzzle, "1", "1", "4") = 1
w_8 : FIRSTPUZZLE(Puzzle) + NEURAL(Puzzle, "1", "2", "1") + NEURAL(Puzzle, "1", "2", "1") = 1
w_9 : FIRSTPUZZLE(Puzzle) + NEURAL(Puzzle, "1", "3", "1") + NEURAL(Puzzle, "1", "3", "2") = 1

Figure 12: NeuPSL Visual-Sudoku-Classification Model

The Visual-Sudoku-Classification model begins by encoding the row, column, and block rules of Sudoku into constraints. Each of these constraints restrict multiple instances of a digit from appearing in a row, column, or block. A simplex constraint is also used to force predictions to be only one digit label per image.

In addition to the rules encoding the constraints of Sudoku, the Visual-Sudoku-Classification model leverages one additional technique. Because the goal of the model is to differentiate digits rather than correctly classifying digits, the model can assign an arbitrary class to each to digit. The model uses the first row of the first correct puzzle from the train set to choose this arbitrary label assignment. Because the puzzle is known to be correct, all digits in the first row are guaranteed to be distinct. By pinning the first row to arbitrary classes, the neural model will have a starting place for differentiating between the different classes. This technique is captured by the *Pin First Row* rules in Figure 12. Additionally, the *Hint Second Row* rules expand on the above technique by providing hints to the second row of the first puzzle. Because the first row of the first training puzzle is pinned to arbitrary labels, the second row (which shares the same block) has fewer options and can be constrained as well. For example, when the first two locations in the first row are assigned to the classes 1 and 2 respectively, then the first two location in the second row are constrained to be in $\{3, 4\}$.

Visual-Sudoku-Classification Hyperparameters

Table 7 shows the hyperparameter values and the tuning ranges of the NeuPSL Visual-Sudoku-Classification models. The hypereparameter search was performed for each data setting and the best

performing model is reported. The *Neural Learning Rate* is the learning rate of the neural model used to predict image labels The hyperparameter search was performed on all data settings and the best results are reported for each data setting. All unspecified values were left at their default values.

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 7: Hyperparameters searched over for the Visual Sudoku experiments

D Baseline Neural Models

In this section we provide details on the CNN baseline models for both the MNIST-Add and Visual-Sudoku-Classification experiments.

D.1 MNIST-Add

The baseline neural models for the MNIST-Add tasks are shown in Table 8. Both models were trained to minimize cross-entropy loss.

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

(a) MNIST-Add1

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

(b) MNIST-Add2

Table 8: The architectures of the network used as the CNN baseline for MNIST-Add experiments (from Badreddine et al. (2022)).

Table 9 shows the hyperparameter values and the tuning ranges for the baseline CNN models. All unspecified values were left at their default values.

Hyperparameter	Tuning Range	Final Value
Learning Rate	{1e-3, 1e-4, 1e-5}	0.001
Batch Size	{16, 32, 64, 128}	16

(a) MNIST-Add1

Hyperparameter	Tuning Range
Learning Rate	{1e-3, 1e-4, 1e-5}
Batch Size	{16, 32, 64, 128}

(b) MNIST-Add2

Table 9: Hyperparameters searched over and used for the MNIST-Add baseline models.

The hypereparameter search was performed for each data setting and the best performing model was reported. Table 10 shows the final hyperparameters of the MNIST-Add baseline models.

	MNIST-Add1			MNIST-Add2		
	Number of Additions					
	300	3,000	25,000	150	1,500	12,500
Final Learning Rate	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
Final Batch Size	32	16	32	32	32	64

Table 10: The final hyperparameters used for the MNIST-Add baseline models.

D.2 Visual-Sudoku-Classification

D.2.1 CNN Baseline

The baseline neural models for the Visual-Sudoku-Classification tasks are shown Table 11. Both models were trained to minimize cross-entropy loss.

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

(a) CNN-Visual

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

(b) CNN-Digit

Table 11: The architectures of the network used as the CNN baseline for Visual-Sudoku-Classification experiments (from Badreddine et al. (2022)).

Table 12 shows the hyperparameter values and the tuning ranges for the baseline CNN models. All unspecified values were left at their default values.

Hyperparameter	Tuning Range
Learning Rate	{1e-3, 1e-4, 1e-5}

(a) CNN-Visual

Hyperparameter	Tuning Range
Learning Rate	{1e-3, 1e-4, 1e-5}

(b) CNN-Digit

Table 12: Hyperparameters searched over and used for the Visual-Sudoku-Classification baseline models.

The hyperparameter search was performed for each data setting and the best performing model was reported. Table 13 shows the final hyperparameters of the Visual-Sudoku-Classification baseline models for each data setting.

	Number of Puzzles			
	10	20	100	200
Final Learning Rate	1e-4	1e-3	1e-2	1e-2

(a) CNN-Visual

	Number of Puzzles			
	10	20	100	200
Final Learning Rate	1e-3	1e-2	1e-2	1e-2

(b) CNN-Digit

Table 13: The final hyperparameters used for the Visual-Sudoku-Classification baseline models.

E Extended Evaluation Details

In this section, we provide additional details for the NeSy baselines used in the experiments. We also provide additional experiments comparing with other NeSy baselines. In Table 14 we present results for the following systems:

DeepProbLog (DPL): All DPL results use the neural and DPL models presented in [Manhaeve et al., 2021a], using default hyperparameters. Code was obtained from https://github.com/ML-KULeuven/deepproblog. *DPL-gm* uses default approximate inference method presented in Manhaeve et al. [2021b]. Results for *DPL-gm* were averaged over 10 folds generated as described in Appendix B. *DPL-exact* uses exact inference method presented in [Manhaeve et al., 2021a]. Results for *DPL-exact* were averaged over 10 folds generated as described in Appendix B. *DPL-reported* presents the reported results from [Manhaeve et al., 2021a].

DeepStochLog (DSL): The DeepStochLog code was obtained from https://github.com/ML-KULeuven/deepstochlog. *DSL-reported* presents the reported results from [Winters et al., 2021].

Logic Tensor Networks (LTNs): All LTN results use the neural and LTN models presented in [Badreddine et al., 2022], using default hyperparamters. Code was obtained from https://github.com/logictensornetworks/logictensornetworks. *LTNs* results were averaged over 10 folds generated as described in Appendix B. *LTNs-reported* presents the reported results from Badreddine et al. [2022], where results are averaged on the top 10 of 15 runs.

Neural Probabilistic Soft Logic (NeuPSL): All NeuPSL results use the neural and NeuPSL models presented in Appendix C. *NeuPSL* results were averaged over 10 folds generated as described in Appendix B.

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

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
DPL-gm	33.41 ± 16.30	67.58 ± 16.59	74.08 ± 15.33	03.52 ± 03.18	84.68 ± 01.95	92.23 ± 01.01
DPL-exact	85.61 ± 01.28	92.59 ± 01.40	–.– ± –.–	71.37 ± 03.90	87.44 ± 02.15	–.– ± –.–
DPL-reported	67.19 ± 25.72	92.18 ± 01.57	97.20 ± 00.45	72.73 ± 03.03	87.21 ± 01.92	95.16 ± 01.70
DSL-reported	–.– ± –.–	–.– ± –.–	97.90 ± 00.10	–.– ± –.–	–.– ± –.–	96.40 ± 00.10
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
LTNs-reported	–.– ± –.–	93.49 ± 0.28	98.04 ± 0.13	–.– ± –.–	88.21 ± 00.63	95.37 ± 00.29
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 14: Test set accuracy and standard deviation on **MNIST-Add**. Most results are run under different data constructions and are not comparable.

F Computational Hardware Details

All timing experiments were performed on an Ubuntu 20.04.3 Linux machine with Intel(R) Xeon(R) Silver 4215R CPUs at 3.20GHz with an NVIDIA Quadro RTX 6000 GPU.