Injecting Logical Constraints into Neural Networks via Straight-Through Estimators

Zhun Yang [1] **Joohyung Lee** [1 2] **Chiyoun Park** [2]

Abstract

Injecting discrete logical constraints into neural network learning is one of the main challenges in neuro-symbolic AI. We find that a straight-through-estimator, a method introduced to train binary neural networks, could effectively be applied to incorporate logical constraints into neural network learning. More specifically, we design a systematic way to represent discrete logical constraints as a loss function; minimizing this loss using gradient descent via a straight-through-estimator updates the neural network's weights in the direction that the binarized outputs satisfy the logical constraints. The experimental results show that by leveraging GPUs and batch training, this method scales significantly better than existing neuro-symbolic methods that require heavy symbolic computation for computing gradients. Also, we demonstrate that our method applies to different types of neural networks, such as MLP, CNN, and GNN, making them learn with no or fewer labeled data by learning directly from known constraints.

1. Introduction

Neuro-symbolic AI (Besold et al., 2017; Mao et al., 2019; De Raedt et al., 2019; Garcez et al., 2019) aims to combine deep neural network learning and symbolic AI reasoning, which look intrinsically different from each other on the surface. It appears hard to incorporate discrete logical reasoning into the conventional gradient descent method that deals with continuous values. Some recent works in neuro-symbolic AI (Manhaeve et al., 2018; Yang et al., 2020; Pogancic et al., 2020; Tsamoura et al., 2021) associate continuous parameters in neural networks (NNs) with logic languages so that logical reasoning applied to NN outputs produces "semantic loss" (Xu et al., 2018). Minimizing such loss leads to updating NN parameters via backpropagation through logic layers. Like human learning that leverages known constraints, these methods have shown promising results that allow NNs to learn effectively with fewer data leveraging the semantic constraints. On the other hand, the symbolic computation performed during backpropagation is implemented by weighted model counting using circuits (Darwiche, 2011; Manhaeve et al., 2018; Tsamoura et al., 2021) or by calling symbolic solvers (Pogančić et al., 2019; Yang et al., 2020), which are often computationally expensive; it takes too long to generate arithmetic circuits or enumerate all models or proofs by calling symbolic solvers.

One main reason for the development of the different ideas is that a naive representation of discrete constraints as a loss function is not meaningfully differentiable. Even for the intervals that it is differentiable, the gradient is zero, so NNs won't update their weights. To address this, we turn to the idea of straight-through estimators (STE) (Courbariaux et al., 2015), which were originally introduced to train binary neural networks (BNNs) — neural networks with binary weights and activation at run-time. The main idea of STE is to use a binarization function in forward propagation while its gradient, which is zero almost everywhere, is replaced by the gradient of a different, meaningfully differentiable function in backward propagation. It turns out that the method works well for NN quantization in practice.

However, adopting STE alone is not enough for learning with constraints. We need a systematic method of encoding logical constraints as a loss function and ensure that its gradient enables NNs to learn logical constraints.

This paper makes the following contributions.

- We design a systematic way to encode logical constraints in propositional logic as a loss function in neural network learning, which we call *CL-STE*. We demonstrate that minimizing this loss function via STE enforces the logical constraints in neural network learning so that neural networks learn from the explicit constraints.

[1] School of Computing and Augmented Intelligence, Fulton Schools of Engineering, Arizona State University, Tempe, AZ, USA [2] Samsung Research, Samsung Electronics Co., Seoul, South Korea. Correspondence to: Joohyung Lee <joolee@asu.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

- We show that by leveraging GPUs and batch training, CL-STE scales significantly better than the other neuro-symbolic learning methods that use heavy symbolic computation for computing gradients.
- We also find that the concept of Training Gate Function (TGF) (Kim et al., 2020), which was applied to channel pruning, is closely related to STE. We establish the precise relationship between them, which gives a new perspective of STE.

The paper is organized as follows. Section 2 presents related works, and Section 3 reviews STE and TFG and establish their relationships. Section 4 presents our loss function representation of logical constraints and proves its properties assuming minimization via STE, and Section 5 shows experimental results.

The implementation of our method is publicly available online at https://github.com/azreasoners/cl-ste.

2. Related Work

Our work is closely related to (Xu et al., 2018), which proposes a semantic loss function to bridge NN outputs and logical constraints. The method treats an NN output as probability and computes semantic loss as the negative logarithm of the probability to generate a state satisfying the logical constraints. Their experiments show that the encoded semantic loss function guides the learner to achieve state-of-the-art results in supervised and semi-supervised learning on multi-class classification. For the efficient computation of a loss function, they encode logical constraints in Sentential Decision Diagram (SDD) (Darwiche, 2011). However, generating SDDs is computationally expensive for most practical tasks.

Several neuro-symbolic formalisms, such as DeepProbLog (Manhaeve et al., 2018), NeurASP (Yang et al., 2020), and NeuroLog (Tsamoura et al., 2021), have been proposed to integrate neural networks with logic programming languages. Since discrete logical inference cannot be in general captured via a differentiable function, they use relaxation to weighted models or probability. While this approach provides a systematic representation of constraints, the symbolic computation is often the bottleneck in training.

Since fuzzy logic operations are naturally differentiable, several works, such as Logic Tensor Network (Serafini & Garcez, 2016), Continuous Query Decomposition (Arakelyan et al., 2020), Semantic Based Regularization (Diligenti et al., 2017; Roychowdhury et al., 2021), directly apply fuzzy operators to neural network outputs. However, as stated in (Marra et al., 2021), the fuzzification procedure alters the logical properties of the original theory (such as satisfiability).

Other works train neural networks for learning satisfiability, such as (Wang et al., 2019; Selsam et al., 2019). SATNet (Wang et al., 2019) builds on a line of research exploring SDP relaxations as a tool for solving MAXSAT, producing tighter approximation guarantees than standard linear programming relaxation.

Graph Neural Networks (GNNs) (Battaglia et al., 2018; Lamb et al., 2020) have been widely applied for logical reasoning. For example, Recurrent Relational Network (RRN) was able to learn how to solve Sudoku puzzles. GNNs use message-passing to propagate logical constraints in neural networks, but they do not have the mechanism to specify the logical constraints directly as we do.

While STE has not been exploited in neuro-symbolic learning to our best knowledge, (Pogancic et al., 2020)'s work is related in that it also uses a gradient that is different from the forward function's gradient. It uses the gradient obtained from a linear relaxation of the forward function. The work also requires a combinatorial solver to compute the gradient.

3. Straight-Through-Estimators and Trainable Gate Function

Review. STEs are used to estimate the gradients of a discrete function. Courbariaux et al. (2015) consider a binarization function b that transforms real-valued weights x into discrete values $b(x)$ as $b(x) = 1$ if $x \geq 0$ and $b(x) = 0$ otherwise. A loss function L is defined on binarized weights $b(x)$, but the gradient descent won't update binarized weights in small increments. However, using STE, we could update the real-valued weights x that are input to $b(x)$. In the end, a quantized model consists of binarized weights $b(x)$ only. More specifically, according to the chain rule, the gradient of loss L w.r.t. x is $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial b(x)} \times \frac{\partial b(x)}{\partial x}$, where $\frac{\partial b(x)}{\partial x}$ is zero almost everywhere. The idea is to replace $\frac{\partial b(x)}{\partial x}$ with an STE $\frac{\partial s(x)}{\partial x}$ for some (sub)differentiable function $s(x)$. The STE $\frac{\partial s(x)}{\partial x}$ is called the *identity STE* (iSTE) if $s(x) = x$ and is called the *saturated STE* (sSTE) if $s(x) = clip(x, [-1, 1]) = min(max(x, -1), 1)$. Since $\frac{\partial s(x)}{\partial x} = 1$, by $\frac{\partial L}{\partial x} \overset{iSTE}{\approx} \frac{\partial L}{\partial b(x)}$, we denote the identification of $\frac{\partial L}{\partial x}$ with $\frac{\partial L}{\partial b(x)}$ under iSTE.

The binarization function $b(x)$ passes only the sign of x while information about the magnitude of x is lost (Simons & Lee, 2019). In XNOR-Net (Rastegari et al., 2016), the input x is normalized to have the zero mean and a small variance before the binarization to reduce the information loss. In this paper, we normalize x by turning it into a probability using softmax or sigmoid activation functions. Indeed, several neuro-symbolic learning methods (e.g., Deep-

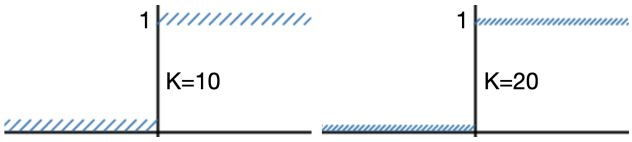

Figure 1. Trainable gate function $\widetilde{b}^K(x)$ when $g(x) = 1$

ProbLog, NeurASP, NeuroLog) assume the neural network outputs fed into the logic layer are normalized as probabilities. To address a probabilistic input, we introduce a variant binarization function $b_p(x)$ for probabilities $x \in [0, 1]$: $b_p(x) = 1$ if $x \geq 0.5$ and $b_p(x) = 0$ otherwise. It is easy to see that iSTE and sSTE work the same with $b_p(x)$ since $x = clip(x, [-1, 1])$ when $x \in [0, 1]$. A vector $\mathbf{x}$ is allowed as input to the binarization functions b and b_p, in which case they are applied to each element of $\mathbf{x}$.

TGF and Its Relation to STE. The concept of STE is closely related to that of the Trainable Gate Function (TGF) from (Kim et al., 2020), which was applied to channel pruning. Instead of replacing the gradient $\frac{\partial b(x)}{\partial x}$ with an STE, TGF tweaks the binarization function $b(x)$ to make it meaningfully differentiable. More specifically, a differentiable binarization function $\widetilde{b}^K$ is defined as

$$\widetilde{b}^K(x) = b(x) + s^K(x)g(x), \tag{1}$$

where K is a large constant; $s^K(x) = \frac{Kx - \lfloor Kx \rfloor}{K}$ is called a *gradient tweaking* function, whose value is less than $\frac{1}{K}$ and whose gradient is always 1 wherever differentiable; $g(x)$ is called a *gradient shaping* function, which could be an arbitrary function, but the authors note that the selection does not affect the results critically and $g(x) = 1$ can be adopted without significant loss of accuracy. As obvious from Figure 1, as K becomes large, TGF $\widetilde{b}^K(x)$ is an approximation of $b(x)$, but its gradient is 1 wherever differentiable.

Proposition 3.1 tells us a precise relationship between TGF and STE: when K is big enough, the binarization function $b(x)$ with iSTE or sSTE can be simulated by TGF. In other words, Proposition 3.1 allows us to visualize $b(x)$ with STE as the TGF $\widetilde{b}^K(x)$ with $K = \infty$ as Figure 1 illustrates.

Proposition 3.1. *When K approaches ∞ and $|g(x)| \leq c$ for some constant c, the value of $\widetilde{b}^K(x)$ converges to $b(x)$:*

$$\lim_{K \to \infty} \widetilde{b}^K(x) = b(x).$$

The gradient $\frac{\partial \widetilde{b}^K(x)}{\partial x}$, wherever defined, is exactly the iSTE of $\frac{\partial b(x)}{\partial x}$ if $g(x) = 1$, or the sSTE of $\frac{\partial b(x)}{\partial x}$ if

$$g(x) = \begin{cases} 1 & \text{if } -1 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

Proposition 3.1 still holds if we replace $b(x)$ with $b_p(x)$.

The proposition yields insights into STE and TGF in terms of each other. As shown in Figure 1, TGF is a sawtooth function that approximates a step function as K becomes large. At large, TGF works like a discrete function, but it is differentiable almost everywhere. In view of Proposition 3.1, this fact gives an idea why the STE method works in practice. On the other hand, the proposition tells that the implementation of TGF can be replaced with STE. That could be better because TGF in equation (1) requires that K approximate infinity and be non-differentiable when x is a multiple of $\frac{1}{K}$ whereas STE is differentiable at every x.

4. Enforcing Logical Constraints using STE

This section presents our method of encoding logical constraints in propositional logic as a loss function so that minimizing its value via STE makes neural network prediction follow the logical constraints.

4.1. Encoding CNF as a Loss Function Using STE

We first review the terminology in propositional logic. A *signature* is a set of symbols called *atoms*. Each atom represents a proposition that is true or false. A *literal* is either an atom p (*positive literal*) or its negation $\neg p$ (*negative literal*). A *clause* is a disjunction over literals, e.g., $p_1 \vee \neg p_2 \vee p_3$. A *Horn clause* is a clause with at most one positive literal. We assume a *(propositional) theory* consisting of a set of clauses (sometimes called a *CNF (Conjunctive Normal Form) theory*). A truth assignment to atoms *satisfies* (denoted by $\models$) a theory if at least one literal in each clause is true under the assignment. A theory is *satisfiable* if at least one truth assignment satisfies the theory. A theory *entails* (also denoted by $\models$) a literal if every truth assignment that satisfies the theory also satisfies that literal.

We define a general loss function L_{cnf} for any CNF theory as follows. Here, bold upper and lower letters (e.g., $\mathbf{C}$ and $\mathbf{v}$) denote matrices and vectors, respectively; $\mathbf{C}[i, j]$ and $\mathbf{v}[i]$ denote their elements.

Consider a propositional signature $\sigma = \{p_1, \ldots, p_n\}$. Given (i) a theory C consisting of m clauses (encoding domain knowledge), (ii) a set F of atoms denoting some atomic facts that we assume known to be true (representing the ground-truth label of a data instance), and (iii) a truth assignment v such that $v \models F$, we construct their matrix/vector representations as

- the matrix $\mathbf{C} \in \{-1, 0, 1\}^{m \times n}$ to represent the theory such that $\mathbf{C}[i, j]$ is 1 (-1, resp.) if p_j ($\neg p_j$, resp.) belongs to the i-th clause in the theory, and is 0 if neither p_j nor $\neg p_j$ belongs to the clause;
- the vector $\mathbf{f} \in \{0, 1\}^n$ to represent F such that $\mathbf{f}[j]$ is 1 if $p_j \in F$ and is 0 otherwise; and

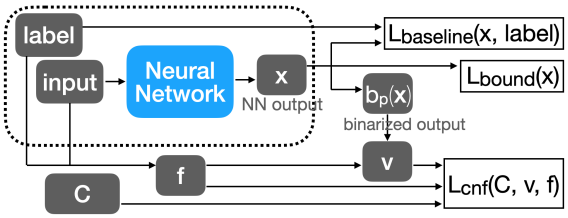

Figure 2. Architecture that overlays constraint loss

- the vector $\mathbf{v} \in \{0,1\}^n$ to represent v such that $\mathbf{v}[j]$ is 1 if $v(p_j) = \text{TRUE}$, and is 0 if $v(p_j) = \text{FALSE}$.

Figure 2 shows an architecture that overlays the two loss functions L_{bound} and L_{cnf} over the neural network output, where L_{cnf} is the main loss function to encode logical constraints and L_{bound} is a regularizer to limit the raw neural network output not to grow too big (more details will follow). The part **input** is a tensor (e.g., images) for a data instance; **label** denotes the labels of input data; $\mathbf{C}$ encodes the domain knowledge, $\mathbf{x} \in [0,1]^n$ denotes the NN output (in probability), and $\mathbf{f} \in \{0,1\}^n$ records the known facts in that data instance (e.g., given digits in Sudoku).[1] Let $\mathbb{1}_{\{k\}}(X)$ denote an indicator function that replaces every element in X with 1 if it is k and with 0 otherwise. Then the binary prediction $\mathbf{v}$ is constructed as $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$, where $\odot$ denotes element-wise multiplication. Intuitively, $\mathbf{v}$ is the binarized NN output with part of it strictly following the given facts specified in $\mathbf{f}$ (ensuring $v \models F$).

Example 4.1. *Consider a simple example* **mnistAdd** *from (Manhaeve et al., 2018), where the task is, given a pair of MNIST digit images and their sum as the label, to let a neural network learn the digit classification of the input images. The example is used to demonstrate how NNs can learn from known constraints. In Figure 2, the input consists of two-digit images i_1 and i_2, and the label is an integer l in $\{0, ..., 18\}$ denoting the sum of i_1 and i_2. The neural network is the same Convolutional Neural Network (CNN) used in (Manhaeve et al., 2018).*

The theory for this problem consists of the following clause for $l \in \{0, \ldots, 18\}$, where $sum(l)$ represents "the sum of i_1 and i_2 is l" and $pred(n_1, n_2)$ represents "the neural network predicts i_1 and i_2 as n_1 and n_2 respectively":

$$\neg sum(l) \vee \bigvee_{\substack{n_1,n_2 \in \{0,\ldots,9\}: \\ n_1+n_2=l}} pred(n_1, n_2).$$

This theory contains $19 + 100 = 119$ atoms for $sum/1$ and $pred/2$ respectively. We construct the matrix $\mathbf{C} \in \{-1,0,1\}^{19\times 119}$, where each row represents a clause. For instance, the row for the clause $\neg sum(1) \vee pred(0,1) \vee pred(1,0)$ is a vector in $\{-1,0,1\}^{1\times 119}$ containing a single -1 for atom $sum(1)$, two 1s for atoms $pred(0,1)$ and $pred(1,0)$, and 116 0s.

Vectors $\mathbf{f}$ and $\mathbf{v}$ are in $\{0,1\}^{119}$ constructed from each data instance $\langle i_1, i_2, l\rangle$. The fact vector $\mathbf{f}$ contains a single 1 for atom $sum(l)$ (ground truth label) and 118 0s. To obtain the prediction vector $\mathbf{v}$, we (i) feed images i_1,i_2 into the CNN (with softmax at the last layer) from (Manhaeve et al., 2018) to obtain the outputs $\mathbf{x}_1, \mathbf{x}_2 \in [0,1]^{10}$ (consisting of probabilities), (ii) construct the vector $\mathbf{x} \in [0,1]^{100}$ (for 100 atoms of $pred/2$) such that $\mathbf{x}[10i+j]$ is $\mathbf{x}_1[i] \times \mathbf{x}_2[j]$ for $i, j \in \{0, \ldots, 9\}$, (iii) update $\mathbf{x}$ as the concatenation of $\mathbf{x}$ and $\{0\}^{19}$, and (iv) finally, let $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$.

Using $\mathbf{C}$, $\mathbf{v}$, and $\mathbf{f}$, we define the *CNF loss* $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ as follows:

$$\mathbf{L}_f = \mathbf{C} \odot \mathbf{f} \tag{2}$$

$$\mathbf{L}_v = \mathbb{1}_{\{1\}}(\mathbf{C}) \odot \mathbf{v} + \mathbb{1}_{\{-1\}}(\mathbf{C}) \odot (1 - \mathbf{v}) \tag{3}$$

$$\mathbf{deduce} = \mathbb{1}_{\{1\}}\Big(sum(\mathbf{C} \odot \mathbf{C}) - sum(\mathbb{1}_{\{-1\}}(\mathbf{L}_f))\Big) \tag{4}$$

$$\mathbf{unsat} = prod(1 - \mathbf{L}_v) \tag{5}$$

$$\mathbf{keep} = sum(\mathbb{1}_{\{1\}}(\mathbf{L}_v) \odot (1 - \mathbf{L}_v) + \mathbb{1}_{\{0\}}(\mathbf{L}_v) \odot \mathbf{L}_v) \tag{6}$$

$$L_{deduce} = sum(\mathbf{deduce} \odot \mathbf{unsat}) \tag{7}$$

$$L_{unsat} = avg(\mathbb{1}_{\{1\}}(\mathbf{unsat}) \odot \mathbf{unsat}) \tag{8}$$

$$L_{sat} = avg(\mathbb{1}_{\{0\}}(\mathbf{unsat}) \odot \mathbf{keep}) \tag{9}$$

$$L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) = L_{deduce} + L_{unsat} + L_{sat} \tag{10}$$

where $prod(X)$, $sum(X)$, and $avg(X)$ compute the product, sum, and average of the elements in X along its last dimension. [2] Although these equations may look complex, it helps to know that they use the form $\mathbb{1}_{\{k\}}(X_1) \odot X_2$, where the indicator function $\mathbb{1}_{\{k\}}(X_1)$ can be seen as a constant that is multiplied to a trainable variable X_2. Take equation (8) as an example. To minimize L_{unsat}, the NN parameters will be updated towards making $\mathbf{unsat}[i]$ to be 0 whenever $\mathbb{1}_{\{1\}}(\mathbf{unsat})$ is 1, i.e., towards making unsatisfied clauses satisfied.

In equations (2) and (3), $\mathbf{f}$ and $\mathbf{v}$ are treated as matrices in $\{0,1\}^{1\times n}$ to have element-wise multiplication (with broadcasting) with a matrix in $\{-1,0,1\}^{m\times n}$. Take equation (2) as an example, $\mathbf{L}_f[i,j] = \mathbf{C}[i,j] \times \mathbf{f}[j]$. $\mathbf{L}_f$ is the matrix in $\{-1,0,1\}^{m\times n}$ such that (i) $\mathbf{L}_f[i,j] = 1$ iff clause i contains literal p_j and $p_j \in F$; (ii) $\mathbf{L}_f[i,j] = -1$ iff clause i contains literal $\neg p_j$ and $p_j \in F$; (iii) otherwise, $\mathbf{L}_f[i,j] = 0$.

[1] In case the length of $\mathbf{x}$ is less than n, we pad $\mathbf{x}$ with 0s for all the atoms that are not related to NN output.

[2] The aggregated dimension is "squeezed," which is the default behavior in PyTorch aggregate functions (`keepdim` is False).

$\mathbf{L}_v$ is the matrix in $\{0,1\}^{m\times n}$ such that $\mathbf{L}_v[i,j] = 1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal is TRUE under v.

In equations (4), (5), and (6), $sum(\mathbf{C} \odot \mathbf{C})$ is a vector in $\mathbb{N}^m$ representing in each clause the number of literals, and $sum(\mathbb{1}_{\{-1\}}(\mathbf{L}_f))$ is a vector in $\mathbb{N}^m$ representing in each clause the number of literals that are FALSE under F (i.e., the number of literals of the form $\neg p$ such that $p \in F$). Consequently, $\mathbf{deduce}$ is a vector in $\{0,1\}^m$ where $\mathbf{deduce}[i]$ is 1 iff clause i has all but one literal being FALSE under F. If $C \cup F$ is satisfiable and a clause has all but one literal being FALSE under F, then we can safely deduce that the remaining literal is TRUE. For instance, in a clause for Sudoku

$$\neg a(1,1,9) \vee \neg a(1,2,9), \tag{11}$$

if $a(1,1,9)$ is in the ground-truth label (i.e., in F) but $a(1,2,9)$ is not, we can safely deduce $\neg a(1,2,9)$ is true. It follows that such a clause is always a Horn clause. Intuitively, the vector $\mathbf{deduce}$ represents the clauses that such deduction can be applied given F.

The vector $\mathbf{unsat} \in \{0,1\}^m$ indicates which clause is not satisfied by the truth assignment v, where $\mathbf{unsat}[i]$ is 1 iff v does not satisfy the i-th clause. The vector $\mathbf{keep} \in \{0\}^m$ consists of m zeros while its gradient w.r.t. $\mathbf{v}$ consists of non-zeros. Intuitively, the gradient of $\mathbf{keep}$ tries to keep the current predictions $\mathbf{v}$ in each clause.

In equations (7), (8), and (9), $L_{deduce} \in \mathbb{N}$ represents the number of clauses that can deduce a literal given F and are not satisfied by v. The vector $\mathbb{1}_{\{1\}}(\mathbf{unsat}) \in \{0,1\}^m$ (and $\mathbb{1}_{\{0\}}(\mathbf{unsat})$, resp.) indicates the clauses that are not satisfied (and are satisfied, resp.) by v. Intuitively, for all clauses, minimizing L_{unsat} makes the neural network change its predictions to decrease the number of unsatisfied clauses. In contrast, minimizing L_{sat} makes the neural network more confident in its predictions in the satisfied clauses. We use avg instead of sum in equations (8) and (9) to ensure that the gradients from L_{unsat} and L_{sat} do not overpower those from L_{deduce}. Formal statements of these intuitive explanations follow in the next section.

For any neural network output $\mathbf{x}$ consisting of probabilities, let $\mathbf{x}^r$ denote the raw value of $\mathbf{x}$ before the activation function (e.g., softmax or sigmoid) in the last layer. Without restriction, the value $\mathbf{x}^r$ may vary in a large range when trained with STE. When such an output is fed into softmax or sigmoid, it easily falls into a saturation region of the activation function (Tang et al., 2017). To resolve this issue, we include another loss function to bound the scale of $\mathbf{x}^r$:

$$L_{bound}(\mathbf{x}) = avg(\mathbf{x}^r \odot \mathbf{x}^r).$$

To enforce constraints, we add the weighted sum of $L_{cnf}(\mathbf{C},\mathbf{v},\mathbf{f})$ and $L_{bound}(\mathbf{x})$ to the baseline loss (if any), where the weight of each loss is a hyperparameter. We call this way of semantic regularization the *CL-STE* (Constraint Loss via STE) method.

Example 4.1 Continued. *Given the matrix $\mathbf{C}$ for the CNF theory, a data instance $\langle i_1, i_2, l\rangle$, the NN outputs $\mathbf{x}_1, \mathbf{x}_2$ for i_1, i_2, and the vectors $\mathbf{f}, \mathbf{v}$ as constructed in Example 4.1, the total loss function used for* **mnistAdd** *problem is*

$$\mathcal{L} = L_{cnf}(\mathbf{C},\mathbf{v},\mathbf{f}) + \sum_{\mathbf{x}\in\{\mathbf{x}_1,\mathbf{x}_2\}} 0.1\times L_{bound}(\mathbf{x}).$$

4.2. Properties of Constraint Loss and Its Gradients

Proposition 4.2 shows the relation between L_{deduce}, L_{unsat}, and L_{sat} components in the constraint loss L_{cnf} and its logical counterpart.

Proposition 4.2. *Given a theory C, a set F of atoms, and a truth assignment v such that $v \models F$, let $\mathbf{C},\mathbf{f},\mathbf{v}$ denote their matrix/vector representations, respectively. Let $C_{deduce} \subseteq C$ denote the set of Horn clauses H in C such that all but one literal in H are of the form $\neg p$ such that $p \in F$.* [3] *Then*

- *the minimum values of L_{deduce}, L_{unsat}, L_{sat}, and $L_{cnf}(\mathbf{C},\mathbf{v},\mathbf{f})$ are 0;*
- *$v \models C_{deduce}$ iff L_{deduce} is 0;*
- *$v \models C$ iff L_{unsat} is 0 iff $L_{cnf}(\mathbf{C},\mathbf{v},\mathbf{f})$ is 0.*

Clause (11) is an example clause in C_{deduce}. There could be many other ways to design $L_{cnf}(\mathbf{C},\mathbf{v},\mathbf{f})$ to satisfy the properties in Proposition 4.2. Propositions 4.3 and 4.5 below justify our design choice.

Proposition 4.3. *Given a theory C with m clauses and n atoms and a set F of atoms such that $C \cup F$ is satisfiable, let $\mathbf{C},\mathbf{f}$ denote their matrix/vector representations, respectively. Given a neural network output $\mathbf{x} \in [0,1]^n$ denoting probabilities, we construct $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$ and a truth assignment v such that $v(p_j) =$ TRUE if $\mathbf{v}[j]$ is 1, and $v(p_j) =$ FALSE if $\mathbf{v}[j]$ is 0. Let $C_{deduce} \subseteq C$ denote the set of Horn clauses H in C such that all but one literal in H are of the form $\neg p$ where $p \in F$. Then, for any $j \in \{1,\ldots,n\}$,*

1. *if $p_j \in F$, all of $\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]}$, $\frac{\partial L_{unsat}}{\partial \mathbf{x}[j]}$, and $\frac{\partial L_{sat}}{\partial \mathbf{x}[j]}$ are zeros;*
2. *if $p_j \notin F$,*

$$\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \begin{cases} -c & \text{if } c>0 \text{ clauses in } C_{deduce} \\ & \text{contain literal } p_j; \\ c & \text{if } c>0 \text{ clauses in } C_{deduce} \\ & \text{contain literal } \neg p_j; \\ 0 & \text{otherwise;} \end{cases}$$

[3] This implies that the remaining literal is either an atom or $\neg p$ such that $p \notin F$.

$$\frac{\partial L_{unsat}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \frac{c_2 - c_1}{m}$$

$$\frac{\partial L_{sat}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \begin{cases} -\frac{c_3}{m} & \text{if } v \models p_j \\ \frac{c_3}{m} & \text{if } v \not\models p_j, \end{cases}$$

where $\overset{iSTE}{\approx}$ *stands for the equivalence of gradients assuming iSTE;* c_1 *(and* c_2*, resp.) is the number of clauses in* C *that are not satisfied by* v *and contain* p_j *(and* $\neg p_j$*, resp.);* c_3 *is the number of clauses in* C *that are satisfied by* v *and contain* p_j *or* $\neg p_j$*.*

Intuitively, Proposition 4.3 ensures the following properties of the gradient $\frac{\partial L_{cnf}(\mathbf{C},\mathbf{v},\mathbf{f})}{\partial \mathbf{x}[j]}$, which consists of $\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]}$, $\frac{\partial L_{unsat}}{\partial \mathbf{x}[j]}$, and $\frac{\partial L_{sat}}{\partial \mathbf{x}[j]}$.

P1. If we know for sure that p_j is TRUE ($p_j \in F$), these gradients w.r.t. $\mathbf{x}[j]$ (real values corresponding to p_j) are 0, so they do not affect the truth value of p_j.

P2. Otherwise (F does not tell whether p_j is TRUE),

1. the gradient $\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]}$ is negative (positive, resp.) to increase (decrease, resp.) the value of $\mathbf{x}[j]$ by the gradient descent if $C \cup F$ entails p_j ($\neg p_j$, resp.);
2. the gradient $\frac{\partial L_{unsat}}{\partial \mathbf{x}[j]}$ is negative (positive resp.) to increase (decrease, resp.) the value of $\mathbf{x}[j]$ by the gradient descent if, among all unsatisfied clauses, more clauses contain p_j than $\neg p_j$ ($\neg p_j$ than p_j, resp.);
3. the gradient $\frac{\partial L_{sat}}{\partial \mathbf{x}[j]}$ is negative (positive resp.) to increase (decrease, resp.) the value of $\mathbf{x}[j]$ by the gradient descent if $v \models p_j$ ($v \not\models p_j$, resp.) and there exist satisfied clauses containing literal p_j or $\neg p_j$.

Intuitively, bullet 1 in **P2** simulates a deduction step, which is always correct, while bullets 2 and 3 simulate two heuristics: "we tend to believe a literal if more unsatisfied clauses contain this literal than its negation" and "we tend to keep our prediction on an atom if many satisfied clauses contain this atom." This justifies another property below.

P3. The sign of the gradient $\frac{\partial L_{cnf}}{\partial \mathbf{x}[j]}$ is the same as the sign of $\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]}$ when the latter gradient is non-zero.

Example 4.4. *Consider the theory* C *below with* $m = 2$ *clauses and 3 atoms*

$$\neg a \vee \neg b \vee c$$
$$\neg a \vee b$$

and consider the set of given facts $F = \{a\}$*. They are represented by matrix* $\mathbf{C} = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 0 \end{bmatrix}$ *and vector* $\mathbf{f} = [1, 0, 0]$*. Suppose a neural network predicts* $\mathbf{x} = [0.3, 0.1, 0.9]$ *as the probabilities of the 3 atoms* $\{a, b, c\}$*.*

With the above matrix and vectors, we can compute

$$b_p(\mathbf{x}) = [0, 0, 1],$$
$$\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x}) = [1, 0, 1].$$

From $\mathbf{v}$*, we construct the truth assignment* $v = \{a = \text{TRUE}, b = \text{FALSE}, c = \text{TRUE}\}$*. Clearly,* v *satisfies the first clause but not the second one. Given* $F = \{a\}$*, we see* C_{deduce} *is* $\neg a \vee b$ *.*

According to Proposition 4.3,

$$\frac{\partial L_{deduce}}{\partial \mathbf{x}} \overset{iSTE}{\approx} [0, -1, 0], \quad \frac{\partial L_{unsat}}{\partial \mathbf{x}} \overset{iSTE}{\approx} [0, -\frac{1}{2}, 0],$$
$$\frac{\partial L_{sat}}{\partial \mathbf{x}} \overset{iSTE}{\approx} [0, \frac{1}{2}, -\frac{1}{2}],$$
$$\frac{\partial L_{cnf}}{\partial \mathbf{x}} = \frac{\partial L_{deduce}}{\partial \mathbf{x}} + \frac{\partial L_{unsat}}{\partial \mathbf{x}} + \frac{\partial L_{sat}}{\partial \mathbf{x}} \overset{iSTE}{\approx} [0, -1, -\frac{1}{2}].$$

Intuitively, given C*,* F*, and the current truth assignment* v*, (***P1***) we know* a *is* TRUE *(*$a \in F$*) thus no need to update it, (***P2.1*** and* ***P3****) we know for sure that the prediction for* b *should be changed to* TRUE *by deduction on clause* $\neg a \vee b$ *and the given fact* $F = \{a\}$*, (***P2.3****) we tend to strengthen our belief in* c *being* TRUE *due to the satisfied clause* $\neg a \vee \neg b \vee c$ *.*

The proposition also holds with another binarization function $b(x)$.

Proposition 4.5. *Proposition 4.3 still holds for* $\mathbf{x} \in \mathbb{R}^n$ *and* $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b(\mathbf{x})$.

5. Evaluation

We conduct an experimental evaluation to answer the following questions.

Q1 Is CL-STE more scalable in injecting discrete constraints into neural network learning than existing neuro-symbolic learning methods?

Q2 Does CL-STE make neural networks learn with no or fewer labeled data by effectively utilizing the given constraints?

Q3 Is CL-STE general enough to overlay constraint loss on different types of neural networks to enforce logical constraints and improve the accuracy of existing networks?

Our implementation takes a CNF theory in DIMACS format (the standard format for input to SAT solvers).[4] Since the

[4] All experiments in this section were done on Ubuntu 18.04.2 LTS with two 10-cores CPU Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz and four GP104 [GeForce GTX 1080].

Table 1. Experiments on **mnistAdd**

	mnistAdd	mnistAdd2	mnistAdd3
DeepProbLog	98.36% 2565s	97.57% 22699s	timeout
NeurASP	97.87% 292s	97.85% 1682s	timeout
CL-STE	97.48% 22s	98.12% 92s	97.78% 402s

CL-STE method alone doesn't have associated symbolic rules, unlike DeepProbLog, NeurASP, and NeuroLog, in this section, we compare these methods on the classification accuracy of the trained NNs (e.g., correctly predicting the label of an MNIST image) instead of query accuracy (e.g., correctly predicting the sum of two MNIST images).

5.1. mnistAdd Revisited

We introduced the CNF encoding and the loss function for the **mnistAdd** problem in Example 4.1. The problem was used in (Manhaeve et al., 2018) and (Yang et al., 2020) as a benchmark.

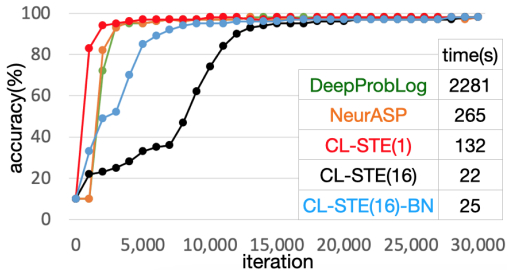

Figure 3. Comparison on mnistAdd

Figure 3 compares the MNIST digit classification accuracy of neural networks trained by different methods on a single epoch of 30,000 addition data from (Manhaeve et al., 2018). "CL-STE(n)" denotes our method with $b_p(x)$ and iSTE using a batch of size n. As we see, DeepProbLog, NeurASP, and CL-STE with a batch size of 1 could quickly converge to near 100% test accuracy. Training time-wise, CL-STE outperforms the other approaches since it does not need to generate arithmetic circuits for every training instance as in DeepProbLog or enumerate all models as in NeurASP. Also, while DeepProbLog and NeurASP do not support batch training, CL-STE could leverage the batch training to reduce the training time to 22s with a batch size of 16 (denoted by CL-STE(16)). We observe that increasing the batch size in CL-STE also increases the number of parameter updates for convergence, which we could decrease by using batch normalization as shown in the blue line denoted by CL-STE(16)-BN.

Furthermore, we apply CL-STE to the variants of **mnistAdd** by training with two-digit sums (**mnistAdd2** (Manhaeve et al., 2018)) and three-digit sums (**mnistAdd3**). Table 1 shows that the CL-STE method scales much better than DeepProbLog and NeurASP. The time and accuracy are reported for a single epoch of training, where the cutoff time is 24 hours after which we report "timeout."

Table 2. Comparison between CL-STE and other approaches: The numbers in parentheses are the times spent by NeuroLog to generate all abductive proofs.

	add2x2	apply2x2	member(3)	member(5)
accuracy(%)				
DeepProbLog	88.4±0.7	100±0	96.3±0.3	timeout
NeurASP	97.6±0.2	100±0	93.5±0.9	timeout
NeuroLog	97.5±0.4	100±0	94.5±1.5	93.9±1.5
$b(x)$ + iSTE	95.5±0.7	100±0	73.2±9.1	51.1±24.9
$b(x)$ + sSTE	95.7±0.5	100±0	83.2±8.4	88.0±7.1
$b_p(x)$ + iSTE	98.0±0.2	100±0	95.5±0.7	95.0±0.5
time(s)				
DeepProbLog	1035±71	586±9	2218±211	timeout
NeurASP	142±2	47±1	253±1	timeout
NeuroLog	2400±46 (1652)	2428±29 (2266)	427±12 (27)	682±40 (114)
$b(x)$ + iSTE	80±2	208±1	45±0	177±1
$b(x)$ + sSTE	81±2	214±8	46±1	181±10
$b_p(x)$ + iSTE	54±4	112±2	43±3	49±4

5.2. Benchmarks from (Tsamoura et al., 2021)

The following are benchmark problems from (Tsamoura et al., 2021). Like the **mnistAdd** problem, labels are not immediately associated with the data instances but with the results of logical operations applied to them.

add2x2 The input is a 2×2 grid of digit images. The output is the four sums of the pairs of digits in each row/column. The task is to train a CNN for digit classification.

apply2x2 The input is three digits and a 2×2 grid of handwritten math operator images in $\{+,-,\times\}$. The output is the four results of applying the two math operators in each row/column in the grid on the three digits. The task is to train a CNN for math operator classification.

member(n) The input is a set of n images of digits and a digit in $\{0,\ldots,9\}$. The output is 0 or 1, indicating whether the digit appears in the set of digit images. The task is to train a CNN for digit classification.

Table 2 compares our method with DeepProbLog, NeurASP, and NeuroLog test accuracy-wise and training time-wise. Note that, instead of comparing the query accuracy as in (Tsamoura et al., 2021), we evaluate and compare the NN classification accuracies.

Our experiments agree with (Yin et al., 2019), which proves the instability issue of iSTE and the convergence guarantees with sSTE in a simple 2-layer CNN. Their experiments also observe a better performance of $b(x)$+sSTE over $b(x)$+iSTE on deep neural networks. Our experimental results (especially for member(5) problem) also reveal the instability issue of $b(x)$+iSTE and show that $b(x)$+sSTE achieves higher and more stable accuracy. Furthermore, we observe that $b_p(x)$ works better than $b(x)$ in terms of both accuracy and time in our experiments. This is because the input x to $b_p(x)$ is normalized into probabilities before binarization, resulting in less information loss (i.e., change in magnitude

Table 3. CNN, NeurASP, and CL-STE on Park 70k Sudoku dataset (80%/20% split) w/ and w/o inference trick

Method	Supervised	Acc_{wo}	Acc_w	time(m)
Park's CNN	Full	0.94%	23.3%	163
Park's CNN+NeurASP	No	1.69%	66.5%	13230
Park's CNN+CL-STE	No	2.38%	93.7%	813

"$b_p(x) - x$") when the neural network accuracy increases.

5.3. CNN + Constraint Loss for Sudoku

The following experimental setting from (Yang et al., 2020) demonstrates unsupervised learning with NeurASP on Sudoku problems. Given a textual representation of a Sudoku puzzle (in the form of a 9×9 matrix where an empty cell is represented by 0), Park (2018) trained a CNN (composed of 9 convolutional layers and a 1×1 convoutional layer, followed by softmax) using 1 million examples and achieved 70% test accuracy using an "inference trick": instead of predicting digits for all empty cells at once, which leads to poor accuracy, predict the most probable grid-cell value one by one. With the same CNN and inference trick, Yang et al. (2020) achieved 66.5% accuracy with only 7% data with no supervision (i.e., 70k data instances without labels) by enforcing semantic constraints in neural network training with NeurASP. In this section, we consider the same unsupervised learning problem for Sudoku while we represent the Sudoku problem in CNF and use L_{cnf} to enforce logical constraints during training.

We use a CNF theory for 9×9 Sudoku problems with $9^3 = 729$ atoms and 8991 clauses as described in Appendix C.6. This CNF can be represented by a matrix $\mathbf{C} \in \{-1, 0, 1\}^{8991 \times 729}$. The dataset consists of 70k data instances, 80%/20% for training/testing. Each data instance is a pair $\langle \mathbf{q}, \mathbf{l} \rangle$ where $\mathbf{q} \in \{0, 1, \ldots, 9\}^{81}$ denotes a 9×9 Sudoku board (0 denotes an empty cell) and $\mathbf{l} \in \{1, \ldots, 9\}^{81}$ denotes its solution ($\mathbf{l}$ is not used in NeurASP and our method during training). The non-zero values in $\mathbf{q}$ are treated as atomic facts F and we construct the matrix $\mathbf{F} \in \{0, 1\}^{81 \times 9}$ such that, for $i \in \{1, \ldots, 81\}$, the i-th row $\mathbf{F}[i, :]$ is the vector $\{0\}^9$ if $\mathbf{q}[i] = 0$ and is the one-hot vector for $\mathbf{q}[i]$ if $\mathbf{q}[i] \neq 0$. Then, the vector $\mathbf{f} \in \{0, 1\}^{729}$ is simply the flattening of $\mathbf{F}$. We feed $\mathbf{q}$ into the CNN and obtain the output $\mathbf{x} \in [0, 1]^{729}$. Finally, the prediction $\mathbf{v} \in \{0, 1\}^{729}$ is obtained as $\mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$, and the total loss function $\mathcal{L}$ we used is $\mathcal{L} = L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + 0.1 \times L_{bound}(\mathbf{x})$.

Table 3 compares the training time and the (whole-board) test accuracies with and without the inference trick (Acc_w and Acc_{wo}, resp.) using $b_p(x)$+iSTE against NeurASP and baseline CNN (Park, 2018). In each experiment, the same CNN is trained with only 70k (labeled/unlabeled) data instances from (Yang et al., 2020) with an average of 43 given digits in a puzzle (min: 26, max: 77). As we can see, our method outperforms NeurASP in both accuracy and time. Accuracy-wise, the CNN model trained using CL-STE is 27.2% more accurate than the CNN model trained using NeurASP when we use the inference trick. Training time-wise, CL-STE is 16 times faster than NeurASP because we directly encode semantic constraints in a loss function, which saves the time to call a symbolic engine externally (e.g., CLINGO to enumerate all stable models as in NeurASP).

Table 4 compares CNN+CL-STE with SATNet trained on Park 70k and tested on both Park 70k and Palm Sudoku dataset (Palm et al., 2018). While CNN is less tailored to logical reasoning than SATNet, our experiment shows that, when it is trained via CL-STE, it performs better than SATNet.

Table 4. SATNet vs. CNN+CL-STE

Method	Train Data (Supv)	Test Data	#Given	Test Accuracy
SATNet	Park 70k (Full)	Park 70k	26-77	67.78%
		Palm	17-34	6.76%
CNN+CL-STE	Park 70k (No)	Park 70k	26-77	93.70%
		Palm	17-34	27.37%

5.4. GNN + Constraint Loss for Sudoku

This section investigates if a GNN training can be improved with the constraint loss functions with STE by utilizing already known constraints without always relying on the labeled data. We consider the Recurrent Relational Network (RRN) (Palm et al., 2018), a state-of-the-art GNN for multi-step relational reasoning that achieves 96.6% accuracy for hardest Sudoku problems by training on 180k labeled data instances. Our focus here is to make RRN learn more effectively using fewer data by injecting known constraints.

The training dataset in (Palm et al., 2018) contains 180k data instances evenly distributed in 18 difficulties with 17-34 given numbers. We use a small subset of this dataset with random sampling. Given a data instance $\langle \mathbf{q}, \mathbf{l} \rangle$ where $\mathbf{q} \in \{0, 1, \ldots, 9\}^{81}$ denotes a 9×9 Sudoku board and $\mathbf{l} \in \{1, \ldots, 9\}^{81}$ denotes its solution, RRN takes $\mathbf{q}$ as input and, after 32 iterations of message passing, outputs 32 matrices of probabilities $\mathbf{X}_s \in \mathbf{R}^{81 \times 9}$ where $s \in \{1, \ldots, 32\}$; for example, $\mathbf{X}_1$ is the RRN prediction after 1 message passing step.

The baseline loss is the sum of the cross-entropy losses between prediction $\mathbf{X}_s$ and label $\mathbf{l}$ for all s.

We evaluate if using constraint loss could further improve the performance of RRN with the same labeled data. We use the same L_{cnf} and L_{bound} defined in CNN (with weights 1 and 0.1, resp.), which are applied to $\mathbf{X}_1$ only so that the RRN could be trained to deduce new digits in a single message passing step. We also use a continuous regularizer L_{sum} below to regularize every $\mathbf{X}_s$ that "the sum of the 9

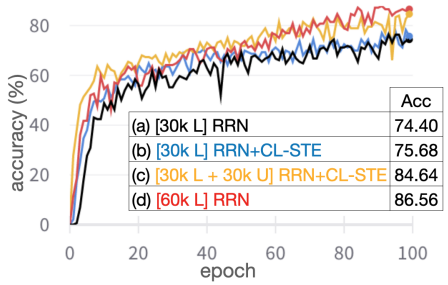

Figure 4. Test accuracy on the same randomly sampled 1k data from Palm Sudoku dataset when trained with RRN(+STE) with 30k to 60k [L]abeled/[U]nlabeled data

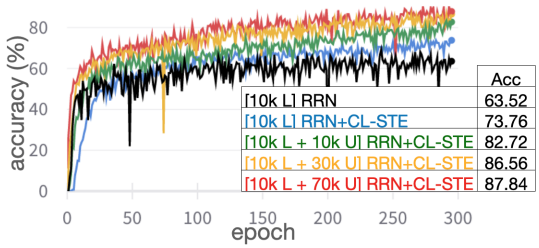

Figure 5. Semi-supervised learning with RRN+STE on Sudoku using only 10k labeled data and varying numbers of unlabeled data from Palm dataset for training and using the same randomly sampled 1k data for testing

probabilities in $\mathbf{X}_s$ in the same row/column/box must be 1":

$$L_{sum} = \sum_{\substack{s\in\{1,\ldots,32\}\\ i\in\{row,col,box\}}} avg((sum(\mathbf{X}_s^i) - 1)^2).$$

Here, $avg(X)$ and $sum(X)$ compute the average and sum of all elements in X along its last dimension; $\mathbf{X}_s^{row}, \mathbf{X}_s^{col}, \mathbf{X}_s^{box} \in \mathbb{R}^{81\times 9}$ are reshaped copies of $\mathbf{X}_s$ such that each row in, for example, $\mathbf{X}_s^{row}$ contains 9 probabilities for atoms $a(1, C, N), \ldots, a(9, C, N)$ for some C and N.

Figure 4 compares the test accuracy of the RRN trained for 100 epochs under 4 settings: (a) the RRN trained with baseline loss using 30k labeled data; (b) the RRN trained with both baseline loss and constraint losses (L_{sum}, L_{cnf}, and L_{bound}) using the same 30k labeled data; (c) the same setting as (b) with additional 30k unlabeled data; (d) same as (a) with additional 30k labeled data. Comparing (a) and (b) indicates the effectiveness of the constraint loss using the same number of labeled data; comparison between (b) and (c) indicates even with the same number of labeled data but adding unlabeled data could increase the accuracy (due to the constraint loss); comparison between (c) and (d) shows that the effectiveness of the constraint loss is comparable to adding additional 30k labels.

Figure 5 assesses the effect of constraint loss using fixed 10k labeled data and varying numbers (10k, 30k, 70k) of unlabeled data. We see that the baseline RRN trained with 10k labeled data ([10k L] RRN) has roughly saturated while the other methods are still slowly improving the accuracy. Training with the same number of labeled data but adding more unlabeled data makes the trained RRN achieve higher test accuracy, indicating that the constraint loss is effective in training even when the labels are unavailable.

5.5. Discussion

Regarding **Q1**, Figure 3, Tables 1 and 2 show that our method achieves comparable accuracy with existing neuro-symbolic formalisms but is much more scalable. Regarding **Q2**, Table 3 and Figures 4 and 5 illustrate our method could be used for unsupervised and semi-supervised learning by utilizing the constraints underlying the data. Regarding **Q3**, we applied constraint loss to MLP, CNN, and GNN, and observed that it improves the existing neural networks' prediction accuracy.

As we noted, the gradient computation in other neuro-symbolic approaches, such as NeurASP, DeepProbLog, and NeuroLog, requires external calls to symbolic solvers to compute stable models or proofs for every data instance, which takes a long time. These approaches may give better quality gradients to navigate to feasible solutions, but their gradient computations are associated with NP-hardness (the worst case exponential size of SDD, computing all proofs or stable models). In comparison, CL-STE treats each clause independently and locally to accumulate small pieces of gradients, allowing us to leverage GPUs and batch training as in the standard deep learning. The method resembles local search and deduction in SAT, and the gradients may not reflect the global property but could be computed significantly faster. Indeed, together with the gradient signals coming from the data, our method works well even when logical constraints are hard to satisfy, e.g., in training a neural network to solve Sudoku where a single feasible solution lies among 9^{47} to 9^{64} candidates when 17-34 digits are given.

6. Conclusion

Constraint loss helps neural networks learn with fewer data, but the state-of-the-art methods require combinatorial computation to compute gradients. By leveraging STE, we demonstrate the feasibility of more scalable constraint learning in neural networks. Also, we showed that GNNs could learn with fewer (labeled) data by utilizing known constraints. Based on the formal properties of the CNF constraint loss and the promising initial experiments here, the next step is to apply the method to larger-scale experiments.

Acknowledgements

We are grateful to Adam Ishay and the anonymous referees for their useful comments. This work was partially supported by the National Science Foundation under Grant IIS-2006747.

References

Arakelyan, E., Daza, D., Minervini, P., and Cochez, M. Complex query answering with neural link predictors. In *International Conference on Learning Representations*, 2020.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Besold, T. R., Garcez, A. d., Bader, S., Bowman, H., Domingos, P., Hitzler, P., Kühnberger, K.-U., Lamb, L. C., Lowd, D., Lima, P. M. V., et al. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*, 2017.

Courbariaux, M., Bengio, Y., and David, J.-P. Binaryconnect: training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pp. 3123–3131, 2015.

Darwiche, A. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, pp. 819, 2011.

De Raedt, L., Manhaeve, R., Dumancic, S., Demeester, T., and Kimmig, A. Neuro-symbolic = neural + logical + probabilistic. In *Proceedings of the 2019 International Workshop on Neural- Symbolic Learning and Reasoning*, pp. 4, 2019. URL https://sites.google.com/view/nesy2019/home.

Diligenti, M., Gori, M., and Sacca, C. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165, 2017.

Garcez, A. d., Gori, M., Lamb, L. C., Serafini, L., Spranger, M., and Tran, S. N. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *arXiv preprint arXiv:1905.06088*, 2019.

Kim, J., Park, C., Jung, H.-J., and Choe, Y. Plug-in, trainable gate for streamlining arbitrary neural networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

Lamb, L. C., Garcez, A., Gori, M., Prates, M., Avelar, P., and Vardi, M. Graph neural networks meet neural-symbolic computing: A survey and perspective. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 4877–4884, 2020.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. Deepproblog: Neural probabilistic logic programming. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 3749–3759, 2018.

Mao, J., Gan, C., Kohli, P., Tenenbaum, J. B., and Wu, J. The neuro-symbolic concept learner: interpreting scenes, words, and sentences from natural supervision. In *Proceedings of International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJgMlhRctm.

Marra, G., Dumančić, S., Manhaeve, R., and De Raedt, L. From statistical relational to neural symbolic artificial intelligence: a survey. *arXiv preprint arXiv:2108.11451*, 2021.

Palm, R., Paquet, U., and Winther, O. Recurrent relational networks. In *Proceedings of Advances in Neural Information Processing Systems*, pp. 3368–3378, 2018.

Park, K. Can convolutional neural networks crack sudoku puzzles? https://github.com/Kyubyong/sudoku, 2018.

Pogančić, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

Pogancic, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. Differentiation of blackbox combinatorial solvers. In *ICLR*, 2020.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Roychowdhury, S., Diligenti, M., and Gori, M. Regularizing deep networks with prior knowledge: A constraint-based approach. 222:106989, 2021.

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations (ICLR 2019)*, 2019.

Serafini, L. and Garcez, A. d. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*, 2016.

Simons, T. and Lee, D.-J. A review of binarized neural networks. *Electronics*, 8(6):661, 2019.

Tang, W., Hua, G., and Wang, L. How to train a compact binary neural network with high accuracy? In *Thirty-First AAAI conference on artificial intelligence*, 2017.

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

Wang, P.-W., Donti, P. L., Wilder, B., and Kolter, Z. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2019.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Van den Broeck, G. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, July 2018. URL http://starai.cs.ucla.edu/papers/XuICML18.pdf.

Yang, Z., Ishay, A., and Lee, J. NeurASP: Embracing neural networks into answer set programming. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1755–1762, 2020. doi: 10.24963/ijcai.2020/243.

Yin, P., Lyu, J., Zhang, S., Osher, S., Qi, Y., and Xin, J. Understanding straight-through estimator in training activation quantized neural nets. In *International Conference on Learning Representations*, 2019.

Appendix

The appendix contains the proofs of all propositions and more details about the experiments in the main body as well as additional experiments.

A. CNF Loss with Basic Math Operations

In Section 4, we defined the CNF loss using broadcasting, which is a common technique used for performing arithmetic operations between tensors having different shapes.[5] We could also give the definition of the CNF without referring to broadcasting as follows.

Consider a propositional signature $\sigma = \{p_1, \ldots, p_n\}$. Recall that we have

- the matrix $\mathbf{C} \in \{-1, 0, 1\}^{m \times n}$ to represent the CNF theory such that $\mathbf{C}[i, j]$ is 1 (-1, resp.) if p_j ($\neg p_j$, resp.) belongs to the i-th clause in the theory, and is 0 if neither p_j nor $\neg p_j$ belongs to the clause;
- the vector $\mathbf{f} \in \{0, 1\}^n$ to represent F such that $\mathbf{f}[j]$ is 1 if $p_j \in F$ and is 0 otherwise; and
- the vector $\mathbf{v} \in \{0, 1\}^n$ to represent v such that $\mathbf{v}[j]$ is 1 if $v(p_j) = \text{TRUE}$, and is 0 if $v(p_j) = \text{FALSE}$.

Using $\mathbf{C}$, $\mathbf{v}$, and $\mathbf{f}$, we define the CNF loss $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ with basic math operations as follows where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

$$
\begin{aligned}
\mathbf{L}_f[i,j] &= \mathbf{C}[i,j] \times \mathbf{f}[j] \\
\mathbf{L}_v[i,j] &= \mathbb{1}_{\{1\}}(\mathbf{C}[i,j]) \times \mathbf{v}[j] \; + \\
&\quad \mathbb{1}_{\{-1\}}(\mathbf{C}[i,j]) \times (1 - \mathbf{v}[j]) \\
\mathbf{deduce}[i] &= \mathbb{1}_{\{1\}}\Big(\sum_j \big(|\mathbf{C}[i,j]|\big) \; - \\
&\quad \sum_j \big(\mathbb{1}_{\{-1\}}(\mathbf{L}_f[i,j])\big)\Big) \\
\mathbf{unsat}[i] &= \prod_j \big(1 - \mathbf{L}_v[i,j]\big) \\
\mathbf{keep}[i] &= \sum_j \Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,j]) \times (1 - \mathbf{L}_v[i,j]) + \\
&\quad \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,j]) \odot \mathbf{L}_v[i,j]\Big)
\end{aligned}
$$

[5] https://towardsdatascience.com/broadcasting-in-numpy-58856f926d73

$$
\begin{aligned}
L_{deduce} &= \sum_i \big(\mathbf{deduce}[i] \times \mathbf{unsat}[i]\big) \\
L_{unsat} &= \frac{1}{n} \sum_i \big(\mathbb{1}_{\{1\}}(\mathbf{unsat}[i]) \times \mathbf{unsat}[i]\big) \\
L_{sat} &= \frac{1}{n} \sum_i \big(\mathbb{1}_{\{0\}}(\mathbf{unsat}[i]) \times \mathbf{keep}[i]\big)
\end{aligned}
$$

$$
L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) = L_{deduce} + L_{unsat} + L_{sat}.
$$

B. Proofs

Proposition 3.1 When K approaches ∞ and $|g(x)| \leq c$ for a constant c, the value of $\widetilde{b}^K(x)$ converges to $b(x)$.

$$
\lim_{K \to \infty} \widetilde{b}^K(x) = b(x)
$$

The gradient $\frac{\partial \widetilde{b}^K(x)}{\partial x}$, whenever defined, is exactly the iSTE of $\frac{\partial b(x)}{\partial x}$ if $g(x) = 1$, or the sSTE of $\frac{\partial b(x)}{\partial x}$ if

$$
g(x) = \begin{cases} 1 & \text{if } -1 \leq x \leq 1 \\ 0 & \text{otherwise.} \end{cases}
$$

[Remark]: Proposition 3.1 in our paper is similar to proposition 1 in (Kim et al., 2020) but not the same. For the value of $\widetilde{b}^K(x)$, we don't have a condition that $g'(x)$ should be bounded. For the gradient of $\widetilde{b}^K(x)$, we have a stronger statement specific for STEs and don't have the condition for K approaching ∞.

Proof. Recall the definition of $\widetilde{b}^K(x)$

$$
\widetilde{b}^K(x) = b(x) + s^K(x) g(x)
$$

where K is a constant; $s^K(x) = \frac{Kx - \lfloor Kx \rfloor}{K}$ is a gradient tweaking function whose value is less than $\frac{1}{K}$ and whose gradient is always 1 whenever differentiable; and $g(x)$ is a gradient shaping function.

[First], we will prove $\lim_{K \to \infty} \widetilde{b}^K(x) = b(x)$. Since $\widetilde{b}^K(x) = b(x) + s^K(x) g(x)$, it's equivalent to proving

$$
\lim_{K \to \infty} s^K(x) g(x) = 0
$$

Since $s^K(x) = \frac{Kx - \lfloor Kx \rfloor}{K}$ and $0 \leq Kx - \lfloor Kx \rfloor \leq 1$, we know $0 \leq s^K(x) \leq \frac{1}{K}$. Since $|g(x)| \leq c$ where c is a constant, $-\frac{c}{K} \leq s^K(x) g(x) \leq \frac{c}{K}$. Thus $0 \leq \lim_{K \to \infty} s^K(x) g(x) \leq 0$, and consequently, $\lim_{K \to \infty} s^K(x) g(x) = 0$.

[Second], we will prove

- when $g(x) = 1$ and $s(x) = x$ (i.e., under iSTE),

$$\frac{\partial \widetilde{b}^K(x)}{\partial x} = \begin{cases} \frac{\partial s(x)}{\partial x} & \text{(if } Kx \neq \lfloor Kx \rfloor\text{)} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Let's prove some general properties of the gradients. Since $s(x) = x$, $g(x) = 1$, and $s^K(x) = \frac{Kx - \lfloor Kx \rfloor}{K}$,

- $\frac{\partial s(x)}{\partial x} = 1$, $\frac{\partial g(x)}{\partial x} = 0$, and
- $\frac{\partial s^K(x)}{\partial x} = 1$ whenever differentiable (i.e., whenever $Kx \neq \lfloor Kx \rfloor$).

Then,

$$\begin{aligned} \frac{\partial \widetilde{b}^K(x)}{\partial x} &= \frac{\partial (b(x) + s^K(x) g(x))}{\partial x} \\ &= \frac{\partial (s^K(x) \times g(x))}{\partial x} \\ &= \frac{\partial s^K(x)}{\partial x} \\ &= \begin{cases} 1 & \text{(if } Kx \neq \lfloor Kx \rfloor\text{)} \\ \text{undefined} & \text{otherwise.} \end{cases} \end{aligned}$$

[Third], we will prove

- when $g(x) = 1$ if $-1 \leq x \leq 1$ and $g(x) = 0$ otherwise, and $s(x) = min(max(x, -1), 1)$ (i.e., under sSTE),

$$\frac{\partial \widetilde{b}^K(x)}{\partial x} = \begin{cases} \frac{\partial s(x)}{\partial x} & \text{(if } Kx \neq \lfloor Kx \rfloor\text{)} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Let's prove some general properties of the gradients. Since $s(x) = min(max(x, -1), 1)$, $g(x) = 1$ if $-1 \leq x \leq 1$ and $g(x) = 0$ otherwise, and $s^K(x) = \frac{Kx - \lfloor Kx \rfloor}{K}$,

- $\frac{\partial s(x)}{\partial x} = 1$ if $-1 \leq x \leq 1$ and $\frac{\partial s(x)}{\partial x} = 0$ otherwise,
- $\frac{\partial g(x)}{\partial x} = 0$, and
- $\frac{\partial s^K(x)}{\partial x} = 1$ whenever differentiable (i.e., whenever $Kx \neq \lfloor Kx \rfloor$).

Then,

$$\begin{aligned} \frac{\partial \widetilde{b}^K(x)}{\partial x} &= \frac{\partial (b(x) + s^K(x) g(x))}{\partial x} \\ &= \frac{\partial (s^K(x) \times g(x))}{\partial x} \\ &= g(x) \times \frac{\partial s^K(x)}{\partial x} + s^K(x) \times \frac{\partial g(x)}{\partial x} \\ &= g(x) \times \frac{\partial s^K(x)}{\partial x} \\ &= \begin{cases} g(x) & \text{(if } Kx \neq \lfloor Kx \rfloor\text{)} \\ \text{undefined} & \text{otherwise.} \end{cases} \\ &= \begin{cases} \frac{\partial s(x)}{\partial x} & \text{(if } Kx \neq \lfloor Kx \rfloor\text{)} \\ \text{undefined} & \text{otherwise.} \end{cases} \end{aligned}$$

□

Proposition 4.2 Given a CNF theory C, a set F of atoms, and a truth assignment v such that $v \models F$, let $\mathbf{C}, \mathbf{f}, \mathbf{v}$ denote their matrix/vector representations, respectively. Let $C_{deduce} \subseteq C$ denote the set of Horn clauses H in C such that all but one literal in H are of the form $\neg p$ where $p \in F$. Then

- the minimum values of L_{deduce}, L_{unsat}, L_{sat}, and $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ are 0;
- $v \models C_{deduce}$ iff L_{deduce} is 0;
- $v \models C$ iff L_{unsat} is 0 iff $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ is 0.

Proof. Recall the definition of L_{cnf}

$$\begin{aligned} \mathbf{L}_f &= \mathbf{C} \odot \mathbf{f} \\ \mathbf{L}_v &= \mathbb{1}_{\{1\}}(\mathbf{C}) \odot \mathbf{v} + \mathbb{1}_{\{-1\}}(\mathbf{C}) \odot (1 - \mathbf{v}) \\ \mathbf{deduce} &= \mathbb{1}_{\{1\}}\Big(sum(\mathbf{C} \odot \mathbf{C}) - sum(\mathbb{1}_{\{-1\}}(\mathbf{L}_f))\Big) \\ \mathbf{unsat} &= prod(1 - \mathbf{L}_v) \\ \mathbf{keep} &= sum(\mathbb{1}_{\{1\}}(\mathbf{L}_v) \odot (1 - \mathbf{L}_v) + \mathbb{1}_{\{0\}}(\mathbf{L}_v) \odot \mathbf{L}_v) \\ L_{deduce} &= sum(\mathbf{deduce} \odot \mathbf{unsat}) \\ L_{unsat} &= avg(\mathbb{1}_{\{1\}}(\mathbf{unsat}) \odot \mathbf{unsat}) \\ L_{sat} &= avg(\mathbb{1}_{\{0\}}(\mathbf{unsat}) \odot \mathbf{keep}) \end{aligned}$$

$$L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) = L_{deduce} + L_{unsat} + L_{sat}$$

and the definitions of $\mathbf{C}, \mathbf{f}, \mathbf{v}$ below.

- the matrix $\mathbf{C}$ is in $\{-1, 0, 1\}^{m \times n}$ such that $\mathbf{C}[i, j]$ is 1 (-1, resp.) if p_j ($\neg p_j$, resp.) belongs to the i-th clause, and is 0 if neither p_j nor $\neg p_j$ belongs to the clause;

- the vector $\mathbf{f}$ is in $\{0,1\}^n$ to represent F such that $\mathbf{f}[j]$ is 1 if $p_j \in F$ and is 0 otherwise; and
- the vector $\mathbf{v}$ is in $\{0,1\}^n$ to represent v such that $\mathbf{v}[j]$ is 1 if $v(p_j) = \text{TRUE}$, and is 0 if $v(p_j) = \text{FALSE}$.

We will prove each bullet in Proposition 4.2 as follows.

1. **[First]**, we will prove
   - $\mathbf{L}_f$ is the matrix in $\{-1,0,1\}^{m\times n}$ such that (i) $\mathbf{L}_f[i,j] = 1$ iff clause i contains literal p_j and $p_j \in F$; and (ii) $\mathbf{L}_f[i,j] = -1$ iff clause i contains literal $\neg p_j$ and $p_j \in F$.
   - $\mathbf{L}_v$ is the matrix in $\{0,1\}^{m\times n}$ such that $\mathbf{L}_v[i,j] = 1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal evaluates to TRUE under v.

   According to the definition, $\mathbf{L}_f[i,j] = \mathbf{C}[i,j] \times \mathbf{f}[j]$. Since $\mathbf{f}[j] \in \{0,1\}$, we have $\mathbf{L}_f[i,j] = 1$ iff "$\mathbf{C}[i,j] = 1$ and $\mathbf{f}[j] = 1$", and according to the definition of $\mathbf{C}$ and $\mathbf{f}$, we have $\mathbf{L}_f[i,j] = 1$ iff "clause i contains literal p_j and $p_j \in F$". Similarly, we have $\mathbf{L}_f[i,j] = -1$ iff "$\mathbf{C}[i,j] = -1$ and $\mathbf{f}[j] = 1$" iff "clause i contains literal $\neg p_j$ and $p_j \in F$".

   According to the definition, $\mathbf{L}_v[i,j] = \mathbb{1}_{\{1\}}(\mathbf{C})[i,j] \times \mathbf{v}[j] + \mathbb{1}_{\{-1\}}(\mathbf{C})[i,j] \times (1 - \mathbf{v}[j])$. Since $\mathbb{1}_{\{1\}}(\mathbf{C})[i,j]$ and $\mathbb{1}_{\{-1\}}(\mathbf{C})[i,j]$ cannot be 1 at the same time and $\mathbf{v}[j] \in \{0,1\}$, we have $\mathbf{L}_v[i,j] = 1$ iff "$\mathbf{C}[i,j] = 1$ and $\mathbf{v}[j] = 1$" or "$\mathbf{C}[i,j] = -1$ and $\mathbf{v}[j] = 0$". According to the definition of $\mathbf{C}$ and $\mathbf{v}$, we have $\mathbf{L}_v[i,j] = 1$ iff "clause i contains literal p_j, which evaluates to TRUE under v" or "clause i contains literal $\neg p_j$, which evaluates to TRUE under v".

   [Second], we will prove
   - $\mathbf{deduce}$ is the vector in $\{0,1\}^m$ such that $\mathbf{deduce}[i] = 1$ iff clause i has all but one literal of the form $\neg p_j$ such that $p_j \in F$.
   - $\mathbf{unsat}$ is the vector in $\{0,1\}^m$ such that $\mathbf{unsat}[i] = 1$ iff clause i evaluates to FALSE under v.
   - $\mathbf{keep}$ is the vector $\{0\}^m$.

   From the definition of $\mathbf{C}$, the matrix $\mathbf{C} \odot \mathbf{C}$ is in $\{0,1\}^{m\times n}$ such that the element at position (i,j) is 1 iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j. Since $sum(X)$ computes the sum of elements in each row of matrix X, for $i \in \{1,\ldots,m\}$ and $k \in \{1,\ldots,n\}$, $sum(\mathbf{C} \odot \mathbf{C})[i] = k$ iff clause i contains k literals. Recall that we proved that $\mathbf{L}_f[i,j] = -1$ iff "clause i contains literal $\neg p_j$ and $p_j \in F$". Consequently, $\mathbb{1}_{\{-1\}}(\mathbf{L}_f)$ is the matrix in $\{0,1\}^{m\times n}$ such that $\mathbb{1}_{\{-1\}}(\mathbf{L}_f)[i,j] = 1$ iff "clause i contains literal $\neg p_j$ and $p_j \in F$". As a result, $sum(\mathbf{C} \odot \mathbf{C}) - sum(\mathbb{1}_{\{-1\}}(\mathbf{L}_f))$ is the vector in $\{0,\ldots,n\}^m$ such that its i-th element is 1 iff "clause i contains all but one literal of the form $\neg p_j$ such that $p_j \in F$". Thus $\mathbf{deduce}$ is the vector in $\{0,1\}^m$ such that $\mathbf{deduce}[i] = 1$ iff "clause i has all but one literal of the form $\neg p_j$ such that $p_j \in F$".

   Since $prod(X)$ computes the product of elements in each row of matrix X, for $i \in \{1,\ldots,m\}$, $\mathbf{unsat}[i] = \prod_{j\in\{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])$. Recall that we proved that $\mathbf{L}_v$ is the matrix in $\{0,1\}^{m\times n}$ such that $\mathbf{L}_v[i,j] = 1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal evaluates to TRUE under v. Thus $\mathbf{unsat}[i] \in \{0,1\}$ and $\mathbf{unsat}[i] = 1$ iff "$\mathbf{L}_v[i,j] = 0$ for $j \in \{1,\ldots,n\}$" iff "for $j \in \{1,\ldots,n\}$, clause i either does not contain a literal for atom p_j or contains a literal for atom p_j while this literal evaluates to FALSE under v" iff "clause i evaluates to FALSE under v". In other words, $\mathbf{unsat}$ is the vector in $\{0,1\}^m$ such that $\mathbf{unsat}[i] = 1$ iff clause i evaluates to FALSE under v.

   Since $\mathbf{L}_v$ is the matrix in $\{0,1\}^{m\times n}$, for $i \in \{1,\ldots,m\}$ and $j \in \{1,\ldots,n\}$, $\mathbb{1}_{\{1\}}(\mathbf{L}_v)[i,j] = 1$ iff $\mathbf{L}_v[i,j] = 1$ iff $(1 - \mathbf{L}_v[i,j]) = 0$. Thus $\mathbb{1}_{\{1\}}(\mathbf{L}_v) \odot (1 - \mathbf{L}_v)$ is the matrix $\{0\}^{m\times n}$ of all zeros. Similarly, $\mathbb{1}_{\{0\}}(\mathbf{L}_v) \odot \mathbf{L}_v)$ is also the matrix $\{0\}^{m\times n}$. As a result, $\mathbf{keep}$ is the vector $\{0\}^m$.

   [Third], we will prove
   - L_{deduce} is an integer in $\{0,\ldots,m\}$ such that $L_{deduce} = k$ iff there are k clauses in C_{deduce} that are evaluated to FALSE under v.
   - L_{unsat} is a number in $\{0, \frac{1}{m}, \ldots, \frac{m}{m}\}$ such that $L_{unsat} = \frac{k}{m}$ iff there are k clauses that are evaluated to FALSE under v.
   - L_{sat} is 0.

   Recall that we proved that $\mathbf{deduce}$ is the vector in $\{0,1\}^m$ such that $\mathbf{deduce}[i] = 1$ iff clause i has all but one literal of the form $\neg p_j$ such that $p_j \in F$; and $\mathbf{unsat}$ is the vector in $\{0,1\}^m$ such that $\mathbf{unsat}[i] = 1$ iff clause i evaluates to FALSE under v. According to the definition of C_{deduce}, $\mathbf{deduce} \odot \mathbf{unsat}$ is the vector in $\{0,1\}^m$ such that its i-th element is 1 iff clause i is in C_{deduce} and evaluates to FALSE under v. As a result, L_{deduce} is an integer in $\{0,\ldots,m\}$ such that $L_{deduce} = k$ iff there are k clauses in C_{deduce} that are evaluated as FALSE under v.

   Since $\mathbf{unsat}$ is the vector in $\{0,1\}^m$ such that $\mathbf{unsat}[i] = 1$ iff clause i evaluates to FALSE under v, and since $\mathbf{unsat}[i] = 1$ iff $\mathbb{1}_{\{1\}}(\mathbf{unsat})[i] = 1$, we know the i-th element in $\mathbb{1}_{\{1\}}(\mathbf{unsat}) \odot \mathbf{unsat}$ is 1 iff clause i evaluates to FALSE under v. $L_{unsat} = avg(\mathbb{1}_{\{1\}}(\mathbf{unsat}) \odot \mathbf{unsat})$ is a number in

$\{0, \frac{1}{m}, \ldots, \frac{m}{m}\}$ such that $L_{unsat} = \frac{k}{m}$ iff there are k clauses that are evaluated as FALSE under v.

Recall that we proved that $\mathbf{keep}$ is the vector $\{0\}^m$. Thus $\mathbb{1}_{\{0\}}(\mathbf{unsat}) \odot \mathbf{keep}$ is the vector $\{0\}^m$. Thus L_{sat} is 0.

[Fourth], we will prove

- the minimum values of L_{deduce}, L_{unsat}, L_{sat}, $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ are 0.

Recall that we proved that L_{deduce} is an integer in $\{0, \ldots, m\}$, L_{unsat} is a number in $\{0, \frac{1}{m}, \ldots, \frac{m}{m}\}$, and L_{sat} is 0. It's obvious that the minimum values of L_{deduce}, L_{unsat}, and L_{sat} are 0. Since (i) $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) = L_{deduce} + L_{unsat} + L_{sat}$, (ii) $L_{deduce} = 0$ when all clauses in C_{deduce} are evaluated to TRUE under v, and (iii) $L_{unsat} = 0$ when all clauses in C are evaluated to TRUE under v, the minimum value of $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ is 0 and is achieved when all clauses in C are evaluated to TRUE under v.

2. We will prove

   - $v \models C_{deduce}$ iff L_{deduce} is 0.

   Recall that we proved that L_{deduce} is an integer in $\{0, \ldots, m\}$ such that $L_{deduce} = k$ iff there are k clauses in C_{deduce} that are evaluated as FALSE under v. Then L_{deduce} is 0 iff "there is no clause in C_{deduce} that evaluates to FALSE under v" iff "every clause in C_{deduce} evaluates to TRUE under v" iff $v \models C_{deduce}$.

3. We will prove

   - $v \models C$ iff L_{unsat} is 0 iff $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ is 0.

   Recall that we proved that L_{unsat} is a number in $\{0, \frac{1}{m}, \ldots, \frac{m}{m}\}$ such that $L_{unsat} = \frac{k}{m}$ iff there are k clauses that are evaluated as FALSE under v. Then L_{unsat} is 0 iff "there is no clause in C that evaluates to FALSE under v" iff $v \models C$.

   Assume L_{unsat} is 0. Then "there is no clause in C that evaluates to FALSE under v". Consequently, "there is no clause in C_{deduce} that is evaluated to FALSE under v". Recall that we proved that L_{deduce} is an integer in $\{0, \ldots, m\}$ such that $L_{deduce} = k$ iff there are k clauses in C_{deduce} that are evaluated as FALSE under v. Then L_{deduce} is 0. Since L_{sat} is 0, $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ is 0.

   Assume $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ is 0, which is the minimum value L_{cnf} can take. It is easy to see that L_{unsat} must be 0.

□

Proposition 4.3 Given a CNF theory C of m clauses and n atoms and a set F of atoms such that $C \cup F$ is satisfiable, let $\mathbf{C}, \mathbf{f}$ denote their matrix/vector representations, respectively. Given a neural network output $\mathbf{x} \in [0,1]^n$ denoting probabilities, we construct $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$ and a truth assignment v such that $v(p_j) =$ TRUE if $\mathbf{v}[j]$ is 1, and $v(p_j) =$ FALSE if $\mathbf{v}[j]$ is 0. Let $C_{deduce} \subseteq C$ denote the set of Horn clauses H in C such that all but one literal in H are of the form $\neg p$ and $p \in F$. Then, for any $j \in \{1, \ldots, n\}$,

1. if $p_j \in F$, all of $\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]}$, $\frac{\partial L_{unsat}}{\partial \mathbf{x}[j]}$, and $\frac{\partial L_{sat}}{\partial \mathbf{x}[j]}$ are zeros;
2. if $p_j \notin F$,

$$\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \begin{cases} -c & \text{if } c > 0 \text{ clauses in } C_{deduce} \\ & \text{contain literal } p_j; \\ c & \text{if } c > 0 \text{ clauses in } C_{deduce} \\ & \text{contain literal } \neg p_j; \\ 0 & \text{otherwise;} \end{cases}$$

$$\frac{\partial L_{unsat}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \frac{c_2 - c_1}{m}$$

$$\frac{\partial L_{sat}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \begin{cases} -\frac{c_3}{m} & \text{if } v \models p_j, \\ \frac{c_3}{m} & \text{if } v \not\models p_j. \end{cases}$$

where $\overset{iSTE}{\approx}$ stands for the equivalence of gradients under iSTE; c_1 (and c_2, resp.) is the number of clauses in C that are not satisfied by v and contain p_j (and $\neg p_j$, resp.); c_3 is the number of clauses in C that are satisfied by v and contain p_j or $\neg p_j$.

Proof. We will prove each bullet in Proposition 4.3 as follows.

1. Take any $k \in \{1, \ldots, n\}$, let's focus on $\mathbf{x}[k]$ and compute the gradient of $L \in \{L_{deduce}, L_{unsat}, L_{sat}\}$ to it with iSTE. According to the chain rule and since $\frac{\partial \mathbf{v}[i]}{\partial b_p(\mathbf{x})[j]} = 0$ for $i \neq j$, we have

$$\frac{\partial L}{\partial \mathbf{x}[k]} = \frac{\partial L}{\partial \mathbf{v}[k]} \times \frac{\partial \mathbf{v}[k]}{\partial b_p(\mathbf{x}[k])} \times \frac{\partial b_p(\mathbf{x}[k])}{\partial \mathbf{x}[k]}.$$

   Under iSTE, the last term $\frac{\partial b_p(\mathbf{x}[k])}{\partial \mathbf{x}[k]}$ is replaced with $\frac{\partial s(\mathbf{x}[k])}{\partial \mathbf{x}[k]} = \frac{\partial \mathbf{x}[k]}{\partial \mathbf{x}[k]} = 1$. Thus

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{x}[k]} &= \frac{\partial L}{\partial \mathbf{v}[k]} \times \frac{\partial \mathbf{v}[k]}{\partial b_p(\mathbf{x}[k])} \times \frac{\partial b_p(\mathbf{x}[k])}{\partial \mathbf{x}[k]} \\ &\overset{iSTE}{\approx} \frac{\partial L}{\partial \mathbf{v}[k]} \times \frac{\partial \mathbf{v}[k]}{\partial b_p(\mathbf{x}[k])} \quad \text{(under iSTE)} \\ &= \frac{\partial L}{\partial \mathbf{v}[k]} \times \frac{\partial(\mathbf{f}[k] + \mathbb{1}_{\{0\}}(\mathbf{f}[k]) \times b_p(\mathbf{x}[k]))}{\partial b_p(\mathbf{x}[k])} \\ &= \begin{cases} \frac{\partial L}{\partial \mathbf{v}[k]} & \text{if } \mathbf{f}[k] = 0, \\ 0 & \text{if } \mathbf{f}[k] = 1. \end{cases} \end{aligned}$$

Since $\mathbf{f}[k] = 1$ iff $p_k \in F$, if $p_k \in F$, then all of $\frac{\partial L_{deduce}}{\partial \mathbf{x}[k]}$, $\frac{\partial L_{unsat}}{\partial \mathbf{x}[k]}$, and $\frac{\partial L_{sat}}{\partial \mathbf{x}[k]}$ are zeros.

2. Recall the definition of L_{cnf}

$$\begin{aligned}
\mathbf{L}_f &= \mathbf{C} \odot \mathbf{f} \\
\mathbf{L}_v &= \mathbb{1}_{\{1\}}(\mathbf{C}) \odot \mathbf{v} + \mathbb{1}_{\{-1\}}(\mathbf{C}) \odot (1 - \mathbf{v}) \\
\mathbf{deduce} &= \mathbb{1}_{\{1\}}\Big(sum(\mathbf{C} \odot \mathbf{C}) - sum(\mathbb{1}_{\{-1\}}(\mathbf{L}_f))\Big) \\
\mathbf{unsat} &= prod(1 - \mathbf{L}_v) \\
\mathbf{keep} &= sum(\mathbb{1}_{\{1\}}(\mathbf{L}_v) \odot (1 - \mathbf{L}_v) + \mathbb{1}_{\{0\}}(\mathbf{L}_v) \odot \mathbf{L}_v) \\
L_{deduce} &= sum(\mathbf{deduce} \odot \mathbf{unsat}) \\
L_{unsat} &= avg(\mathbb{1}_{\{1\}}(\mathbf{unsat}) \odot \mathbf{unsat}) \\
L_{sat} &= avg(\mathbb{1}_{\{0\}}(\mathbf{unsat}) \odot \mathbf{keep})
\end{aligned}$$

$$L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) = L_{deduce} + L_{unsat} + L_{sat}$$

We know $p_k \notin F$ iff $\mathbf{f}[k] = 0$. As proved in the first bullet, for $L \in \{L_{deduce}, L_{unsat}, L_{sat}\}$, if $p_k \notin F$, then $\frac{\partial L}{\partial \mathbf{x}[k]} \overset{iSTE}{\approx} \frac{\partial L}{\partial \mathbf{v}[k]}$. We further analyze the value of $\frac{\partial L}{\partial \mathbf{v}[k]}$ for each L under the condition that $\mathbf{f}[k] = 0$.

[L_{deduce}] According to the definition,

$$\begin{aligned}
&L_{deduce} \\
&= \sum_{i \in \{1,\dots,m\}} \Big(\mathbf{deduce}[i] \times \mathbf{unsat}[i]\Big) \\
&= \sum_{i \in \{1,\dots,m\}} \Big(\mathbf{deduce}[i] \times \prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])\Big)
\end{aligned}$$

$$\begin{aligned}
&\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} \\
&= \sum_{i \in \{1,\dots,m\}} \frac{\partial\Big(\mathbf{deduce}[i] \times \prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])\Big)}{\partial \mathbf{v}[k]} \\
&= \sum_{i \in \{1,\dots,m\}} \Big(\frac{\partial \mathbf{deduce}[i]}{\partial \mathbf{v}[k]} \times \prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j]) + \\
&\qquad \mathbf{deduce}[i] \times \frac{\partial \prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\Big)
\end{aligned}$$

Since $\mathbf{deduce}$ is the result of an indicator function, $\frac{\partial \mathbf{deduce}[i]}{\partial \mathbf{v}[k]} = 0$. Then,

$$\begin{aligned}
&\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} \\
&= \sum_{i \in \{1,\dots,m\}} \Big(\mathbf{deduce}[i] \times \frac{\partial \prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\Big).
\end{aligned}$$

Let $U \subseteq \{1, \dots, m\}$ denote the set of indices of all clauses in C_{deduce}. Since $\mathbf{deduce}[i] = 1$ iff $i \in U$,

$$\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} = \sum_{i \in U} \Big(\frac{\partial \prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\Big).$$

Let $G_{i,k}$ denote $\frac{\partial \prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}$. Then

$$\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} = \sum_{i \in U} G_{i,k}.$$

Let's analyze the value of $G_{i,k}$ where $i \in U$ and $k \in \{1, \dots, n\}$ such that $\mathbf{f}[k] = 0$. According to the product rule below,

$$\frac{d}{dx}\left[\prod_{i=1}^{k} f_i(x)\right] = \left(\prod_{i=1}^{k} f_i(x)\right)\left(\sum_{i=1}^{k} \frac{f_i'(x)}{f_i(x)}\right)$$

we have

$$\begin{aligned}
&G_{i,k} \\
&= \frac{\partial \prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]} \\
&= \left(\prod_{j \in \{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])\right) \times \sum_{j \in \{1,\dots,n\}} \frac{\frac{\partial (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}}{1 - \mathbf{L}_v[i,j]}
\end{aligned}$$

Since $\mathbf{L}_v[i,j] = \mathbb{1}_{\{1\}}(\mathbf{C})[i,j] \times \mathbf{v}[j] + \mathbb{1}_{\{-1\}}(\mathbf{C})[i,j] \times (1 - \mathbf{v}[j])$, we know

- **(a)** for $j \in \{1, \dots, n\}$ such that $j \neq k$, $\frac{\partial (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]} = 0$ and $\frac{\partial \mathbf{L}_v[i,j]}{\partial \mathbf{v}[k]} = 0$;
- **(b)** when clause i doesn't contain a literal for atom p_k, $\frac{\partial (1 - \mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]} = 0$ and $\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]} = 0$;
- **(c)** when clause i contains literal p_k, $\frac{\partial (1 - \mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]} = -1$ and $\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]} = 1$;
- **(d)** when clause i contains literal $\neg p_k$, $\frac{\partial (1 - \mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]} = 1$ and $\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]} = -1$.

We will refer to the above 4 bullets with their identifiers.

Since $i \in U$, we know clause i has all but one literal of the form $\neg p_j$ such that $p_j \in F$. Since $\mathbf{f}[k] = 0$, we know $p_k \notin F$. Then, when clause i contains literal p_k or $\neg p_k$, all other literals in clause i must be of the form $\neg p_j$ where $p_j \in F$. For every literal $\neg p_j$ in clause i where $j \neq k$, we know $p_j \in F$, thus $\mathbf{f}[j] = 1$; since $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$, then $\mathbf{v}[j] = 1$; consequently, the literal $\neg p_j$ evaluates to FALSE under v. Recall that $\mathbf{L}_v[i,j] \in \{0,1\}$, and $\mathbf{L}_v[i,j] = 1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal evaluates to TRUE under v, then we know

- when $i \in U$, $\mathbf{f}[k] = 0$, and clause i contains literal p_k or $\neg p_k$, $\mathbf{L}_v[i,j] = 0$ for $j \in \{1,\dots,n\}$ such that $j \neq k$.

Then we have

$$
\begin{aligned}
&G_{i,k}\\
=&\left(\prod_{j\in\{1,\dots,n\}}(1-\mathbf{L}_v[i,j])\right)\times\sum_{j\in\{1,\dots,n\}}\frac{\frac{\partial(1-\mathbf{L}_v[i,j])}{\partial\mathbf{v}[k]}}{1-\mathbf{L}_v[i,j]}\\
=&\left(\prod_{j\in\{1,\dots,n\}}(1-\mathbf{L}_v[i,j])\right)\times\frac{\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial\mathbf{v}[k]}}{1-\mathbf{L}_v[i,k]}\ \text{(due to \textbf{(a)})}\\
=&\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial\mathbf{v}[k]}\times\prod_{\substack{j\in\{1,\dots,n\}\\ j\neq k}}(1-\mathbf{L}_v[i,j])\\
=&\begin{cases}0 & \text{if clause } i \text{ doesn't contain a literal}\\ & \text{for atom } p_k \text{ (due to \textbf{(b)})}\\ -1 & \text{if clause } i \text{ contains a literal } p_k \text{ (due to \textbf{(c)})}\\ 1 & \text{if clause } i \text{ contains a literal } \neg p_k \text{ (due to \textbf{(d)})}\end{cases}
\end{aligned}
$$

Since $i \in U$ and $\mathbf{f}[k] = 0$, when clause i contains a literal l_k for atom p_k, we know $F \not\models l_j$ for every literal l_j in clause i such that $j \neq k$. Since $C \cup F$ is satisfiable, we know $C \cup F \models l_k$ and there cannot be two clauses in C_{deduce} containing different literals p_k and $\neg p_k$. Thus, when $\mathbf{f}[k] = 0$,

$$
\begin{aligned}
&\frac{\partial L_{deduce}}{\partial\mathbf{v}[k]}\\
=&\sum_{i\in U}G_{i,k}\\
=&\begin{cases}-c & \text{if } c>0 \text{ clauses in } C_{deduce} \text{ contain literal } p_k,\\ c & \text{if } c>0 \text{ clauses in } C_{deduce} \text{ contain literal } \neg p_k,\\ 0 & \text{otherwise.}\end{cases}
\end{aligned}
$$

Note that the first 2 cases above are disjoint since there cannot be two clauses in C_{deduce} containing different literals p_k and $\neg p_k$.

Finally, if $p_k \notin F$,

$$
\begin{aligned}
&\frac{\partial L_{deduce}}{\partial\mathbf{x}[k]}\\
\overset{iSTE}{\approx}&\frac{\partial L_{deduce}}{\partial\mathbf{v}[k]}\\
=&\begin{cases}-c & \text{if } c>0 \text{ clauses in } C_{deduce}\\ & \text{contain literal } p_k;\\ c & \text{if } c>0 \text{ clauses in } C_{deduce}\\ & \text{contain literal } \neg p_k;\\ 0 & \text{otherwise;}\end{cases}
\end{aligned}
$$

$[L_{unsat}]$ According to the definition,

$$
\begin{aligned}
L_{unsat}=&avg(\mathbb{1}_{\{1\}}(\mathbf{unsat})\odot\mathbf{unsat})\\
=&\frac{1}{m}\sum_{i\in\{1,\dots,m\}}\Big(\mathbb{1}_{\{1\}}(\mathbf{unsat}[i])\times\\
&\prod_{j\in\{1,\dots,n\}}(1-\mathbf{L}_v[i,j])\Big)
\end{aligned}
$$

Recall that we proved that $\mathbb{1}_{\{1\}}(\mathbf{unsat})[i] \in \{0,1\}$ is the output of an indicator function whose value is 1 iff clause i evaluates to FALSE under v. Let $U \subseteq \{1,\dots,m\}$ denote the set of indices of clauses in C that are evaluated as FALSE under v.

$$
L_{unsat}=\frac{1}{m}\sum_{i\in U}\Big(\prod_{j\in\{1,\dots,n\}}(1-\mathbf{L}_v[i,j])\Big)
$$

Then the gradient of L_{unsat} w.r.t. $\mathbf{v}[k]$ is

$$
\frac{\partial L_{unsat}}{\partial\mathbf{v}[k]}=\frac{1}{m}\sum_{i\in U}\Big(\frac{\partial\prod_{j\in\{1,\dots,n\}}(1-\mathbf{L}_v[i,j])}{\partial\mathbf{v}[k]}\Big).
$$

Recall that $\mathbf{L}_v[i,j] \in \{0,1\}$, and $\mathbf{L}_v[i,j] = 1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal evaluates to TRUE under v. When $i \in U$, clause i evaluates to FALSE under v. Thus when $i \in U$, all literals in clause i must be evaluated as FALSE under v, and consequently, $\mathbf{L}_v[i,j] = 0$ for all $j \in \{1,\dots,m\}$. Then

$$
\begin{aligned}
\frac{\partial L_{unsat}}{\partial\mathbf{v}[k]}=&\frac{1}{m}\sum_{i\in U}\Big(\frac{\partial\prod_{j\in\{1,\dots,n\}}(1-\mathbf{L}_v[i,j])}{\partial\mathbf{v}[k]}\Big)\\
=&\frac{1}{m}\sum_{i\in U}\Big(\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial\mathbf{v}[k]}\Big)\ \text{(due to \textbf{(a)})}\\
=&\frac{c_2-c_1}{m}\ \text{(due to \textbf{(b), (c), (d)})}
\end{aligned}
$$

where c_1 (and c_2, resp.) is the number of clauses in U that contain p_k (and $\neg p_k$, resp.). Finally, if $p_k \notin F$,

$$
\frac{\partial L_{unsat}}{\partial\mathbf{x}[k]}\overset{iSTE}{\approx}\frac{\partial L_{unsat}}{\partial\mathbf{v}[k]}=\frac{c_2-c_1}{m}
$$

where c_1 (and c_2, resp.) is the number of clauses in C that are not satisfied by v and contain p_k (and $\neg p_k$, resp.).

$[L_{sat}]$ Recall that we proved that $\mathbb{1}_{\{0\}}(\mathbf{unsat})[i] \in \{0,1\}$ is the output of an indicator function whose value is 1 iff clause i evaluates to TRUE under v. Let $S \subseteq \{1,\dots,m\}$ denote the set of indices of clauses in

C that are evaluated as TRUE under v. Then

$$\begin{aligned}
&L_{sat}\\
=&avg(\mathbb{1}_{\{0\}}(\mathbf{unsat}) \odot \mathbf{keep})\\
=&\frac{1}{m}\sum_{i\in\{1,\dots,m\}}\Big(\mathbb{1}_{\{0\}}(\mathbf{unsat}[i]) \times \mathbf{keep}[i]\Big)\\
=&\frac{1}{m}\sum_{i\in S}\mathbf{keep}[i]\\
=&\frac{1}{m}\sum_{i\in S}\sum_{j\in\{1,\dots,n\}}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,j]) \times (1-\mathbf{L}_v[i,j])\\
&\qquad + \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,j]) \times \mathbf{L}_v[i,j]\Big)
\end{aligned}$$

Then the gradient of L_{sat} w.r.t. $\mathbf{v}[k]$ is

$$\begin{aligned}
&\frac{\partial L_{sat}}{\partial \mathbf{v}[k]}\\
=&\frac{1}{m}\sum_{i\in S}\sum_{j\in\{1,\dots,n\}}\Big(\\
&\qquad \mathbb{1}_{\{1\}}(\mathbf{L}_v[i,j]) \times \frac{\partial(1-\mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\\
&\qquad + \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,j]) \times \frac{\partial \mathbf{L}_v[i,j]}{\partial \mathbf{v}[k]}\Big)
\end{aligned}$$

$$\begin{aligned}
=&\frac{1}{m}\sum_{i\in S}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k]) \times \frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]}\\
&\qquad + \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k]) \times \frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]}\Big) \text{ (due to \textbf{(a)})}
\end{aligned}$$

$$\begin{aligned}
=&\frac{1}{m}\sum_{\substack{i\in S\\ \text{clause } i \text{ contains}\\ \text{literal } p_k}}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k]) \times \frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]}\\
&\qquad + \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k]) \times \frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]}\Big)+\\
&\frac{1}{m}\sum_{\substack{i\in S\\ \text{clause } i \text{ contains}\\ \text{literal } \neg p_k}}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k]) \times \frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]}\\
&\qquad + \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k]) \times \frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]}\Big) \text{ (due to \textbf{(b)})}
\end{aligned}$$

$$\begin{aligned}
=&\frac{1}{m}\sum_{\substack{i\in S\\ \text{clause } i \text{ contains}\\ \text{literal } p_k}}\Big(-\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k]) + \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])\Big)\\
&+\frac{1}{m}\sum_{\substack{i\in S\\ \text{clause } i \text{ contains}\\ \text{literal } \neg p_k}}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k]) - \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])\Big)\\
&\text{(due to \textbf{(c)} and \textbf{(d)})}
\end{aligned}$$

Recall that $\mathbf{L}_v[i,j] \in \{0,1\}$, and $\mathbf{L}_v[i,j] = 1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal evaluates to TRUE under v. It's easy to check that

- when clause i contains literal p_k, the value of $-\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k]) + \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])$ is -1 if $v \models p_k$ and is 1 if $v \not\models p_k$;
- when clause i contains literal $\neg p_k$, the value of $\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k]) - \mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])$ is -1 if $v \models p_k$ and is 1 if $v \not\models p_k$.

Thus

$$\frac{\partial L_{sat}}{\partial \mathbf{v}[k]} = \begin{cases} -\frac{c}{m} & \text{if } v \models p_k,\\ \frac{c}{m} & \text{if } v \not\models p_k.\end{cases}$$

where c is the number of clauses in S that contain a literal for atom p_k. Finally, if $p_k \notin F$,

$$\frac{\partial L_{sat}}{\partial \mathbf{x}[k]} \overset{iSTE}{\approx} \frac{\partial L_{sat}}{\partial \mathbf{v}[k]} = \begin{cases} -\frac{c}{m} & \text{if } v \models p_k,\\ \frac{c}{m} & \text{if } v \not\models p_k;\end{cases}$$

where c is the number of clauses in C that are satisfied by v and contain p_k or $\neg p_k$.

□

Proposition 4.5 Proposition 4.3 still holds for $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b(\mathbf{x})$.

[Complete Statement] Given a CNF theory C of m clauses and n atoms and a set F of atoms such that $C \cup F$ is satisfiable, let $\mathbf{C}, \mathbf{f}$ denote their matrix/vector representations, respectively. Given a neural network output $\mathbf{x} \in \mathbb{R}^n$ in logits (i.e., real numbers instead of probabilities), we construct $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b(\mathbf{x})$ and a truth assignment v such that $v(p_j) =$ TRUE if $\mathbf{v}[j]$ is 1, and $v(p_j) =$ FALSE if $\mathbf{v}[j]$ is 0. Let $C_{deduce} \subseteq C$ denote the set of Horn clauses H in C such that all but one literal in H are of the form $\neg p$ and $p \in F$. Then, for any $j \in \{1,\dots,n\}$,

1. if $p_j \in F$, all of $\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]}$, $\frac{\partial L_{unsat}}{\partial \mathbf{x}[j]}$, and $\frac{\partial L_{sat}}{\partial \mathbf{x}[j]}$ are zeros;
2. if $p_j \notin F$,

$$\frac{\partial L_{deduce}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \begin{cases} -c & \text{if } c>0 \text{ clauses in } C_{deduce}\\ & \text{contain literal } p_j;\\ c & \text{if } c>0 \text{ clauses in } C_{deduce}\\ & \text{contain literal } \neg p_j;\\ 0 & \text{otherwise;}\end{cases}$$

$$\frac{\partial L_{unsat}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \frac{c_2 - c_1}{m}$$

$$\frac{\partial L_{sat}}{\partial \mathbf{x}[j]} \overset{iSTE}{\approx} \begin{cases} -\frac{c_3}{m} & \text{if } v \models p_j,\\ \frac{c_3}{m} & \text{if } v \not\models p_j.\end{cases}$$

where $\overset{iSTE}{\approx}$ stands for the equivalence of gradients under iSTE; c_1 (and c_2, resp.) is the number of clauses in C that are not satisfied by v and contain p_j (and $\neg p_j$, resp.); c_3 is the number of clauses in C that are satisfied by v and contain p_j or $\neg p_j$.

Proof. Recall the definition of L_{cnf}

$$\begin{aligned}
\mathbf{L}_f &= \mathbf{C} \odot \mathbf{f} \\
\mathbf{L}_v &= \mathbb{1}_{\{1\}}(\mathbf{C}) \odot \mathbf{v} + \mathbb{1}_{\{-1\}}(\mathbf{C}) \odot (1 - \mathbf{v}) \\
\mathbf{deduce} &= \mathbb{1}_{\{1\}}\Big(sum(\mathbf{C} \odot \mathbf{C}) - sum(\mathbb{1}_{\{-1\}}(\mathbf{L}_f))\Big) \\
\mathbf{unsat} &= prod(1 - \mathbf{L}_v) \\
\mathbf{keep} &= sum(\mathbb{1}_{\{1\}}(\mathbf{L}_v) \odot (1 - \mathbf{L}_v) + \mathbb{1}_{\{0\}}(\mathbf{L}_v) \odot \mathbf{L}_v) \\
L_{deduce} &= sum(\mathbf{deduce} \odot \mathbf{unsat}) \\
L_{unsat} &= avg(\mathbb{1}_{\{1\}}(\mathbf{unsat}) \odot \mathbf{unsat}) \\
L_{sat} &= avg(\mathbb{1}_{\{0\}}(\mathbf{unsat}) \odot \mathbf{keep})
\end{aligned}$$

$$L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) = L_{deduce} + L_{unsat} + L_{sat}$$

We will prove each bullet in Proposition 4.5 as follows. This proof is almost the same as the proof for Proposition 4.3 since the choice of $b(x)$ v.s. $b_p(x)$ doesn't affect the gradient computation from L_{cnf} to $\mathbf{x}$ under iSTE.

1. Take any $k \in \{1, \ldots, n\}$, let's focus on $\mathbf{x}[k]$ and compute the gradient of $L \in \{L_{deduce}, L_{unsat}, L_{sat}\}$ to it with iSTE. According to the chain rule and since $\frac{\partial \mathbf{v}[i]}{\partial b(\mathbf{x})[j]} = 0$ for $i \neq j$, we have

$$\frac{\partial L}{\partial \mathbf{x}[k]} = \frac{\partial L}{\partial \mathbf{v}[k]} \times \frac{\partial \mathbf{v}[k]}{\partial b(\mathbf{x}[k])} \times \frac{\partial b(\mathbf{x}[k])}{\partial \mathbf{x}[k]}.$$

Under iSTE, the last term $\frac{\partial b(\mathbf{x}[k])}{\partial \mathbf{x}[k]}$ is replaced with $\frac{\partial s(\mathbf{x}[k])}{\partial \mathbf{x}[k]} = \frac{\partial \mathbf{x}[k]}{\partial \mathbf{x}[k]} = 1$. Thus

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{x}[k]} &= \frac{\partial L}{\partial \mathbf{v}[k]} \times \frac{\partial \mathbf{v}[k]}{\partial b(\mathbf{x}[k])} \times \frac{\partial b(\mathbf{x}[k])}{\partial \mathbf{x}[k]} \\
&\overset{iSTE}{\approx} \frac{\partial L}{\partial \mathbf{v}[k]} \times \frac{\partial \mathbf{v}[k]}{\partial b(\mathbf{x}[k])} \quad \text{(under iSTE)} \\
&= \frac{\partial L}{\partial \mathbf{v}[k]} \times \frac{\partial(\mathbf{f}[k] + \mathbb{1}_{\{0\}}(\mathbf{f}[k]) \times b(\mathbf{x}[k]))}{\partial b(\mathbf{x}[k])} \\
&= \begin{cases} \frac{\partial L}{\partial \mathbf{v}[k]} & \text{if } \mathbf{f}[k] = 0, \\ 0 & \text{if } \mathbf{f}[k] = 1. \end{cases}
\end{aligned}$$

Since $\mathbf{f}[k] = 1$ iff $p_k \in F$, if $p_k \in F$, then all of $\frac{\partial L_{deduce}}{\partial \mathbf{x}[k]}$, $\frac{\partial L_{unsat}}{\partial \mathbf{x}[k]}$, and $\frac{\partial L_{sat}}{\partial \mathbf{x}[k]}$ are zeros.

2. We know $p_k \notin F$ iff $\mathbf{f}[k] = 0$. As proved in the first bullet, for $L \in \{L_{deduce}, L_{unsat}, L_{sat}\}$, if $p_k \notin F$, then $\frac{\partial L}{\partial \mathbf{x}[k]} = \frac{\partial L}{\partial \mathbf{v}[k]}$. We further analyze the value of $\frac{\partial L}{\partial \mathbf{v}[k]}$ for each L under the condition that $\mathbf{f}[k] = 0$.

[L_{deduce}] According to the definition,

$$\begin{aligned}
&L_{deduce} \\
&= \sum_{i \in \{1,\ldots,m\}} \Big(\mathbf{deduce}[i] \times \mathbf{unsat}[i]\Big) \\
&= \sum_{i \in \{1,\ldots,m\}} \Big(\mathbf{deduce}[i] \times \prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])\Big)
\end{aligned}$$

$$\begin{aligned}
&\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} \\
&= \sum_{i \in \{1,\ldots,m\}} \frac{\partial\Big(\mathbf{deduce}[i] \times \prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])\Big)}{\partial \mathbf{v}[k]} \\
&= \sum_{i \in \{1,\ldots,m\}} \Big(\frac{\partial \mathbf{deduce}[i]}{\partial \mathbf{v}[k]} \times \prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j]) + \\
&\qquad \mathbf{deduce}[i] \times \frac{\partial \prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\Big)
\end{aligned}$$

Since $\mathbf{deduce}$ is the result of an indicator function, $\frac{\partial \mathbf{deduce}[i]}{\partial \mathbf{v}[k]} = 0$. Then,

$$\begin{aligned}
&\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} \\
&= \sum_{i \in \{1,\ldots,m\}} \Big(\mathbf{deduce}[i] \times \frac{\partial \prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\Big).
\end{aligned}$$

Let $U \subseteq \{1, \ldots, m\}$ denote the set of indices of all clauses in C_{deduce}. Since $\mathbf{deduce}[i] = 1$ iff $i \in U$,

$$\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} = \sum_{i \in U} \Big(\frac{\partial \prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\Big).$$

Let $G_{i,k}$ denote $\frac{\partial \prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}$. Then

$$\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} = \sum_{i \in U} G_{i,k}.$$

Let's analyze the value of $G_{i,k}$ where $i \in U$ and $k \in \{1, \ldots, n\}$ such that $\mathbf{f}[k] = 0$. According to the product rule below,

$$\frac{d}{dx}\left[\prod_{i=1}^{k} f_i(x)\right] = \left(\prod_{i=1}^{k} f_i(x)\right)\left(\sum_{i=1}^{k} \frac{f_i'(x)}{f_i(x)}\right)$$

we have

$$\begin{aligned}
&G_{i,k} \\
&= \frac{\partial \prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]} \\
&= \left(\prod_{j \in \{1,\ldots,n\}} (1 - \mathbf{L}_v[i,j])\right) \times \sum_{j \in \{1,\ldots,n\}} \frac{\frac{\partial(1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}}{1 - \mathbf{L}_v[i,j]}
\end{aligned}$$

Since $\mathbf{L}_v[i,j] = \mathbb{1}_{\{1\}}(\mathbf{C})[i,j] \times \mathbf{v}[j] + \mathbb{1}_{\{-1\}}(\mathbf{C})[i,j] \times (1 - \mathbf{v}[j])$, we know

- **(a)** for $j \in \{1,\dots,n\}$ such that $j \neq k$, $\frac{\partial(1-\mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]} = 0$ and $\frac{\partial \mathbf{L}_v[i,j]}{\partial \mathbf{v}[k]} = 0$;
- **(b)** when clause i doesn't contain a literal for atom p_k, $\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]} = 0$ and $\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]} = 0$;
- **(c)** when clause i contains literal p_k, $\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]} = -1$ and $\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]} = 1$;
- **(d)** when clause i contains literal $\neg p_k$, $\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]} = 1$ and $\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]} = -1$.

We will refer to the above 4 bullets with their identifiers.

Since $i \in U$, we know clause i has all but one literal of the form $\neg p_j$ such that $p_j \in F$. Since $\mathbf{f}[k] = 0$, we know $p_k \notin F$. Then, when clause i contains literal p_k or $\neg p_k$, all other literals in clause i must be of the form $\neg p_j$ where $p_j \in F$. For every literal $\neg p_j$ in clause i where $j \neq k$, we know $p_j \in F$, thus $\mathbf{f}[j] = 1$; since $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b(\mathbf{x})$, then $\mathbf{v}[j] = 1$; consequently, the literal $\neg p_j$ evaluates to FALSE under v. Recall that $\mathbf{L}_v[i,j] \in \{0,1\}$, and $\mathbf{L}_v[i,j] = 1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal evaluates to TRUE under v, then we know

- when $i \in U$, $\mathbf{f}[k] = 0$, and clause i contains literal p_k or $\neg p_k$, $\mathbf{L}_v[i,j] = 0$ for $j \in \{1,\dots,n\}$ such that $j \neq k$.

Then we have

$$
\begin{aligned}
&G_{i,k} \\
=&\left(\prod_{j\in\{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])\right) \times \sum_{j\in\{1,\dots,n\}} \frac{\frac{\partial(1-\mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}}{1 - \mathbf{L}_v[i,j]} \\
=&\left(\prod_{j\in\{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])\right) \times \frac{\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]}}{1 - \mathbf{L}_v[i,k]} \text{ (due to \textbf{(a)})} \\
=&\frac{\partial(1 - \mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]} \times \prod_{\substack{j\in\{1,\dots,n\} \\ j\neq k}} (1 - \mathbf{L}_v[i,j]) \\
=&\begin{cases}
0 & \text{if clause } i \text{ doesn't contain a literal} \\
 & \text{for atom } p_k \text{ (due to \textbf{(b)})} \\
-1 & \text{if clause } i \text{ contains a literal } p_k \text{ (due to \textbf{(c)})} \\
1 & \text{if clause } i \text{ contains a literal } \neg p_k \text{ (due to \textbf{(d)})}
\end{cases}
\end{aligned}
$$

Since $i \in U$ and $\mathbf{f}[k] = 0$, when clause i contains a literal l_k for atom p_k, we know $F \not\models l_j$ for every literal l_j in clause i such that $j \neq k$. Since $C \cup F$ is satisfiable, we know $C \cup F \models l_k$ and there cannot be two clauses in C_{deduce} containing different literals p_k and $\neg p_k$. Thus, when $\mathbf{f}[k] = 0$,

$$
\begin{aligned}
&\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} \\
=&\sum_{i\in U} G_{i,k} \\
=&\begin{cases}
-c & \text{if } c > 0 \text{ clauses in } C_{deduce} \text{ contain literal } p_k, \\
c & \text{if } c > 0 \text{ clauses in } C_{deduce} \text{ contain literal } \neg p_k, \\
0 & \text{otherwise.}
\end{cases}
\end{aligned}
$$

Note that the first 2 cases above are disjoint since there cannot be two clauses in C_{deduce} containing different literals p_k and $\neg p_k$.
Finally, if $p_k \notin F$,

$$
\begin{aligned}
&\frac{\partial L_{deduce}}{\partial \mathbf{x}[k]} \\
\overset{iSTE}{\approx}&\frac{\partial L_{deduce}}{\partial \mathbf{v}[k]} \\
=&\begin{cases}
-c & \text{if } c > 0 \text{ clauses in } C_{deduce} \\
 & \text{contain literal } p_k; \\
c & \text{if } c > 0 \text{ clauses in } C_{deduce} \\
 & \text{contain literal } \neg p_k; \\
0 & \text{otherwise;}
\end{cases}
\end{aligned}
$$

$[L_{unsat}]$ According to the definition,

$$
\begin{aligned}
L_{unsat} =& avg(\mathbb{1}_{\{1\}}(\mathbf{unsat}) \odot \mathbf{unsat}) \\
=&\frac{1}{m} \sum_{i\in\{1,\dots,m\}} \Big(\mathbb{1}_{\{1\}}(\mathbf{unsat}[i]) \times \\
&\prod_{j\in\{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])\Big)
\end{aligned}
$$

Recall that we proved that $\mathbb{1}_{\{1\}}(\mathbf{unsat})[i] \in \{0,1\}$ is the output of an indicator function whose value is 1 iff clause i evaluates to FALSE under v. Let $U \subseteq \{1,\dots,m\}$ denote the set of indices of clauses in C that are evaluated as FALSE under v.

$$
L_{unsat} = \frac{1}{m} \sum_{i\in U} \Big(\prod_{j\in\{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])\Big)
$$

Then the gradient of L_{unsat} w.r.t. $\mathbf{v}[k]$ is

$$
\frac{\partial L_{unsat}}{\partial \mathbf{v}[k]} = \frac{1}{m} \sum_{i\in U} \Big(\frac{\partial \prod_{j\in\{1,\dots,n\}} (1 - \mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\Big).
$$

Recall that $\mathbf{L}_v[i,j] \in \{0,1\}$, and $\mathbf{L}_v[i,j] = 1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal evaluates to TRUE under v. When $i \in U$, clause i evaluates to FALSE under v. Thus when $i \in U$, all literals in clause i must be evaluated

as FALSE under v, and consequently, $\mathbf{L}_v[i,j]=0$ for all $j\in\{1,\dots,m\}$. Then

$$\begin{aligned}\frac{\partial L_{unsat}}{\partial \mathbf{v}[k]} =&\frac{1}{m}\sum_{i\in U}\Big(\frac{\partial \prod_{j\in\{1,\dots,n\}}(1-\mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\Big)\\ =&\frac{1}{m}\sum_{i\in U}\Big(\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]}\Big) \text{ (due to \textbf{(a)})}\\ =&\frac{c_2-c_1}{m}\end{aligned}$$

where c_1 (and c_2, resp.) is the number of clauses in U that contain p_k (and $\neg p_k$, resp.). Finally, if $p_k\notin F$,

$$\frac{\partial L_{unsat}}{\partial \mathbf{x}[k]}\overset{iSTE}{\approx}\frac{\partial L_{unsat}}{\partial \mathbf{v}[k]}=\frac{c_2-c_1}{m}$$

where c_1 (and c_2, resp.) is the number of clauses in C that are not satisfied by v and contain p_k (and $\neg p_k$, resp.).

$[L_{sat}]$ Recall that we proved that $\mathbb{1}_{\{0\}}(\mathbf{unsat})[i]\in\{0,1\}$ is the output of an indicator function whose value is 1 iff clause i evaluates to TRUE under v. Let $S\subseteq\{1,\dots,m\}$ denote the set of indices of clauses in C that are evaluated as TRUE under v. Then

$$\begin{aligned}L_{sat} =&avg(\mathbb{1}_{\{0\}}(\mathbf{unsat})\odot\mathbf{keep})\\ =&\frac{1}{m}\sum_{i\in\{1,\dots,m\}}\Big(\mathbb{1}_{\{0\}}(\mathbf{unsat}[i])\times\mathbf{keep}[i]\Big)\\ =&\frac{1}{m}\sum_{i\in S}\mathbf{keep}[i]\\ =&\frac{1}{m}\sum_{i\in S}\sum_{j\in\{1,\dots,n\}}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,j])\times(1-\mathbf{L}_v[i,j])\\ &+\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,j])\times\mathbf{L}_v[i,j]\Big)\end{aligned}$$

Then the gradient of L_{sat} w.r.t. $\mathbf{v}[k]$ is

$$\begin{aligned}&\frac{\partial L_{sat}}{\partial \mathbf{v}[k]}\\ =&\frac{1}{m}\sum_{i\in S}\sum_{j\in\{1,\dots,n\}}\Big(\\ &\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,j])\times\frac{\partial(1-\mathbf{L}_v[i,j])}{\partial \mathbf{v}[k]}\\ &+\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,j])\times\frac{\partial \mathbf{L}_v[i,j]}{\partial \mathbf{v}[k]}\Big)\end{aligned}$$

$$\begin{aligned}=&\frac{1}{m}\sum_{i\in S}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k])\times\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]}\\ &+\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])\times\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]}\Big) \text{ (due to \textbf{(a)})}\end{aligned}$$

$$\begin{aligned}=&\frac{1}{m}\sum_{\substack{i\in S\\ \text{clause } i \text{ contains}\\ \text{literal } p_k}}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k])\times\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]}\\ &+\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])\times\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]}\Big)+\\ &\frac{1}{m}\sum_{\substack{i\in S\\ \text{clause } i \text{ contains}\\ \text{literal } \neg p_k}}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k])\times\frac{\partial(1-\mathbf{L}_v[i,k])}{\partial \mathbf{v}[k]}\\ &+\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])\times\frac{\partial \mathbf{L}_v[i,k]}{\partial \mathbf{v}[k]}\Big) \text{ (due to \textbf{(b)})}\end{aligned}$$

$$\begin{aligned}=&\frac{1}{m}\sum_{\substack{i\in S\\ \text{clause } i \text{ contains}\\ \text{literal } p_k}}\Big(-\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k])+\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])\Big)\\ &+\frac{1}{m}\sum_{\substack{i\in S\\ \text{clause } i \text{ contains}\\ \text{literal } \neg p_k}}\Big(\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k])-\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])\Big)\end{aligned}$$

(due to **(c)** and **(d)**)

Recall that $\mathbf{L}_v[i,j]\in\{0,1\}$, and $\mathbf{L}_v[i,j]=1$ iff clause i contains a literal (p_j or $\neg p_j$) for atom p_j and this literal evaluates to TRUE under v. It's easy to check that

- when clause i contains literal p_k, the value of $-\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k])+\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])$ is -1 if $v\models p_k$ and is 1 if $v\not\models p_k$;
- when clause i contains literal $\neg p_k$, the value of $\mathbb{1}_{\{1\}}(\mathbf{L}_v[i,k])-\mathbb{1}_{\{0\}}(\mathbf{L}_v[i,k])$ is -1 if $v\models p_k$ and is 1 if $v\not\models p_k$.

Thus

$$\frac{\partial L_{sat}}{\partial \mathbf{v}[k]}=\begin{cases}-\frac{c}{m} & \text{if } v\models p_k,\\ \frac{c}{m} & \text{if } v\not\models p_k.\end{cases}$$

where c is the number of clauses in S that contain a literal for atom p_k. Finally, if $p_k\notin F$,

$$\frac{\partial L_{sat}}{\partial \mathbf{x}[k]}\overset{iSTE}{\approx}\frac{\partial L_{sat}}{\partial \mathbf{v}[k]}=\begin{cases}-\frac{c}{m} & \text{if } v\models p_k,\\ \frac{c}{m} & \text{if } v\not\models p_k;\end{cases}$$

where c is the number of clauses in C that are satisfied by v and contain p_k or $\neg p_k$.

□

C. More Details about Experiments

C.1. mnistAdd2

In **mnistAdd2** problem (Manhaeve et al., 2018), a data instance is a 5-tuple $\langle i_1,i_2,i_3,i_4,l\rangle$ such that i_* are images of digits and l is an integer in $\{0,\dots,198\}$ denoting the

sum of two 2-digit numbers $i_1 i_2$ and $i_3 i_4$. The task is, given 15k data instances of $\langle i_1, i_2, i_3, i_4, l\rangle$, to train a CNN for digit classification given such weak supervision. The CNF for **mnistAdd2** consists of the 199 clauses of the form

$$\neg sum(l) \vee \bigvee_{\substack{n_1,n_2,n_3,n_4\in\{0,\ldots,9\}:\\ 10(n_1+n_3)+n_2+n_4=l}} pred(n_1, n_2, n_3, n_4)$$

for $l \in \{0, \ldots, 198\}$. Intuitively, this clause says that "if the sum of $i_1 i_2$ and $i_3 i_4$ is l, then their individual labels n_1, n_2, n_3, n_4 must satisfy $10(n_1 + n_3) + n_2 + n_4 = l$."

This CNF contains 199 clauses and $10^4 + 199 = 10199$ atoms for $pred/4$ and $sum/1$, respectively. According to the definition, we can construct the matrix $\mathbf{C} \in \{-1, 0, 1\}^{199\times 10199}$ where each row represents a clause.

To construct $\mathbf{f}$ and $\mathbf{v}$ for a data instance $\langle i_1, i_2, i_3, i_4, l\rangle$, the facts $\mathbf{f}$ is simply a vector in $\{0, 1\}^{10199}$ with 10198 0s and a single 1 for atom $sum(l)$; while the prediction $\mathbf{v}$ is a vector in $\{0, 1\}^{10199}$ obtained as follows. We (i) feed images i_1,i_2,i_3,i_4 into the CNN and obtain the outputs $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \in \mathbb{R}^{10}$ (consisting of probabilities); (ii) construct $\mathbf{x} \in \mathbb{R}^{10000}$ such that its $(1000a+100b+10c+d)$-th element is $\mathbf{x}_1[a] \times \mathbf{x}_2[b] \times \mathbf{x}_3[c] \times \mathbf{x}_4[d]$ for $a, b, c, d \in \{0, \ldots, 9\}$; and (iii) $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$.

The loss function used for **mnistAdd2** problem is

$$\mathcal{L} = \alpha L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + \sum_{\mathbf{x}\in\{\mathbf{x}_1,\ldots,\mathbf{x}_4\}} \beta L_{bound}(\mathbf{x})$$

where $\alpha = 1$ and $\beta = 0.01$.

C.2. mnistAdd using b(x) and iSTE

In **mnistAdd** problem, a data instance is a 3-tuple $\langle i_1, i_2, l\rangle$ where i_1, i_2 are 2 images of digits and l is an integer in $\{0, \ldots, 18\}$ indicating the sum of the 2 digit images. The propositional signature σ in this problem consists of 139 atoms: 19 atoms of the form $sum(i_1, i_2, s)$ for $s \in \{0, \ldots, 18\}$, 20 atoms of the form $digit(i, n)$ for $i \in \{i_1, i_2\}$ and for $n \in \{0, \ldots, 9\}$, and 100 atoms of the form $conj(i_1, n_1, i_2, n_2)$ for $n_1, n_2 \in \{0, \ldots, 9\}$ (denoting the conjunction of $digit(i_1, n_1)$ and $digit(i_2, n_2)$). The CNF for this problem consists of 111 clauses: 19 clauses of the form

$$\neg sum(i_1, i_2, s) \vee \bigvee_{\substack{n_1,n_2\in\{0,\ldots,9\}\\ n_1+n_2=s}} conj(i_1, n_1, i_2, n_2) \quad (12)$$

for $s \in \{0, \ldots, 18\}$, 2 clauses of the form

$$digit(i, 0) \vee \cdots \vee digit(i, 9) \quad (13)$$

for $i \in \{i_1, i_2\}$, and 90 clauses of the form

$$\neg digit(i, n_1) \vee \neg digit(i, n_2) \quad (14)$$

for $i \in \{i_1, i_2\}$ and for $n_1, n_2 \in \{0, \ldots, 9\}$ such that $n_1 < n_2$. Intuitively, clause (12) says that "if the sum of i_1 and i_2 is s, then we should be able to predict the labels n_1, n_2 of i_1, i_2 such that they sum up to s." Clauses (13) and (14) define the existence and uniqueness constraints on the label of i. Note that clauses (13) and (14) are not needed if we use $b_p(x)$+iSTE since these constraints will be enforced by the softmax function in the last layer of the neural network, which is widely and inherently used in most neuro-symbolic formalisms.

This CNF can be represented by the matrix $\mathbf{C} \in \{-1, 0, 1\}^{111\times 139}$. To construct $\mathbf{f}$ and $\mathbf{v}$ for a data instance $\langle i_1, i_2, l\rangle$, the facts $\mathbf{f}$ is simply a vector in $\{0, 1\}^{139}$ with 138 0s and a single 1 for atom $sum(i_1, i_2, l)$; while the prediction $\mathbf{v}$ is a vector in $\{0, 1\}^{139}$ obtained as follows. We (i) feed images i_1,i_2 into the CNN and obtain the outputs $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{10}$ (consisting of probabilities); (ii) construct $\mathbf{x} \in \mathbb{R}^{139}$ such that its $(10a + b)$-th element is $\mathbf{x}_1[a] \times \mathbf{x}_2[b]$ for $a, b \in \{0, \ldots, 9\}$ and its remaining elements are 0; and (iii) $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$.

Then, the total loss is defined as

$$\mathcal{L} = \alpha L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + \sum_{\mathbf{x}\in\{\mathbf{x}_1,\mathbf{x}_2\}} \beta L_{bound}(\mathbf{x})$$

where $\alpha = 1$ and $\beta = 0.1$.

C.3. add2x2

In **add2x2** problem, a data instance is a 8-tuple $\langle i_1, i_2, i_3, i_4, row_1, row_2, col_1, col_2\rangle$ where i_* are 4 images of digits arranged in the following order in a grid

$$\begin{matrix} i_1 & i_2 \\ i_3 & i_4 \end{matrix},$$

and each row_* or col_* is an integer in $\{0, \ldots, 18\}$ denoting the sum of 2 digits on the specified row/column in the above grid. The task is to train a CNN for digit classification given such weak supervision.

For $o, o' \in \{(i_1, i_2), (i_3, i_4), (i_1, i_3), (i_2, i_4)\}$, and for $r \in \{0, \ldots, 18\}$ the CNF contains the following clause:

$$\neg sum(o, o', r) \vee \bigvee_{\substack{i,j\in\{0,\ldots,9\}\\ i+j=r}} conj(o, i, o', j).$$

This clause can be read as "if the sum of o and o' is r, then o and o' must be some values i and j such that $i+j = r$." This CNF contains $4 \times 19 = 76$ clauses and $76 + 4 \times 10 \times 10 = 476$ atoms (for $sum/3$ and $conj/4$, resp.). This CNF can be represented by the matrix $\mathbf{C} \in \{-1, 0, 1\}^{76\times 476}$.

To construct $\mathbf{f}$ and $\mathbf{v}$ for a data instance $\langle i_1, i_2, i_3, i_4, row_1, row_2, col_1, col_2\rangle$, the facts $\mathbf{f}$ is simply a vector in $\{0, 1\}^{476}$ with 472 0s and four 1s for atoms

$sum(i_1, i_2, row_1)$, $sum(i_3, i_4, row_2)$, $sum(i_1, i_3, col_1)$, and $sum(i_2, i_4, col_2)$; while the prediction $\mathbf{v}$ is a vector in $\{0,1\}^{476}$ obtained as follows. We (i) feed images i_1,i_2,i_3,i_4 into the CNN and obtain the outputs $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \in \mathbb{R}^{10}$ (consisting of probabilities); (ii) construct $\mathbf{x} \in \mathbb{R}^{476}$ as the concatenation of $\langle \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \{0\}^{76} \rangle$ where

$$\mathbf{v}_1 = \mathbf{x}_1^T \cdot \mathbf{x}_2, \qquad \mathbf{v}_2 = \mathbf{x}_3^T \cdot \mathbf{x}_4,$$
$$\mathbf{v}_3 = \mathbf{x}_1^T \cdot \mathbf{x}_3, \qquad \mathbf{v}_4 = \mathbf{x}_2^T \cdot \mathbf{x}_4;$$

and (iii) $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$.

Then, the total loss is defined as

$$\mathcal{L} = \alpha L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + \sum_{\mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_4\}} \beta L_{bound}(\mathbf{x})$$

where $\alpha = 1$ and $\beta = 0.1$.

C.4. apply2x2

In **apply2x2** problem, a data instance is a 11-tuple $\langle d_1, d_2, d_3, o_{11}, o_{12}, o_{21}, o_{22}, row_1, row_2, col_1, col_2 \rangle$ where d_* are digits in $\{0, \ldots, 9\}$, o_* are 4 images of operators in $\{+, -, \times\}$ arranged in the following order in a grid

$$\begin{array}{cc} o_{11} & o_{12} \\ o_{21} & o_{22} \end{array},$$

and each row_* or col_* is an integer denoting the value of the formula (e.g., $(4 \times 7) - 9$)

$$(d_1\ o_1\ d_2)\ o_2\ d_3 \tag{15}$$

where $(o_1, o_2) \in \{(o_{11}, o_{12}), (o_{21}, o_{22}), (o_{11}, o_{21}), (o_{12}, o_{22})\}$ denotes the two operators on the specified row/column in the above grid. The task is to train a CNN for digit classification given such weak supervision.

We construct a CNF to relate formula (15) and its value and will apply the CNF loss for $(o_1, o_2) \in \{ (o_{11}, o_{12}), (o_{21}, o_{22}), (o_{11}, o_{21}), (o_{12}, o_{22})\}$.

For $d_1, d_2, d_3 \in \{0, \ldots, 10\}$, and for all possible r such that $(d_1\ Op_1\ d_2)\ Op_2\ d_3 = r$ for some $Op_1, Op_2 \in \{+, -, \times\}$, the CNF contains the following clause:

$$\neg apply(d_1, o_1, d_2, o_2, d_3, r) \vee \bigvee_{\substack{Op_1, Op_2 \in \{+,-,\times\} \\ (d_1\ Op_1\ d_2)\ Op_2\ d_3 = r}} (operators(o_1, Op_1, o_2, Op_2)).$$

This clause can be read as "if the result is r after applying o_1 and o_2 to the three digits, then o_1 and o_2 must be some values Op_1 and Op_2 such that $(d_1\ Op_1\ d_2)\ Op_2\ d_3 = r$." This CNF contains 10597 clauses and 10606 atoms and can be represented by the matrix $\mathbf{C} \in \{-1, 0, 1\}^{10597 \times 10606}$.

Given a data instance $\langle d_1, d_2, d_3, o_{11}, o_{12}, o_{21}, o_{22}, row_1, row_2, col_1, col_2 \rangle$, we construct $\mathbf{v}_i, \mathbf{f}_i \in \{0,1\}^{10606}$ for $i \in \{1, \ldots, 4\}$, one for each $\langle o_1, o_2, r \rangle \in \{\langle o_{11}, o_{12}, row_1 \rangle, \langle o_{21}, o_{22}, row_2 \rangle, \langle o_{11}, o_{21}, col_1 \rangle, \langle o_{12}, o_{22}, col_2 \rangle\}$. The detailed steps to construct $\mathbf{f}$ and $\mathbf{v}$ for $\langle o_1, o_2, r \rangle$ is as follows.

First, the facts $\mathbf{f}$ is simply a vector in $\{0,1\}^{10606}$ with 10605 0s and a single 1 for atom $apply(d_1, o_1, d_2, o_2, d_3, r)$. Second, the prediction $\mathbf{v}$ is a vector in $\{0,1\}^{10606}$ obtained as follows. We (i) feed images o_1,o_2 into the CNN and obtain the outputs $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^3$ (consisting of probabilities); (ii) construct $\mathbf{x} \in \mathbb{R}^{10606}$ such that its $(3a + b)$-th element is $\mathbf{x}_1[a] \times \mathbf{x}_2[b]$ for $a, b \in \{0, \ldots, 2\}$ and its remaining elements are 0; and (iii) $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$.

Then, the total loss is defined as

$$\mathcal{L} = \sum_{i \in \{1,\ldots,4\}} \alpha L_{cnf}(\mathbf{C}, \mathbf{v}_i, \mathbf{f}_i) + \sum_{\mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_4\}} \beta L_{bound}(\mathbf{x})$$

where $\alpha = 1$ and $\beta = 0.1$.

C.5. member(n)

We take **member(3)** problem as an example. In **member(3)** problem, a data instance is a 5-tuple $\langle i_1, i_2, i_3, d, l \rangle$ where i_1, i_2, i_3 are 3 images of digits, d is a digit in $\{0, \ldots, 9\}$, and l is an integer in $\{0, 1\}$ indicating whether d appears in the set of digit images. The task is to train a CNN for digit classification given such weak supervision. The CNF for this problem consists of the 2 kinds of clauses in table 5.

Table 5. Clauses in the CNF for **member(3)** Problem

Clause	Reading
$\neg in(d, 1) \vee digit(i_1, d) \vee digit(i_2, d) \vee digit(i_3, d)$ (for $d \in \{0, \ldots, 9\}$)	if d appears in the 3 images, then i_1 or i_2 or i_3 must be digit d
$\neg in(d, 0) \vee \neg digit(i, d)$ (for $d \in \{0, \ldots, 9\}$ and $i \in \{i_1, i_2, i_3\}$)	if d doesn't appear in the 3 images, then each image i cannot be digit d

This CNF contains $10 + 10 \times 3 = 40$ clauses and $3 \times 10 + 2 \times 10 = 50$ atoms for $digit/2$ and $in/2$ respectively. According to the definition, we can construct the matrix $\mathbf{C} \in \{-1, 0, 1\}^{40 \times 50}$ where each row represents a clause. For instance, the row for the clause $\neg in(5, 1) \vee digit(i_1, 5) \vee digit(i_2, 5) \vee digit(i_3, 5)$ is a row vector in $\{-1, 0, 1\}^{1 \times 50}$ containing 46 0s, a single -1 for atom $in(5, 1)$, and three 1s for atoms $digit(i_1, 5), digit(i_2, 5), digit(i_3, 5)$.

To construct $\mathbf{f}$ and $\mathbf{v}$ for a data instance $\langle i_1, i_2, i_3, d, l \rangle$, the facts $\mathbf{f}$ is simply a vector in $\{0,1\}^{50}$ with 49 0s and a single 1 for atom $in(d, l)$; while the prediction $\mathbf{v}$ is a vector in $\{0,1\}^{50}$ obtained as follows. We (i) feed images i_1,i_2,i_3 into the CNN and obtain the NN outputs

$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathbb{R}^{10}$ consisting of probabilities, (ii) construct $\mathbf{x} \in \mathbb{R}^{50}$ by concatenating $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and the vector $\{0\}^{20}$, and (iii) $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$.

The total loss function is

$$\mathcal{L} = \alpha L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + \sum_{\mathbf{x} \in \{\mathbf{x}_1, \ldots, \mathbf{x}_3\}} \beta L_{bound}(\mathbf{x})$$

where $\alpha = 1$ and $\beta = 0.1$.

C.6. Sudoku

We use a typical CNF for 9×9 Sudoku problem. The CNF is defined on a propositional signature $\sigma = \{a(R, C, N) \mid R, C, N \in \{1, \ldots, 9\}\}$ where $a(R, C, N)$ represents "digit N is assigned at row R column C". The CNF consists of the following $1 + \binom{9}{2} = 37$ clauses for each of the $4 \times 9 \times 9 = 324$ different sets A of atoms

$$\bigvee_{p \in A} p$$
$$\neg p_i \vee \neg p_j \quad (\text{for } p_i, p_j \in A \text{ and } i < j)$$

where the $4 \times 9 \times 9$ definitions of A can be split into the following 4 categories, each consisting of 9×9 definitions.

1. (UEC on row indices)
   For $C, N \in \{1, \ldots, 9\}$, A is the set of atoms $\{a(1, C, N), \ldots, a(9, C, N)\}$.
2. (UEC on column indices)
   For $R, N \in \{1, \ldots, 9\}$, A is the set of atoms $\{a(R, 1, N), \ldots, a(R, 9, N)\}$.
3. (UEC on 9 values in each cell)
   For $R, C \in \{1, \ldots, 9\}$, A is the set of atoms $\{a(R, C, 1), \ldots, a(R, C, 9)\}$.
4. (Optional: UEC on 9 cells in the same 3×3 box)
   For $B, N \in \{1, \ldots, 9\}$, A is the set of atoms $\{a(R_1, C_1, N), \ldots, a(R_9, C_9, N)\}$ such that the 9 cells (R_i, C_i) for $i \in \{1 \ldots, 9\}$ are the 9 cells in the B-th box in the 9×9 grid for value N. Note that the clauses in bullet 4 are optional under the setting $b_p(x)$+iSTE since they are already enforced by the softmax function used in the last layer to turn NN output to probabilities.

   This CNF can be represented by a matrix $\mathbf{C} \in \{-1, 0, 1\}^{8991 \times 729}$. The dataset in the CNN experiments consists of 70k data instances, 20% supervised for testing, and 80% unsupervised for training. Each unsupervised data instance is a single vector $\mathbf{q} \in \{0, 1, \ldots, 9\}^{81}$ representing a 9×9 Sudoku board (0 denotes an empty cell). The non-zero values in $\mathbf{q}$ are treated as atomic facts F and we construct the matrix $\mathbf{F} \in \{0, 1\}^{81 \times 9}$ such that, for $i \in \{1, \ldots, 81\}$, the i-th row $\mathbf{F}[i, :]$ is the vector $\{0\}^9$ if $\mathbf{q}[i] = 0$ and is the one-hot vector for $\mathbf{q}[i]$ if $\mathbf{q}[i] \neq 0$. Then, the vector $\mathbf{f} \in \{0, 1\}^{729}$ is simply the flattened version of $\mathbf{F}$. We feed $\mathbf{q}$ into the CNN and obtain the output $\mathbf{x} \in \mathbb{R}^{729}$ consisting of probabilities. The prediction $\mathbf{v} \in \{0, 1\}^{729}$ is obtained as $\mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$.

   Then, the total loss function $\mathcal{L}$ used to train the CNN for Sudoku is

$$\mathcal{L} = \alpha L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + \beta L_{bound}(\mathbf{x})$$

   where $\alpha = 1$ and $\beta = 0.1$.

D. Ablation Study with Sudoku-GNN

To better analyze the effect of constraint losses on general GNN, in this section, we apply constraint losses to a publicly available GNN for Sudoku problem.[6] The graph for Sudoku problem consists of 81 nodes, one for each cell in the Sudoku board, and 1620 edges, one for each pair of nodes in the same row, column, or 3×3 non-overlapping box. The GNN consists of an embedding layer, 8 iterations of message passing layers, and an output layer.

For each data instance $\langle \mathbf{q}, \mathbf{l} \rangle$, the GNN takes $\mathbf{q} \in \{0, 1, \ldots, 9\}^{81}$ as input and outputs a matrix of probabilities $\mathbf{X} \in \mathbf{R}^{81 \times 9}$ after 8 message passing steps.

The baseline loss $L_{baseline}$ is the cross-entropy loss defined on prediction $\mathbf{X}$ and label $\mathbf{l}$.

$$L_{baseline} = L_{cross\_entropy}(\mathbf{X}, \mathbf{l})$$

The constraint loss L_{cl} is the same as the total loss in Appendix C.6 where $\mathbf{x}$ is the flattening of $\mathbf{X}$.

$$L_{cl} = L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + 0.1 \times L_{bound}(\mathbf{x}). \tag{16}$$

In addition, we designed the following domain-specific loss functions for Sudoku problem as semantic regularizers for comparison. Intuitively, L_{hint} says that "the given digits must be predicted" and L_{sum} says that "the sum of the 9 probabilities in $\mathbf{X}$ in the same row/column/box must be 1".

$$L_{hint} = avg\Big(\mathbf{f} \odot (1 - b_p(\mathbf{x}))\Big)$$
$$L_{sum} = \sum_{\substack{s \in \{1, \ldots, 32\} \\ i \in \{row, col, box\}}} avg((sum(\mathbf{X}_s^i) - 1)^2).$$

Here, $avg(X)$ and $sum(X)$ compute the average and sum of all elements in X along its last dimension; $\mathbf{X}_s^{row}, \mathbf{X}_s^{col}, \mathbf{X}_s^{box} \in \mathbb{R}^{81 \times 9}$ are reshaped copies of $\mathbf{X}_s$ such that each row in, for example, $\mathbf{X}_s^{row}$ contains 9 probabilities for atoms $a(1, C, N), \ldots, a(9, C, N)$ for some C and N.

[6] The GNN is from https://www.kaggle.com/matteoturla/can-graph-neural-network-solve-sudoku, along with the dataset.

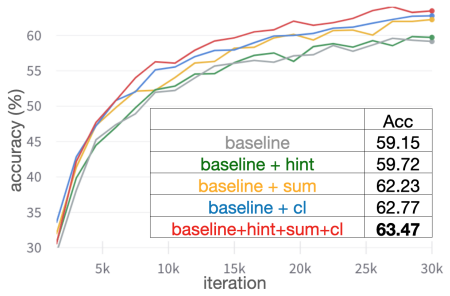

Figure 6. Acc with 30k dataset under different losses

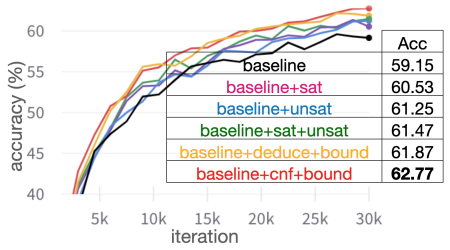

Figure 7. Acc with 30k dataset under different losses in L_{cl}

Figure 6 shows the test accuracy of the GNN after 20 epochs of training on 30k data instances (with full supervision) using different loss functions (denoted by subscripts of losses). It shows monotonic improvement from each loss and the best result is achieved when we add all losses.

Figure 7 further shows the monotonic improvement from each component in

$$L_{cl} = L_{deduce} + L_{sat} + L_{unsat} + 0.1 \times L_{bound}$$

where we split $L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f})$ in equation (16) into its 3 components. We can see that the most improvement comes from $L_{deduce} + 0.1 \times L_{bound}$, which aligns with Proposition 4.3 since L_{deduce} has dominant gradients that enforces a deduction step. Noticeably, L_{bound} is necessary for L_{deduce} to bound the size of raw NN output.

Figure 8 shows the test accuracy of the GNN after 60 epochs of training on 60k data instances (with full supervision). We can see that the monotonic improvement from each loss is kept in the experiments with 60k data instances and the best result is still achieved when we add all losses. However, the most improvement is from L_{sum} instead of L_{cl}. This is because most semantic information in L_{cl} are from L_{deduce} (i.e., one step deduction from the given digits), which can be eventually learned by the GNN with more data instances.

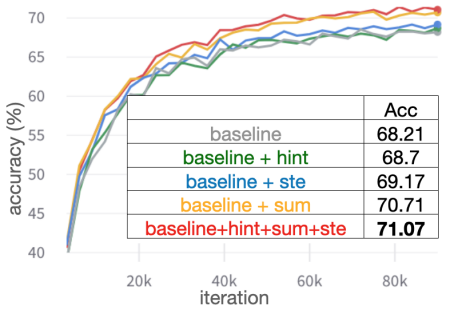

Figure 8. Acc with 60k dataset under different losses

E. More Examples

E.1. Learning to Solve the Shortest Path Problem

The experiment is about, given a graph and two points, finding the shortest path between them. We use the dataset from (Xu et al., 2018), which was used to demonstrate the effectiveness of semantic constraints for enhanced neural network learning. The dataset is divided into 80/20 train/test examples. Each example is a 4 by 4 grid $G = (V, E)$, where $|V| = 16, |E| = 24$, two-terminal (i.e., source and the destination) nodes are randomly picked up from 16 nodes, and 8 edges are randomly removed from 24 edges to increase the difficulty. The dataset consists of 1610 data instances, each is a pair $\langle \mathbf{i}, \mathbf{l} \rangle$ where $\mathbf{i} \in \{0,1\}^{40}$ and $\mathbf{l} \in \{0,1\}^{24}$. The ones in the first 24 values in $\mathbf{i}$ denote the (non-removed) edges in the grid, the ones in the last 16 values in $\mathbf{i}$ denote the terminal nodes, and ones in $\mathbf{l}$ denote the edges in the shortest path.

We define a CNF with 40 atoms and 120 clauses to represent "each terminal node is connected to exactly one edge in the shortest path". To start with, let's identify each node in the 4×4 grid by a pair (i, j) for $i, j \in \{1, \ldots, 4\}$ and identity the edge between nodes (i_1, j_1) and (i_2, j_2) as $((i_1, j_1), (i_2, j_2))$. Then, we introduce the following 2 atoms.

- $terminal(i, j)$ represents that node (i, j) is one of the two terminal nodes.
- $sp((i_1, j_1), (i_2, j_2))$ represents edge $((i_1, j_1), (i_2, j_2))$ is in the shortest path.

Then, the CNF for the shortest path problem consists of 120 clauses: 16 clauses of the form

$$\neg terminal(i_1, j_1) \vee \bigvee_{\substack{i_2, j_2: \\ ((i_1,j_1),(i_2,j_2)) \\ \text{is an edge}}} sp((i_1, j_1), (i_2, j_2))$$

for $i_1, j_1 \in \{1, \ldots, 4\}$, and 104 clauses of the form

$$\neg terminal(i_1, j_1) \vee \neg sp((i_1, j_1), (i_2, j_2)) \vee \neg sp((i_1, j_1), (i_3, j_3))$$

for $i_*, j_* \in \{1, \ldots, 4\}$ such that $((i_1, j_1), (i_2, j_2))$ and $((i_1, j_1), (i_3, j_3))$ are different edges.

This CNF can be represented by a matrix $\mathbf{C} \in \{-1, 0, 1\}^{120\times 40}$.

To construct $\mathbf{f}$ and $\mathbf{v}$ for a data instance $\langle \mathbf{i}, \mathbf{l} \rangle$, the facts $\mathbf{f} \in \{0, 1\}^{40}$ is simply the concatenation of $\mathbf{i}[24:]$ and $\{0\}^{24}$; while the prediction $\mathbf{v}$ is a vector in $\{0, 1\}^{40}$ obtained as follows. We (i) feed $\mathbf{i}$ into the same MLP from (Xu et al., 2018) and obtain the NN output $\mathbf{x} \in [0, 1]^{24}$ consisting of probabilities, (ii) extend $\mathbf{x}$ with 16 0s (in the beginning) so as to have a 1-1 correspondence between 40 elements in $\mathbf{x}$ and 40 atoms in the CNF, and (iii) $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b_p(\mathbf{x})$.

Finally, the total loss function $\mathcal{L}_{base}$ used in the baseline is

$$\mathcal{L}_{base} = L_{cross}(\mathbf{x}, \mathbf{l})$$

where L_{cross} is the cross-entropy loss.

The loss function $\mathcal{L}$ used for shortest path problem is

$$\mathcal{L} = L_{cross}(\mathbf{x}, \mathbf{l}) + \alpha L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + \beta L_{bound}(\mathbf{x})$$

where $\alpha = 0.2$ and $\beta = 1$. We set $\alpha = 0.2$ in our experiments to balance the gradients from the CNF loss and cross entropy loss. Indeed, a similar accuracy can be achieved if we compute α dynamically as $\frac{g_{cross}}{g_{cnf}}$ where g_{cnf} and g_{cross} are the maximum absolute values in the gradients $\frac{\partial L_{cnf}(\mathbf{C},\mathbf{v},\mathbf{f})}{\partial \mathbf{x}}$ and $\frac{\partial L_{cross}(\mathbf{x},\mathbf{l})}{\partial \mathbf{x}}$ respectively. Intuitively, the weight α makes sure that the semantic regularization from CL-STE won't overwrite the hints from labels.

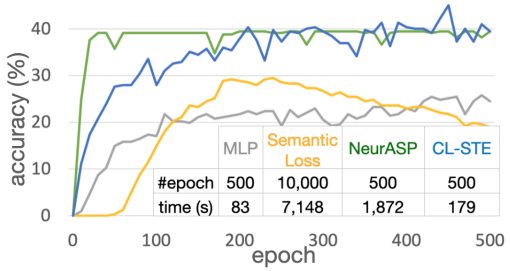

Figure 9. MLP+CL-STE on Shortest Path Problem

Figure 9 compares the test accuracy of the same Multi-Layer Perceptron (MLP) trained by different learning methods during 500 epochs of training (except that the accuracy for Semantic Loss method is reported for 10k epochs). As we can see, it only took 83s for baseline and 179s for CL-STE to complete all 500 epochs (including the time to compute training and testing accuracy) since they are all trained on GPU with a batch size of 32. Besides, CL-STE achieves comparable accuracy to NeurASP with only about $\frac{1}{10}$ of time. The training time of Semantic Loss in Figure 9 was recorded when it was trained on CPU. We re-did the Semantic Loss experiment on GPU with early stopping and found that it still takes 1032s to achieve the highest accuracy 30.75% after 2900 epochs of training.

E.2. Semi-Supervised Learning for MNIST and FASHION Dataset

Xu *et al.* (2018) show that minimizing semantic loss could enhance semi-supervised multi-class classification results by enforcing the constraint that a model must assign a unique label even for unlabeled data. Their method achieves state-of-the-art results on the permutation invariant MNIST classification problem, a commonly used testbed for semi-supervised learning algorithms, and a slightly more challenging problem, FASHION-MNIST.

For both tasks, we apply $b(x)$+iSTE to the same MLP (without softmax in the last layer) as in (Xu et al., 2018), i.e., an MLP of shape (784, 1000, 500, 250, 250, 250, 10), where the output $\mathbf{x} \in \mathbb{R}^{10}$ denotes the digit/cloth prediction.

The CNF for this problem consists of 46 clauses: 1 clause

$$pred(i, 0) \vee \cdots \vee pred(i, 9)$$

and 45 clauses of the form

$$\neg pred(i, n_1) \vee \neg pred(i, n_2)$$

for $n_1, n_2 \in \{0, \ldots, 9\}$ such that $n_1 < n_2$. Intuitively, these 2 clauses define the existence and uniqueness constraints on the label of image i. This CNF can be represented by the matrix $\mathbf{C} \in \{-1, 0, 1\}^{46\times 10}$.

The vectors $\mathbf{f}$ and $\mathbf{v}$ are constructed in the same way for both supervised data instance $\langle i, l \rangle$ and unsupervised data instance $\langle i \rangle$. The facts $\mathbf{f}$ is simply $\{0\}^{10}$; while the prediction $\mathbf{v}$ is a vector in $\{0, 1\}^{10}$ obtained as follows. We (i) feed image i into the CNN and obtain the outputs $\mathbf{x} \in \mathbb{R}^{10}$ (consisting of real values not probabilities); and (ii) $\mathbf{v} = \mathbf{f} + \mathbb{1}_{\{0\}}(\mathbf{f}) \odot b(\mathbf{x})$. Then, the total loss for unsupervised data instances is defined as

$$\mathcal{L} = L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) + L_{bound}(\mathbf{x}),$$

which enforces that each image should map to exactly one digit or one cloth type. The total loss for supervised data instance simply contains $\mathcal{L}$ as well as the typical cross-entropy loss.

We train the network using 100, 500, and 1,000 partially labeled data and full (60,000) labeled data, respectively. We run experiments for 50k batch updates with a batch size of 32. Each experiment is repeated 10 times, and we report the mean and the standard deviation of classification accuracy (%).

Table 6 shows that the MLP with the CNF loss achieves similar accuracy with the implementation of semantic loss from (Xu et al., 2018). Time-wise, each experiment using the method from (Xu et al., 2018) took up about 12 minutes, and each experiment using the CL-STE method took about 10 minutes. There is not much difference in training time

Table 6. Accuracy on MNIST & FASHION dataset

Method	Number of labeled examples used			
	100	500	1000	All (60,000)
MNIST (Xu et al.)	85.3±1.1	94.2±0.5	95.8±0.2	98.8±0.1
MNIST ($b(x)$+iSTE)	84.4±1.5	94.1±0.3	95.9±0.2	98.8±0.1
FASHION (Xu et al.)	70.0±2.0	78.3±0.6	80.6±0.3	87.3±0.2
FASHION ($b(x)$+iSTE)	71.0±1.2	78.6±0.7	80.7±0.5	87.2±0.1

since the logical constraints for this task in the implementation of semantic loss (Xu et al., 2018) are simple enough to be implemented in python scripts without constructing an arithmetic circuit and inference on it.