DeepSaDe: Learning Neural Networks that Guarantee Domain Constraint Satisfaction

Kshitij Goyal[a], Sebastijan Dumancic[b] and Hendrik Blockeel[a]

Abstract.   As machine learning models, specifically neural networks, are becoming increasingly popular, there are concerns regarding their trustworthiness, especially in safety-critical applications, e.g., actions of an autonomous vehicle must be *safe*. There are approaches that can train neural networks where such domain requirements are enforced as constraints, but they either cannot guarantee that the constraint will be satisfied by all possible predictions (even on unseen data) or they are limited in the type of constraints that can be enforced. In this work, we present an approach to train neural networks which can enforce a wide variety of constraints and guarantee that the constraint is satisfied by all possible predictions. The approach builds on earlier work where learning linear models is formulated as a constraint satisfaction problem (CSP). To make this idea applicable to neural networks, two crucial new elements are added: constraint propagation over the network layers, and weight updates based on a mix of gradient descent and CSP solving. Evaluation on various machine learning tasks demonstrates that our approach is flexible enough to enforce a wide variety of domain constraints and is able to guarantee them in neural networks.

1   Introduction

Widespread use of state-of-the-art machine learning (ML) techniques has given rise to concerns regarding the trustworthiness of these models, especially in safety-critical and socially-sensitive domains. For example, in autonomous vehicles that employ ML approaches to predict the next action, the actions must be safe. Such domain requirements can often be formulated as logical constraints on combinations of inputs and outputs (e.g., whenever the input satisfies some condition A, the output must satisfy condition B). Crucially, these domain constraints must be satisfied for all possible inputs to the model, not just the training data. This has motivated researchers to develop approaches that can train ML models that satisfy a given constraint for all possible predictions.

A general approach to enforcing constraints in ML models is to include a regularization term in the cost function, which typically adds a cost for every violation of a constraint in the training set (e.g., [31, 7]). Such an approach can reduce the number of violations in the training set, but it does not necessarily eliminate them. Moreover, even when it does, this does not guarantee that other instances (outside the training set) cannot violate the constraint. Alternatively, for some model types, such as neural networks, the architecture of the model can be chosen in such a way that certain types of constraints are guaranteed to be satisfied for each possible input (not just training data) ([26, 17]). But this is typically possible only for specific combinations of model and constraint types.

This raises the question of whether generality and certainty can be combined. Is it possible to come up with a *generally applicable* approach that guarantees the satisfaction of constraints not only on the training set but on the *entire* input space, and this for any kind of model? A step in this direction was made by sade, who propose a relatively general solution for linear models. Their approach translates the learning problem into a MaxSMT setting. MaxSMT stands for Maximum Satisfiability Modulo Theories. It is an extension of SAT solving that can take background theories into account (e.g., for reasoning about the real numbers) and that distinguishes soft and hard constraints: it returns a solution that satisfies all hard constraints and as many soft constraints as possible. sade model the requirements as hard constraints and maximize the fit to the data using soft constraints. Their approach works for a wide range of constraint types but only handles linear models, and assumes a bounded input domain.

In this paper, we substantially extend the applicability of that approach by showing how it can be used to train feedforward neural networks. Two key modifications to the network's architecture and training procedure suffice for achieving this: (1) propagating the constraints over the network layers to the last layer, which involves adding skip connections ([15]) that copy the input to the penultimate layer and deriving bounds on the penultimate layer from the bounds on the input ([27]), and (2) training the network using a hybrid procedure that uses MaxSMT to determine the weights in the output layer and gradient descent for all other weights. We demonstrate that with these changes, neural networks can be trained that have good performance and guarantee the satisfaction of given constraints. In the following, we first describe the problem setting, then briefly describe the existing approach that we build on, before detailing our approach. Afterward, we compare our approach to related work and evaluate it experimentally.

2   Problem Statement

In this paper, we focus on semantic constraints which constrain the behavior of the model: the predictions are required to adhere to certain requirements, e.g. safety constraints ([19]), structured output constraints ([31]), and hierarchical constraints ([17]). Specifically, we focus on domain constraints: constraints that must hold for all instances in the domain. We assume that the constraints can be written with universally quantified logic formulas. In particular, we consider

the constraints of the form:

$$K : \forall \mathbf{x} \in \bigtimes_{i=1}^{n} [l_i, u_i], \mathbf{x} \models P \Rightarrow f_\mathbf{w}(\mathbf{x}) \models C \tag{1}$$

Which states that if an input $\mathbf{x}$ satisfies a condition P, the output must satisfy a condition C. An example of a safety constraint that can be represented in this way is: "if an object comes in front of a moving vehicle, the vehicle must stop". Solving for such a constraint exactly using specific constraint solvers allows us to enforce the constraint for all possible inputs. The reason for focusing on this type of constraints is mostly practical: the constraint solving technology we use was found to scale well enough for these types of constraints. It is not a theoretical limitation: any constraint that can be handled effectively by current constraint solving technology can be handled by the approach we develop. As demonstrated later in section 6, our chosen constraint formulation (equation 1) already provides us with a variety of tasks to work with.

To make the search procedure tractable, and because features in ML problems are typically bounded (e.g., a pixel in an image takes a value in $[0, 255]$), we use bounded domains $\bigtimes_{i=1}^{n} [l_i, u_i]$ instead of $\mathbb{R}^n$. As a natural choice, the training data $D \subseteq \mathcal{X} \times \mathcal{Y}$ can be used to calculate these bounds, i.e. $l_i = \min_D(\mathbf{x}_i)$ & $u_i = \max_D(\mathbf{x}_i)$. We rely on the SMT solver Z3 ([24]) to solve such constraints. Other prominent constraint-solving paradigms like MILP and CP do not support such constraints over continuous domains ([25]).

We are now ready to formulate our problem statement:

Definition 1. Learning problem. *Given* a training set $D \subseteq \mathcal{X} \times \mathcal{Y}$, a set of domain constraints $\mathcal{K}$, a loss function $\mathcal{L}$, and a hypothesis space containing functions $f_\mathbf{w} : \mathcal{X} \to \mathcal{Y}$; *find* $\mathbf{w}$ such that $f_\mathbf{w}$ satisfies constraints in $\mathcal{K}$ and $\mathcal{L}(f_\mathbf{w}, D)$ is minimal among all such $f_\mathbf{w}$.

$f_\mathbf{w}$ is assumed to be a feedforward neural network. The output layer is real-valued without any activation, and a softmax layer is used to calculate class probabilities for classification problems. The language of the constraints in $\mathcal{K}$ is a subset of first-order logic which allows for universal quantifiers. In practice, we focus on the constraints of the form in equation 1, and $\mathcal{K}$ can be a set of multiple such universally quantified constraints.

3 Background - Satisfiability Descent (SaDe)

Satisfiability (SAT) is the problem of finding a solution that satisfies a given Boolean formula, e.g. $\neg a \lor b$. Satisfiability Modulo Theories (SMT) extend SAT such that formulas can be expressed in other theories, such as Real Arithmetic, e.g., $(a + b > 3) \land (b < 1)$ with real a and b. Maximum Satisfiability (MaxSMT) generalizes the SMT problem: given a set of hard constraints $\mathcal{H}$ and soft constraints $\mathcal{S}$, it aims to find a solution that satisfies all constraints in $\mathcal{H}$ and as many as possible in $\mathcal{S}$.

Our work builds on *SaDe* ([11]) which is a learning algorithm that can enforce constraints in linear models and guarantee satisfaction. SaDe modifies the parameter update process of the mini-batch gradient descent ([9]). Unlike gradient descent, which updates the solution in the direction that minimizes the loss, SaDe solves a MaxSMT problem to find the solution at each iteration, which is formulated in such a way that its solution satisfies the domain constraint and is close to the solution that gradient descent might lead to. At each iteration, a local search space is defined for the MaxSMT problem around the previous solution. This search space is a fixed-sized n-cube where each edge of the n-cube is a hyperparameter called *maximal step size* that upper

bounds the size of the update in each dimension (illustrated for two parameters in figure 1(a)). SaDe iteratively improves the performance while learning solutions that satisfy the constraint. Formulations of the MaxSMT problem and the local search space at each iteration are provided next.

Formulation of MaxSMT problem: A MaxSMT problem is defined for a batch of instances in each iteration, where the soft constraints encode a certain quality of fit of the model on the instances in the batch, and the domain constraints are hard constraints. A soft constraint, for a given instance $(\mathbf{x}, y)$, is defined as a logical constraint that is fulfilled when the prediction of the model $f_\mathbf{w}$ for $\mathbf{x}$ is "sufficiently consistent" with the true label y. For regression, given some error e, a soft constraint is defined as: $|y - e| \leqslant f_\mathbf{w}(\mathbf{x}) \leqslant |y + e|$. For classification, the sign of $f_\mathbf{w}(\mathbf{x})$ is assumed to be the indicator of the class and the magnitude indicates the certainty of prediction, the soft constraint takes the form (for a threshold τ):

$$f_\mathbf{w}(\mathbf{x}) > \tau \qquad \text{if } y = 1$$
$$f_\mathbf{w}(\mathbf{x}) < -\tau \qquad \text{if } y = -1$$

For each instance, multiple soft constraints are formulated for different values of the error e or threshold τ. Satisfying a maximum of these soft constraints, which is what the MaxSMT problem tries to achieve, correlates with minimizing the prediction loss. Thus, the solution to this MaxSMT problem at every iteration reduces the loss while satisfying the domain constraints.

Local search space: At every iteration, the MaxSMT problem searches for the next solution in a local search space defined by the n-cube around the previous solution. It is encoded as an additional hard constraint in the MaxSMT problem, with a "box" constraint, which states that the next solution must be inside the axis-parallel box defined by $\hat{\mathbf{w}}$ and $\hat{\mathbf{w}} - \alpha \cdot \text{sgn}(\mathbf{g})$, where the sign function is applied component-wise to a vector (a modified sign function is used where $\text{sgn}(0) = 1$), $\hat{\mathbf{w}}$ is the previous solution, $\mathbf{g}$ is the gradient of the loss at $\hat{\mathbf{w}}$, and α is the maximal step size. The box constraint serves two important purposes. Firstly, it provides a general direction in which the loss is minimized, as the box in each dimension is aligned with the negative gradient. Secondly, it stabilizes learning by limiting the size of the updates. Interestingly, using a regularized loss instead of a standard loss has no impact as the constraint is satisfied at each step of training, making the regularization 0.

SaDe is limited to training linear models. Training neural networks with the same procedure would require solving the MaxSMT problem with highly non-linear soft constraints, e.g. $|y - e| \leqslant f_\mathbf{w}(\mathbf{x}) \leqslant |y + e|$ where $f_\mathbf{w}(\mathbf{x})$ is a neural network, which is not possible with state-of-the-art SMT solvers.

4 DeepSaDe: Deep Satisfiability Descent

We now present our approach *DeepSaDe*, which utilizes the MaxSMT framework proposed in sade to train neural networks with constraints. DeepSaDe exploits the structure of neural networks, which transform the input domain through a series of non-linear layers before a final linear layer maps it to the output. As the network output only explicitly depends on the last layer, enforcing the constraint on the last layer is sufficient to enforce the constraint on the network. DeepSaDe, therefore, uses batch learning with a hybrid procedure that uses MaxSMT to determine the weights in the last layer and gradient descent for all other weights. In the following, we introduce some notation before we formalize the MaxSMT problem in the last layer in section 4.1, and then detail the learning algorithm in section 4.2.
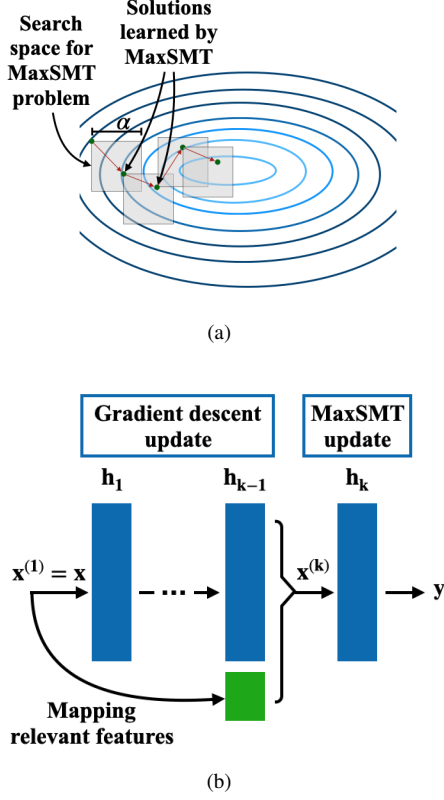
Figure 1: **(a.) Illustration of SaDe:** each grey quadrant represents the local search space for the MaxSMT problem (*maximal step size: α*), defined by the gradients of the loss, the green points are the solutions found with MaxSMT; **(b.) DeepSaDe Architecture:** Last layer is updated using the MaxSMT framework and the layers before are updated with gradient descent. The input features relevant to the domain constraint (in green) are mapped to the penultimate layer via skip connections.

$f_{\mathbf{w}}$ is a fully-connected neural network with k layers such that $f_{\mathbf{w}}(\mathbf{x}) = h_k(h_{k-1}(...h_1(\mathbf{x})...))$ for input $\mathbf{x} \in \mathcal{X}$, where h_n is the nth layer. The input to the nth layer is $\mathbf{x}^{(n)} = h_{n-1}(...h_1(\mathbf{x})...)$, where $\mathbf{x}^{(1)} = \mathbf{x}$ and the latent input space for the h_n is represented by $\mathcal{X}^{(n)}$, where $\mathcal{X}^{(1)} = \mathcal{X}$. The size of a layer, $|h_n|$, is the number of neurons in it. Weight and bias parameters of h_n are denoted by matrices $W^{(n)}$ and $B^{(n)}$ of dimensions $|h_{n-1}| \times |h_n|$ and $1 \times |h_n|$ respectively ($|h_0| = |\mathbf{x}^{(1)}|$). Elements of $W^{(n)}$ and $B^{(n)}$ are referred with lower-case letters.

4.1 Formulation of the maximum satisfiability problem at the last layer of the network

To formalize the MaxSMT problem for the last layer, it is necessary to express both the soft constraints and the hard constraint (domain constraint) in terms of the latent inputs $\mathbf{x}^{(k)}$ rather than the original inputs $\mathbf{x}^{(1)}$. The soft constraint at the last linear layer can be formulated in a similar way to what was presented in section 3. Specifically, for a regression task, the constraint is given by $|y - e| \leqslant h_k(\mathbf{x}^{(k)}) \leqslant |y + e|$, whereas for classification, it takes the form of:

$$h_k(\mathbf{x}^{(k)}) > \tau \qquad \text{if } y = 1$$
$$h_k(\mathbf{x}^{(k)}) < -\tau \qquad \text{if } y = -1$$

To formulate the hard domain constraints, we propose a method for translating each original constraint K (equation 1) into a constraint K' for the MaxSMT problem at the last layer, given the parameters of layers $h_1, \ldots, h_{k-1}$, such that a solution to K', combined with $h_1, \ldots, h_{k-1}$, is a solution to K.

$$\mathrm{K}' : \forall \mathbf{x}' \in \bigtimes_{i=1}^{|h_{k-1}|} [l_i^{(k)}, u_i^{(k)}], \mathbf{x}' \models P \Rightarrow h_k(\mathbf{x}') \models C \qquad (2)$$

Here $\mathbf{x}'$ represent the quantified latent variable in the domain of $\mathcal{X}^{(k)}$ bounded by $\bigtimes_{i=1}^{|h_{k-1}|}[l_i^{(k)}, u_i^{(k)}]$. We explain how this translation is achieved in the next two paragraphs. Given a network, with this translation, the last layer of a network can be updated to satisfy the constraint by the network.

Domain Bound Propagation: To formulate K', we first consider the bounds of the quantified variable for the latent space of $\mathcal{X}^{(k)}$. The bounds of the latent space must be such that enforcing the constraint within these bounds enforces the constraint on the original input bounds. To construct such latent bounds, we rely on interval arithmetic ([27]). This involves calculating the bounds of the output of a layer based on the bounds of the input and the layer parameters, such that any input to the layer takes the output value within the output bounds. Given the lower and upper bounds $l^{(n)}$ and $u^{(n)}$ for the input $\mathbf{x}^{(n)}$ of the layer h_n, following the approach used in gowal2018effectiveness, the bounds for the output $\mathbf{x}^{(n+1)}$ are computed as (more details in Appendix A.1):

$$l_i^{(n+1)} = \text{act}(b_i^{(n)} + \sum_{j:w_{j,i}^{(n)} \geqslant 0} w_{j,i}^{(n)} l_j^{(n)} + \sum_{j:w_{j,i}^{(n)} < 0} w_{j,i}^{(n)} u_j^{(n)})$$
$$u_i^{(n+1)} = \text{act}(b_i^{(n)} + \sum_{j:w_{j,i}^{(n)} \geqslant 0} w_{j,i}^{(n)} u_j^{(n)} + \sum_{j:w_{j,i}^{(n)} < 0} w_{j,i}^{(n)} l_j^{(n)}) \qquad (3)$$

Where 'act' is the activation function (like ReLu, Sigmoid, and Tanh). Given the bounds of the input space, the bounds for the latent space $\mathcal{X}^{(k)}$ can be calculated recursively. Enforcing a constraint within the latent bounds enforces the constraint on any input within the input bounds.

Identity Mapping of Relevant Features: The translation also takes into account that some domain constraints may be dependent on the input space, e.g., "*if an object comes in front of the vehicle, the vehicle must stop*". To encode such conditions in K' (i.e., $\mathbf{x}' \models P$), we make the relevant features, i.e. the features that are needed to encode the property P, available at the latent space $\mathcal{X}^{(k)}$. For this, we use skip-connections ([15]) that map these features to the second to last layer h_{k-1} using an identity mapping, as illustrated in figure 1(b). The network, consequently, is no longer fully-connected and these features take the same value as the input. These mapped features become a part of $\mathcal{X}^{(k)}$ and the input property can be expressed at the last layer. These features are identified in advance. Finally, as the output of the network is the same as the output of the last layer, the constraint on the network output can be encoded with $h_k(\mathbf{x}')$. This completes the formulation of K'.

4.2 Learning Algorithm

We now present the algorithm while referring to the pseudo-code in algorithm 1. DeepSaDe modifies mini-batch gradient descent ([9]), where forward and backward pass at every iteration (lines 5-7) are kept the same, but the parameter update is split into two parts. First, only the last layer's parameters are updated to satisfy the domain

Algorithm 1 Deep Satisfiability Descent (DeepSaDe)

input: training data D, validation data V, domain constraints $\mathcal{K}$, batch size b, epochs e, loss $\mathcal{L}$, maximal step size $\boldsymbol{\alpha}$, learning rate $\boldsymbol{\eta}$, line search steps $\mathbf{s}$

1: $f_{\hat{W}}(.)$: Model initialized with a standard approach
2: $\mathcal{K}' = \text{TranslateDomainConstraint}(\mathcal{K}, f_{\hat{W}}(.))$
3: Update $\hat{W}$ such that $\hat{W}^{(k)} \models \mathcal{K}'$
4: restart = False; $\hat{W}_{out}$ = undefined; Partition D into batches of size b
5: **for** each epoch **do**
6:    **for** each batch B in D **do**
7:      $\mathcal{G} = \nabla \mathcal{L}(\hat{W})$
8:      $\mathcal{K}' = \text{TranslateDomainConstraint}(\mathcal{K}, f_{\hat{W}}(.))$
9:      **if** restart **then**
10:        Randomly flip the sign of each element of $\mathcal{G}^{(k)}$
11:      **end if**
12:      $\hat{W}^{(k)}$ = undefined
13:      *// try line search, if no solution use MaxSMT*
14:      let $W_i = \hat{W}^{(k)} - \eta \mathcal{G}^{(k)} i/s$, for $i = 1, 2, \ldots, s$
15:      **if** $\{i \mid W_i \models \mathcal{K}'\} \neq \varnothing$ **then**
16:        $\hat{W}^{(k)} = \hat{W}_{\max\{i \mid W_i \models \mathcal{K}'\}}$
17:      **else**
18:        $\mathcal{G}_S = sgn(\mathcal{G}^{(k)})$
19:        $\mathcal{S} = \text{SoftConstraints}(h_{k-1}(...h_1(B)...))$
20:        $\mathcal{H} = \{W^{(k)} \in \text{Box}(\hat{W}^{(k)}, \hat{W}^{(k)} - \alpha \mathcal{G}_S)\} \cup \mathcal{K}'$
21:        *// assuming that MaxSMT returns undefined if unsatisfiable*
22:        $\hat{W}^{(k)} = \text{MaxSMT}(\mathcal{S}, \mathcal{H})$
23:      **end if**
24:      **if** $\hat{W}^{(k)}$ is undefined **then**
25:        *// no solution found, restart in another direction*
26:        restart = True
27:      **else**
28:        *// remember best solution found*
29:        **if** $eval(f_{\hat{W}}, V) > eval(f_{\hat{W}_{out}}, V)$ **then**
30:          $\hat{W}_{out} = \hat{W}$
31:        **end if**
32:        *// perform gradient descent on earlier layers*
33:        **for** $n = 1, ..., k-1$ **do**
34:          $W^{(n)} = \hat{W}^{(n)} - \eta * \mathcal{G}^{(n)}$
35:        **end for**
36:        restart = False
37:      **end if**
38:    **end for**
39: **end for**
40: **return** $f_{\hat{W}_{out}}(.)$

constraint using the MaxSMT formulation presented in section 4.1 (lines 8-27) (details in next paragraph). Second, the earlier layers are updated using gradient descent to optimize predictive loss (lines 33-35). Importantly, after the latter update, the network does not guarantee constraint satisfaction due to changes in the latent space of the last layer, used to formulate the MaxSMT problem. Therefore, the network before the latter update is used for evaluation and a validation set is used to select the best model (lines 29-31).

For updating the last layer, first, a line search is used along the vector that minimizes the prediction loss, and a fixed number of candidate solutions, from furthest to closest, are checked if they satisfy $\mathcal{K}'$ and first one that does is picked (lines 14-16). If this search does

not yield a solution, the MaxSMT problem is formulated based on the inputs to the last layer (section 4.1), and a solution is searched in the local search space around the previous solution defined based on the gradients calculated during the backward pass (lines 18-23) (similar to the local search space in SaDe). Line search is employed first because checking if a point satisfies a constraint is faster than searching for the solution in a domain, finding a solution by merely checking a few candidate points speeds up learning.

Sometimes a solution for the last layer cannot be found because the MaxSMT problem could not be solved within the local search space, possibly due to the gradient pointing to a solution space that violates the constraint. In such cases, a restart procedure is initiated where the signs of the gradients are randomly flipped (lines 9-11, 24-26) to randomize the direction of the update. This may slightly decrease predictive performance, but it effectively restarts the learning process when it gets stuck.

Since our approach is iterative, starting from an initial configuration that satisfies the domain constraint is crucial. Otherwise, we may begin in a solution space far from the constrained space, leading to no updates. We ensure this by first initializing the network using a standard method ([14]) (line 1), and then updating the weights of the last layer that satisfy the translated constraint $\mathcal{K}'$ (lines 2-3). It is important to note that we are solving a satisfiability problem here, not the maximum satisfiability problem, as we are not using any soft constraints for this purpose.

5   Related Work

A standard approach for enforcing constraints in ML models is *regularization*, where a penalty is added to the prediction loss whenever the model makes a prediction that violates the constraint $((1 - \lambda) * \text{loss} + \lambda * \text{regularization})$. xu2018semantic propose a regularization defined on the weighted model count (WMC) ([4]) of the constraint defined over the network output given the predicted probabilities. diligenti2017semantic and serafini2016logic propose a fuzzy logic-based regularization for constraints in first-order logic. There are many other regularization approaches in the literature, e.g. fischer2019dl2, hu2016harnessing, stewart2017label. Regularization can enforce a variety of constraints, but it does not guarantee constraint satisfaction. Additionally, high regularization loss with a large value of λ may provide stronger constrain satisfaction but impacts the predictive performance negatively.

Some approaches guarantee constraints by construction but are generally limited to enforcing specific types of constraints. For example, monotonic lattices ([13]), deep lattice networks ([32]), and COMET ([26]) are approaches to enforce monotonicity in neural networks; leino2022self & lin2020art propose approaches to satisfy some safety specifications. A more general approach, MultiplexNet, was proposed in hoernle2022multiplexnet. They use a multiplexer layer to satisfy constraints in disjunctive normal form (DNF). However, DNF representation is limiting as certain constraints have worst-case representations in DNF which may lead to exponentially many terms. Additionally, constraints conditioned on the input space cannot be enforced. Another approach, DeepProbLog ([22]), trains neural networks within the ProbLog framework, where constraints can be enforced with ProbLog. However, it is limited to modeling discrete variables, and cannot model regression problems.

Our work relates to combinatorial optimization approaches as we use a MaxSMT-based approach. These approaches, however, except sade, are limited to discrete models like decision trees ([12, 2, 28, 6]) and decision sets ([33, 18]). Maximum satisfiability, specifically,

has also been used in various ML tasks ([1, 5, 21]). None of these, however, focus on training neural networks.

There are approaches that rely on the idea of bound propagation, also used in our work, to train adversarially robust neural networks ([10, 34, 30]). However, the constraints that can be enforced are limited to input-output bounds (if the input is in a given bound, the output should be in a specific bound). Our approach is more general and can, in theory, handle any constraint that can be written as an SMT formula, e.g., structured output constraints.

Finally, there are approaches to verify if a network satisfies a constraint ([20, 29, 3]). DeepSaDe trains networks that do not require verification because constraints are guaranteed by construction.

6 Experiments

We evaluate multiple use cases in various ML tasks with complex domain constraints. We first outline the research questions, then describe the use cases, our evaluation method, and the results.

Q1: Can existing methods satisfy domain constraints for all predictions in practice, even if they do not guarantee it?

Q2: How does the predictive performance of DeepSaDe models compare to the baselines?

Q3: Do the DeepSaDe models have a higher training time compared to the baselines?

6.1 Use Cases

UC1, UC2 & UC3 are from sade, UC4 is novel, and UC5 is from xu2018semantic.

UC1: A multi-target regression to predict 5 household expenses using 13 attributes, with 41417 data instances. We enforce two constraints: *"sum of all the expenses must be smaller than the total household income"* and *"going out expense must be smaller than 5% of the household income"*.

UC2: A binary classification problem of predicting if a person should be given a loan or not based on 13 attributes, with 492 data instances. We enforce the constraint: *"a person with a salary less than 5000\$ and an absent credit history must be denied a loan"*.

UC3: A multiclass classification problem to classify a song to one of 5 music genres based on 13 attributes, with 793 data instances. We enforce the constraint: *"a song by 'the Beatles' must be classified as either rock or pop"*.

UC4: A multi-label classification problem of identifying the labels from a sequence of 4 MNIST images. We enforce the constraint: *"the sum of the predicted labels must be greater than 10"*. 20000 instances are generated by selecting 4 images at random from the MNIST dataset.

UC5: A preference learning problem to predict the preference order of different sushi, with constraint: *"the prediction must have a coherent preference order"*. The preference order of 6 out of 10 sushi is used to predict the preference order of the remaining 4 sushi. The dataset contains 4926 instances. The preference ordering over n items is encoded as a flattened binary matrix $\{X_{ij}\}$ where X_{ij} denotes that item i is at position j. Under this encoding, each instance has 36 features and 16 targets.

6.2 Evaluation Methodology

Evaluation: Constraint satisfaction is typically evaluated by the *constraint accuracy* metric used in xu2018semantic and fischer2019dl2, which corresponds to the percent of instances where the constraint is not violated by the prediction. Such an evaluation, however, is limited to a finite sample of the population. Hence, as a second measure, we also calculate the *Adversity Index (AdI)* ([11]), which is the fraction of instances for which a counter-example to the constraint can be constructed in the neighborhood, defined by an l_∞ ball of radius δ around the instance. AdI takes a value between 0 and 1, a higher value of AdI implies that the model violates the constraint on more points similar to data instances. AdI is calculated on full data (training and test) because this provides more instances to evaluate constraint satisfaction; we want the constraints to be satisfied on all data, not only test data. For DeepSaDe, both constraint accuracy and AdI are zero by construction but we still calculate it as a sanity check. We use the neural network verification software *Marabou* ([20]) to find counter-examples. However, we only compute the AdI for UC1-3 because *Marabou* only handles inequality constraints. We don't know any verification tool that can verify the constraints in UC4-5. For predictive performance, we use MSE for UC1 and accuracy for UC2-3. For UC4-5, we use *coherent accuracy* which is the fraction of instances for which the model predicts the entire configuration correctly, *flattened accuracy* which is the fraction of individually correct binary labels, and the *Jaccard accuracy* which is the average Jaccard index for each multi-label prediction compared to true labels. Performance is only evaluated for instances where the true label does not violate the constraint as there can be data instances in practice where this happens (e.g. racially biased data to predict crime likelihood), comparison with such instances makes the evaluation biased.

Baselines: For UC2 and UC3, we use regularization baselines based on xu2018semantic (SL) and diligenti2017semantic (SBR). For UC1, we design a custom regularization loss REG (details in appendix A.2). For UC4, we could not find any approaches that can enforce such a constraint. Hence, we simply compare with a feedforward network (FFN). For UC5, we choose SL and FFN as baselines.

Experimental Setup: For solving the MaxSMT problem, we implement the Fu-Malik algorithm ([8]) over the Z3 solver for NRA (Quantified Nonlinear Real Arithmetic) formulas. We ran experiments on an Intel(R) Xeon(R) Silver 4214 CPU @ 2.20GHz machine with 125 GB RAM. For each use case, we run 5 experiments with 5-fold cross-validation, and the data is split 70/20/10 into train/test/validation. Every feature is scaled in [0, 1], and the radius $\delta = 0.1$ is chosen for AdI, which is significantly smaller than the mean ℓ_∞ distance between two points: this distance for UC1 is 0.75, for UC2 is 0.97, and for UC3 is 0.89. For regularization, the smallest value of λ, in [0, 1], that leads to minimum violations on the validation set is selected via cross-validation. Refer to Appendix A.2 for details on the architectures and hyper-parameters.

6.3 Results (Tables 1 & 2)

Constraint Satisfaction: DeepSaDe finds a model that achieves 100% constraint accuracy for each use case and AdI = 0 for UC1-3. For UC1, the constraint accuracy for REG is 93.5%, and counterexamples can be constructed close to 97% of the instances (AdI = 0.97). Similar behavior is seen for UC3 for both SL and SBR, with SBR proving to be more effective in enforcing constraints because of lower AdI. For UC2, SL and SBR both lead to 100% constraint accuracy, but counterexamples can still be constructed as AdI > 0. For UC4 and UC5, FFN fails to satisfy constraints on test set. For UC5, SL can satisfy the constraint but the predictive performance is much worse than DeepSaDe. Thus, in general, existing approaches, in contrast to DeepSaDe, *do not* satisfy domain constraint satisfaction in practice, which answers **Q1**.

UC	Approach	Constraint	AdI($\delta = 0.1$)	Accuracy/MSE	Runtime (sec)
UC1	DeepSaDe	$\mathbf{100}_{+0}$	$\mathbf{0}_{+0}$	*$38.36_{+4.59}$	102341_{+40198}
	REG	$93.50_{+2.01}$	$0.97_{+0.003}$	*$\mathbf{30.50}_{+\mathbf{6.49}}$	227_{+70}
UC2	DeepSaDe	$\mathbf{100}_{+0}$	$\mathbf{0}_{+0}$	$80.04_{+4.29}$	447_{+105}
	SL	$\mathbf{100}_{+0}$	$0.002_{+0.006}$	$\mathbf{80.17}_{+\mathbf{3.88}}$	45_{+30}
	SBR	$\mathbf{100}_{+0}$	$0.002_{+0.004}$	$80.04_{+3.95}$	45_{+27}
UC3	DeepSaDe	$\mathbf{100}_{+0}$	$\mathbf{0}_{+0}$	$80.11_{+4.99}$	6580_{+1915}
	SL	$99.97_{+0.03}$	$0.42_{+0.28}$	$\mathbf{82.53}_{+\mathbf{3.58}}$	101_{+45}
	SBR	$99.97_{+0.05}$	$0.29_{+0.10}$	$76.51_{+7.50}$	196_{+68}

Table 1: Results: UC1, UC2 & UC3 (*MSE, lower MSE value is better)

UC	Approach	Constraint	Coherent	Flattened	Jaccard	Runtime (sec)
UC4	DeepSaDe	$\mathbf{100}_{+0}$	$6.62_{+1.72}$	$78.52_{+1.73}$	$62.97_{+2.28}$	227928_{+30559}
	FFN	$88.00_{+3.26}$	$\mathbf{23.94}_{+\mathbf{4.25}}$	$\mathbf{85.81}_{+\mathbf{1.18}}$	$\mathbf{71.55}_{+\mathbf{2.40}}$	3215_{+2663}
UC5	DeepSaDe	$\mathbf{100}_{+0}$	$\mathbf{11.08}_{+\mathbf{2.61}}$	$67.17_{+1.48}$	$\mathbf{25.94}_{+\mathbf{2.89}}$	17586_{+5074}
	FFN	$0.04_{+0.15}$	$0.01_{+0.04}$	$\mathbf{75.69}_{+\mathbf{0.15}}$	$13.04_{+1.06}$	48_{+10}
	SL	$\mathbf{100}_{+0}$	$4.06_{+3.33}$	$63.16_{+2.62}$	$18.08_{+3.33}$	298_{+110}

Table 2: Results: UC4 & UC5

Predictive Performance: DeepSaDe treats the domain constraint as a hard constraint, which limits the solution space to the regions where the constraint is guaranteed. Thus, the predictive performance of DeepSaDe models can be worse than existing approaches which do not guarantee constraint satisfaction. For UC1, the predictive performance of DeepSaDe is slightly worse than REG, while the difference is not statistically significant for UC2 & UC3. The performance of DeepSaDe is worse for UC4 on all prediction metrics compared to FFN. For UC5, SL regularization with a high value of λ allows for satisfying all the constraints on the test set but performs much worse than DeepSaDe. To study this further, we plot the prediction loss (cross-entropy) for SL models on the test set for various λ between 0 and 0.9, averaged over 5 folds, in Figure 2. For comparison, the average loss for DeepSaDe is also plotted. DeepSaDe achieves constraint satisfaction in addition to having better performance than SL with a high λ. High regularization makes the prediction loss insignificant compared to the regularization loss, leading to worse predictive performance.
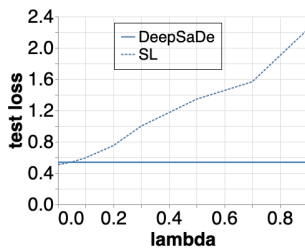

Figure 2: Test loss for UC5

In DeepSaDe, where the solution at every iteration is learned with MaxSMT, the solver tries to satisfy as many soft constraints as possible in addition to satisfying the domain constraints. DeepSaDe, thus, is a more stable learner compared to regularization with high λ. This answers **Q2**. Although the constraint satisfaction with DeepSaDe comes at the cost of predictive performance in some cases, in applications where constraints are crucial, like in safety-critical domains, this may be acceptable.

Training Time: DeepSaDe requires between 10 to 500 times more time than baselines across different use cases. This is because DeepSaDe solves a MaxSMT problem at each iteration, which makes the training slower than the numerical updates in the baselines. This positively answers the research question **Q3**. In applications where constraints are imperative, training time is less relevant, e.g., a network trained over a week and guarantees safety is still more valuable to the autonomous vehicle compared to one trained for a few hours but cannot do so. Our work is a starting point of such an approach that combines the exact solving of universal quantifiers with gradient-based learning to train neural nets. With further research into solver technology, it can be made more scalable. Additionally, for future work, possible modifications to improve the efficiency of DeepSaDe include using an incomplete MaxSMT approach like stochastic local search ([23]) instead of a complete one like Fu-Malik, and using compact bounds for the latent space ([29]).

7 Conclusion

We proposed DeepSaDe to train feedforward neural networks which can enforce a variety of constraints and guarantee constraint satisfaction for all possible predictions, using a satisfiability framework combined with gradient-based optimization. DeepSaDe is effective in a variety of ML tasks and provides a flexible representation of constraints, though sometimes at some cost of performance. It can enforce any constraint written in an SMT formula, as long as it is feasible to solve it with the solver. We rely on the Z3 solver but our framework is not dependent on it; any solver that can solve MaxSMT problems with universally quantified constraints can be used. We believe that evolving solver capabilities will allow DeepSaDe to handle more complex constraints. Extension of DeepSaDe to other architectures (e.g. Convolutional Neural Networks) is left for future work.

Acknowledgments

References

[1] Otto Jeremias Berg, Antti Juhani Hyttinen, Matti Juhani Järvisalo, et al., 'Applications of maxsat in data analysis', *Pragmatics of SAT*, (2019).

[2] Dimitris Bertsimas and Jack Dunn, 'Optimal classification trees', *Machine Learning*, **106**(7), 1039–1082, (2017).

[3] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda, 'A unified view of piecewise linear neural network verification', *Advances in Neural Information Processing Systems*, **31**, (2018).

[4] Mark Chavira and Adnan Darwiche, 'On probabilistic inference by weighted model counting', *Artificial Intelligence*, **172**(6-7), 772–799, (2008).

[5] James Cussens, 'Bayesian network learning by compiling to weighted max-sat', *arXiv preprint arXiv:1206.3244*, (2012).

[6] Emir Demirović, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J Stuckey, 'Murtree: Optimal decision trees via dynamic programming and search', *Journal of Machine Learning Research*, **23**(26), 1–47, (2022).

[7] Michelangelo Diligenti, Marco Gori, and Claudio Sacca, 'Semantic-based regularization for learning and inference', *Artificial Intelligence*, **244**, 143–165, (2017).

[8] Zhaohui Fu and Sharad Malik, 'On solving the partial max-sat problem', in *International Conference on Theory and Applications of Satisfiability Testing*, pp. 252–265. Springer, (2006).

[9] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning*, volume 1, MIT Press, 2016.

[10] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli, 'On the effectiveness of interval bound propagation for training verifiably robust models', *arXiv preprint arXiv:1810.12715*, (2018).

[11] Kshitij Goyal, Sebastijan Dumancic, and Hendrik Blockeel, 'Sade: Learning models that provably satisfy domain constraints', in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2022, Grenoble, France, September 19–23, 2022, Proceedings, Part V*, pp. 410–425. Springer, (2023).

[12] Oktay Günlük, Jayant Kalagnanam, Minhan Li, Matt Menickelly, and Katya Scheinberg, 'Optimal decision trees for categorical data via integer programming', *Journal of global optimization*, **81**(1), 233–260, (2021).

[13] Maya Gupta, Andrew Cotter, Jan Pfeifer, Konstantin Voevodski, Kevin Canini, Alexander Mangylov, Wojciech Moczydlowski, and Alexander Van Esbroeck, 'Monotonic calibrated interpolated look-up tables', *The Journal of Machine Learning Research*, **17**(1), 3790–3836, (2016).

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 'Delving deep into rectifiers: Surpassing human-level performance on imagenet classification', in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, (2015).

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, 'Deep residual learning for image recognition', in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, (2016).

[16] Geoffrey E Hinton and Sam Roweis, 'Stochastic neighbor embedding', *Advances in neural information processing systems*, **15**, (2002).

[17] Nick Hoernle, Rafael Michael Karampatsis, Vaishak Belle, and Kobi Gal, 'Multiplexnet: Towards fully satisfied logical constraints in neural networks', in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 5700–5709, (2022).

[18] Alexey Ignatiev, Edward Lam, Peter J Stuckey, and Joao Marques-Silva, 'A scalable two stage approach to computing optimal decision sets', in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3806–3814, (2021).

[19] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer, 'Reluplex: An efficient smt solver for verifying deep neural networks', in *International conference on computer aided verification*, pp. 97–117. Springer, (2017).

[20] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al., 'The marabou framework for verification and analysis of deep neural networks', in *International Conference on Computer Aided Verification*, pp. 443–452. Springer, (2019).

[21] Dmitry Malioutov and Kuldeep S Meel, 'Mlic: A maxsat-based framework for learning interpretable classification rules', in *CP*, (2018).

[22] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt, 'Deepproblog: Neural probabilistic logic programming', *Advances in Neural Information Processing Systems*, **31**, (2018).

[23] António Morgado, Federico Heras, Mark Liffiton, Jordi Planes, and Joao Marques-Silva, 'Iterative and core-guided maxsat solving: A survey and assessment', *Constraints*, **18**(4), 478–534, (2013).

[24] Leonardo de Moura and Nikolaj Bjørner, 'Z3: An efficient smt solver', in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, (2008).

[25] Nicholas Nethercote, Peter J Stuckey, Ralph Becket, Sebastian Brand, Gregory J Duck, and Guido Tack, 'Minizinc: Towards a standard cp modelling language', in *International Conference on Principles and Practice of Constraint Programming*, p. 534. Springer, (2007).

[26] Aishwarya Sivaraman, Golnoosh Farnadi, Todd Millstein, and Guy Van den Broeck, 'Counterexample-guided learning of monotonic neural networks', *Advances in Neural Information Processing Systems*, **33**, 11936–11948, (2020).

[27] Teruo Sunaga, 'Theory of interval algebra and its applications to numerical analysis', *RAAG Memoirs, 2-Misc. II*, 547–564, (1958).

[28] Sicco Verwer and Yingqian Zhang, 'Learning optimal classification trees using a binary linear program formulation', in *Proceedings of the AAAI conference on artificial intelligence*, pp. 1625–1632, (2019).

[29] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana, 'Formal security analysis of neural networks using symbolic intervals', in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1599–1614, (2018).

[30] Eric Wong and Zico Kolter, 'Provable defenses against adversarial examples via the convex outer adversarial polytope', in *International conference on machine learning*, pp. 5286–5295. PMLR, (2018).

[31] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck, 'A semantic loss function for deep learning with symbolic knowledge', in *International conference on machine learning*, pp. 5502–5511. PMLR, (2018).

[32] Seungil You, David Ding, Kevin Canini, Jan Pfeifer, and Maya Gupta, 'Deep lattice networks and partial monotonic functions', *Advances in neural information processing systems*, **30**, (2017).

[33] Jinqiang Yu, Alexey Ignatiev, Peter J Stuckey, and Pierre Le Bodic, 'Learning optimal decision sets and lists with sat', *Journal of Artificial Intelligence Research*, **72**, 1251–1279, (2021).

[34] Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh, 'Towards stable and efficient training of verifiably robust neural networks', *arXiv preprint arXiv:1906.06316*, (2019).

A Appendix

A.1 DeepSaDe: Domain Bound Propagation

This appendix provides more details on the calculation of the latent bounds in the domain-bound propagation approach in section 4.1. The calculation detailed below is based on the interval arithmetic approach ([27, 10]). Given the lower and upper bounds $l^{(n)}$ and $u^{(n)}$ for the input $\mathbf{x}^{(n)}$ of layer h_n, the output of the ith neuron of layer h_n for an arbitrary input $\mathbf{x}^{(n)}$ can be written as (assuming activation):

$$x_i^{(n+1)} = \text{activation}(b_i^{(n)} + \sum_{j=1}^{|h_{n-1}|} w_{j,i}^{(n)} x_j^{(n)})$$

Assuming that the activation function is monotonically non-decreasing, which the typically used activations (like ReLu, Sigmoid, Tanh) are, we can bound $x_i^{(n+1)}$ by lower and upper bounds $l_i^{(n+1)}$ and $u_i^{(n+1)}$ where:

$$l_i^{(n+1)} = \text{activation}(\min_{\mathbf{x}^{(n)}}(b_i^{(n)} + \sum_{j=1}^{|h_{n-1}|} w_{j,i}^{(n)} x_j^{(n)}))$$

$$u_i^{(n+1)} = \text{activation}(\max_{\mathbf{x}^{(n)}}(b_i^{(n)} + \sum_{j=1}^{|h_{n-1}|} w_{j,i}^{(n)} x_j^{(n)}))$$

In each formula, all terms can be maximized or minimized independently because the domain is an axis-parallel hyperrectangle. The bounds can be written as:

$$l_i^{(n+1)} = \text{activation}(b_i^{(n)} + \sum_{j=1}^{|h_{n-1}|} \min_{\mathbf{x}^{(n)}}(w_{j,i}^{(n)} x_j^{(n)}))$$

$$u_i^{(n+1)} = \text{activation}(b_i^{(n)} + \sum_{j=1}^{|h_{n-1}|} \max_{\mathbf{x}^{(n)}}(w_{j,i}^{(n)} x_j^{(n)}))$$

Note that for any linear function wx with $x \in [l, u]$ is maximal when $x = u$ for positive w, and when $x = l$ for negative w. Thus, the bounds can be calculated with the following expressions:

$$l_i^{(n+1)} = \text{activation}(b_i^{(n)} + \sum_{j:w_{j,i}^{(n)} \geq 0} w_{j,i}^{(n)} l_j^{(n)} + \sum_{j:w_{j,i}^{(n)} < 0} w_{j,i}^{(n)} u_j^{(n)})$$

$$u_i^{(n+1)} = \text{activation}(b_i^{(n)} + \sum_{j:w_{j,i}^{(n)} \geq 0} w_{j,i}^{(n)} u_j^{(n)} + \sum_{j:w_{j,i}^{(n)} < 0} w_{j,i}^{(n)} l_j^{(n)})$$

To conclude, with this approach, the bounds for the output of a layer h_n can be calculated in terms of the bounds of the input and the parameters of the layer. These bounds can thus be propagated from the input space to the latent space of the last layer. An illustration of this approach is provided in figure 3. It is important to note that the bounds calculated with this approach as very wide, but are suitable for our purposes and are cheap to compute. There are approaches to calculate more compact bounds ([29]) that may improve the efficiency of the model, this is left for future work.

A.2 Experimental Details

In this appendix, we provide details of the logical formulation of domain constraints for each use case in SMT, along with information regarding the network architecture and the hyper-parameters selected for each use case. We also provide an illustration of the Adversity
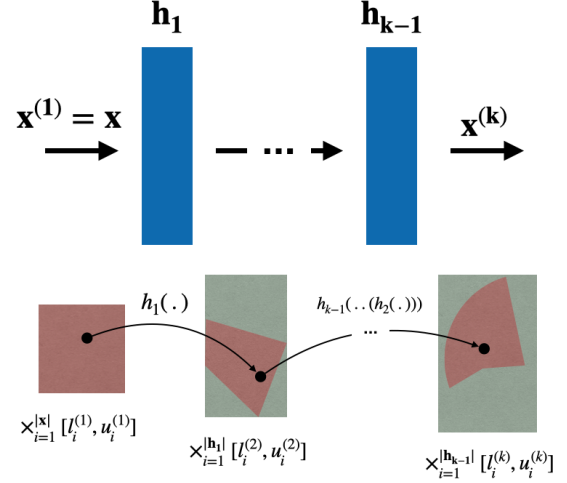

Figure 3: Domain Bound Propagation: the bounds for the output of each layer are constructed in such a way that any input to the network takes a value in the latent space within the constructed latent bounds. The input domain (in pink), when propagated through the layers, lies completely in the constructed latent domains (in green). Thus, enforcing a constraint on the latent bound of the last layer guarantees that the constraint is satisfied on any input within input bounds. In our approach, this construction is essential for achieving constraint satisfaction guarantees for neural networks.

Index (AdI) in figure 4. In the following, for a compact representation, we assume $\hat{y}$ represents the real-valued output of the model. Additionally, to express the component of a variable $\mathbf{x}$, we use subscript with the name of the component (e.g., $\mathbf{x}_{name}$ is the value of the 'name' feature of the variable $\mathbf{x}$). For each network, the ReLU activation is used in the hidden layers.

UC1: UC1 is the expense prediction problem with two domain constraints. The dataset contains 35317 violations of either of the constraints in 41417 instances. The 'household income' feature is a part of the domain constraints, it is mapped to the last layer with skip connections. Formulation of the domain constraint at the last layer of the DeepSaDe approach is:

$$\mathcal{K}' : \forall \mathbf{x} \in \bigtimes_{i=1}^{|h_{k-1}|} [l_i^{(k)}, u_i^{(k)}], \left((\text{SUM}(\hat{y}) \leqslant \mathbf{x}_{household-income}) \wedge \right.$$
$$\left. (\hat{y}_{going-out} \leqslant 0.05 * \mathbf{x}_{household-income})\right)$$

For the baseline REG, we use two regularization parameters λ and γ for each constraint with the weight on the predictive loss being $(1 - \lambda - \gamma)$:

$$\mathcal{L} = (1-\lambda-\gamma)*\text{MSE}+\lambda*\frac{1}{\|D\|}*\sum_{\mathbf{x} \in D} \max(0, \sum f_{\mathbf{w}}(\mathbf{x}) - \mathbf{x}_{income}) +$$
$$\gamma * \frac{1}{\|D\|} * \sum_{\mathbf{x} \in D} \max(0, f_{\mathbf{w}}(\mathbf{x})_{going\_out} - 0.05 * \mathbf{x}_{income})$$

Both λ and γ are tuned in the range of $[0, 0.49]$ using the 5-fold nested cross validation. For each model, a network with three hidden layers with sizes 50, 50, & 14 is considered. The learning rate for the baseline and DeepSaDe is selected to be 0.001 with a batch size of 5. For REG, the mode is trained for 10 epochs while for DeepSaDe the model is trained for 5 epochs. The maximal step size for DeepSaDe is
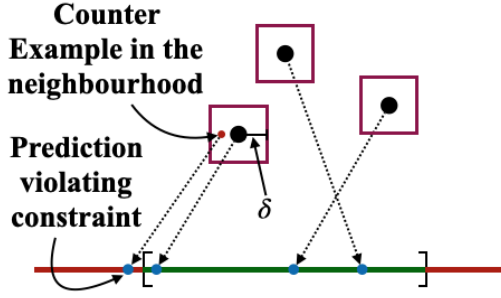
Figure 4: **Illustration of AdI:** 3 instances are mapped to real value predictions using a neural network. The green region is the output region that satisfies the constraint, the red region does not. It is possible, only for the leftmost data point, to construct a new point in the ℓ_∞ ball such that its prediction violates the constraint, even though the prediction of the original instance does not. Hence, AdI $= 1/3$.

selected to be 0.1 and the error threshold of $[0.1]$ is used for defining the soft constraints.

UC2: UC2 is the loan prediction task that is modeled by a binary classification task. The dataset contains 40 violations of the constraint in 492 instances. Features 'income' and 'credit history' are mapped to the last layer with identity mapping. Formulation of the domain constraint at the last layer of the DeepSaDe approach is:

$$\mathcal{K}' : \forall \mathbf{x} \in \overset{|h_{k-1}|}{\underset{i=1}{\times}} [l_i^{(k)}, u_i^{(k)}],$$
$$\Big((\mathbf{x}_{income} < 5000) \wedge (\mathbf{x}_{credit\_history} = 0)\Big) \implies$$
$$\Big(\hat{y}_{no-loan} > \hat{y}_{loan}\Big)$$

For each model, a network with three hidden layers with sizes 50, 30, & 10 is considered. For SL and SBR, the learning rate of 0.1, and batch size of 5 are selected and the models are trained for 400 epochs. For DeepSaDe, the learning rate of 0.1, maximal step size of 0.1, batch size of 5, and the margins (τ) of $[0, 1, 2]$ are selected, and the models are trained for 50 epochs.

UC3: UC3 is the music genre prediction task which is modeled by a multiclass classification task. The dataset contains 41 violations in 793 instances. Feature 'artist' is mapped to the last layer. Formulation of the domain constraint at the last layer of the DeepSaDe approach is:

$$\mathcal{K}' : \forall \mathbf{x} \in \overset{|h_{k-1}|}{\underset{i=1}{\times}} [l_i^{(k)}, u_i^{(k)}], (\mathbf{x}_{artist} = \text{the beatles}) \implies$$
$$\Big((\hat{y}_{classical} < 0) \wedge (\hat{y}_{electronic} < 0) \wedge$$
$$(\hat{y}_{metal} < 0) \wedge (\hat{y}_{pop} > 0 \vee \hat{y}_{rock} > 0)\Big)$$

For each model, a network with three hidden layers with sizes 50, 30, & 10 is considered. For SL and SBR, the learning rate of 0.1, and batch size of 5 are selected and the models are trained for 300 epochs. For DeepSaDe, the learning rate of 0.1, maximal step size of 0.1, batch size of 5, and the margins (τ) of $[0, 1, 2]$ are selected, and the models are trained for 50 epochs.

UC4: UC4 is the multi-label prediction task of classifying a sequence of four MNIST images to their respective classes. The dataset

contains no violations of the domain constraint. The domain constraint in this problem does not depend on any of the input features. Formulation of the domain constraint at the last layer of the DeepSaDe approach is:

$$\mathcal{K}' : \forall \mathbf{x} \in \overset{|h_{k-1}|}{\underset{i=1}{\times}} [l_i^{(k)}, u_i^{(k)}], \Big(\text{SUM}([\text{IF}(\hat{y}_i > 0, \text{i}, 0) \text{ for i in range}(10)]) > 10\Big)$$

For each model, a network with three hidden layers with sizes 50, 50, & 10 is considered. For FFN, the learning rate of 0.1, and batch size of 10 are selected and the models are trained for 40 epochs. For DeepSaDe, the learning rate of 0.1, maximal step size of 0.1, batch size of 10, and the margins (τ) of $[0, 1, 2]$ are selected, and the models are trained for 60 epochs.

UC5: UC5 is the preference learning task with the coherence condition as the domain constraint. The dataset contains no violations of the domain constraint. The domain constraint in this problem does not depend on any of the input features. The constraints are formulated in DeepSaDe using the following encoding:

$$\mathcal{K}' : \forall \mathbf{x} \in \overset{|h_{k-1}|}{\underset{i=1}{\times}} [l_i^{(k)}, u_i^{(k)}],$$
$$\Big(\text{PbEq}([\hat{y}_0 > 0, \hat{y}_1 > 0, \hat{y}_2 > 0, \hat{y}_3 > 0], 1) \wedge$$
$$\text{PbEq}([\hat{y}_4 > 0, \hat{y}_5 > 0, \hat{y}_6 > 0, \hat{y}_7 > 0], 1) \wedge$$
$$\text{PbEq}([\hat{y}_8 > 0, \hat{y}_9 > 0, \hat{y}_{10} > 0, \hat{y}_{11} > 0], 1) \wedge$$
$$\text{PbEq}([\hat{y}_{12} > 0, \hat{y}_{13} > 0, \hat{y}_{14} > 0, \hat{y}_{15} > 0], 1) \wedge$$
$$\text{PbEq}([\hat{y}_0 > 0, \hat{y}_4 > 0, \hat{y}_8 > 0, \hat{y}_{12} > 0], 1) \wedge$$
$$\text{PbEq}([\hat{y}_1 > 0, \hat{y}_5 > 0, \hat{y}_9 > 0, \hat{y}_{13} > 0], 1) \wedge$$
$$\text{PbEq}([\hat{y}_2 > 0, \hat{y}_6 > 0, \hat{y}_{10} > 0, \hat{y}_{14} > 0], 1) \wedge$$
$$\text{PbEq}([\hat{y}_3 > 0, \hat{y}_7 > 0, \hat{y}_{11} > 0, \hat{y}_{15} > 0], 1)\Big)$$

Where 'PbEq' encodes the exactly-k constraint, PbEq($[a, b, c], k$) enforces that exactly k of the boolean constraints a, b, and c must be true, where $k \in \{1, 2, 3\}$ in this example. For each model, a network with three hidden layers with sizes 25, 25, & 10 is considered. For SL and FFN, the learning rate of 0.1, and batch size of 10 are selected and the models are trained for 40 epochs. For DeepSaDe, the learning rate of 0.1, maximal step size of 0.1, batch size of 10, and the margins (τ) of $[0, 1, 2]$ are selected, and the models are trained for 40 epochs.

A.3 Results

In addition to the experiments presented in the main text, we also perform some supplementary experiments where we visualize the latent space for the networks learned with DeepSaDe. These experiments demonstrate that DeepSaDe, similar to existing gradient-based approaches, is also able to identify patterns in the latent space, which makes it easier to satisfy the constraint at the last layer. This experiment also adds validity to DeepSaDe as a learning approach.

Visualizing the Latent Space: First, we consider the multiclass classification problem of UC3 where the constraint is only enforced for a part of the input space that satisfies the input property (i.e., when the artist = 'the Beatles'). In Figure 5, we plot the t-SNE (t-distributed Stochastic Neighbour Embedding) ([16]) transformation of the latent space of the last layer. t-SNE captures the structure in the sense that neighboring points in the latent space should be closer in the t-SNE representation. In the left figure, data points that satisfy the input
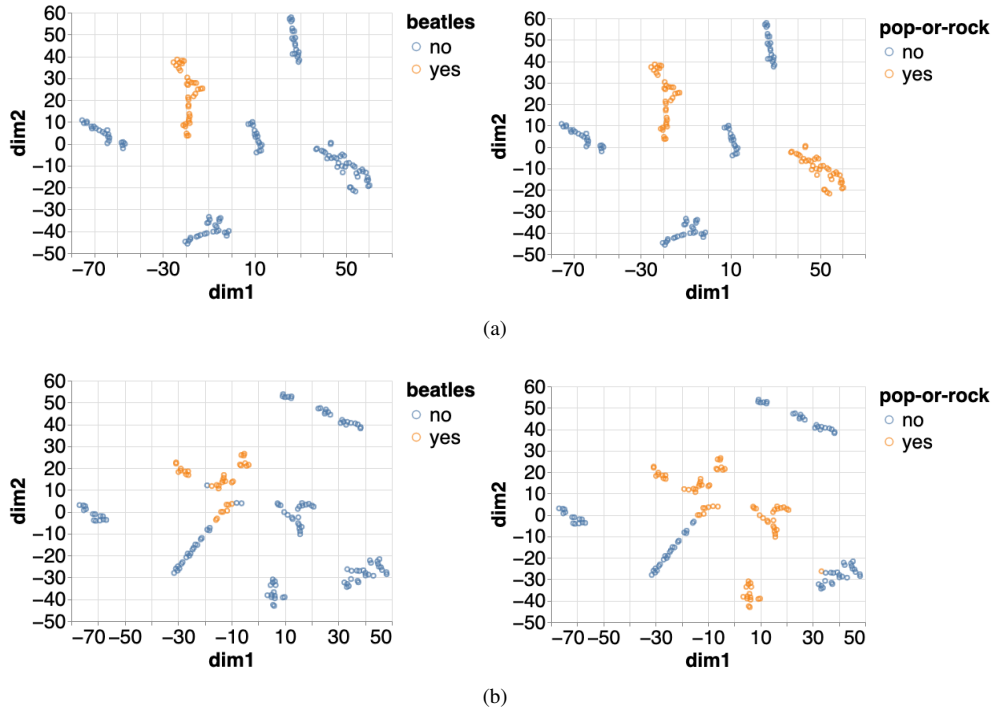
Figure 5: For UC3, a visualization of the latent space of the input to the last layer for the network trained by (a) DeepSaDe and (b) an unconstrained model, on the test data. A 2-dimensional reduction of the latent space using t-SNE is plotted on both plots. In the left plot, the points in orange are the ones that satisfy the input property (i.e., artist = 'the Beatles'). In the right plot, the orange points are the ones for which the network predicts the genre as either pop or rock.
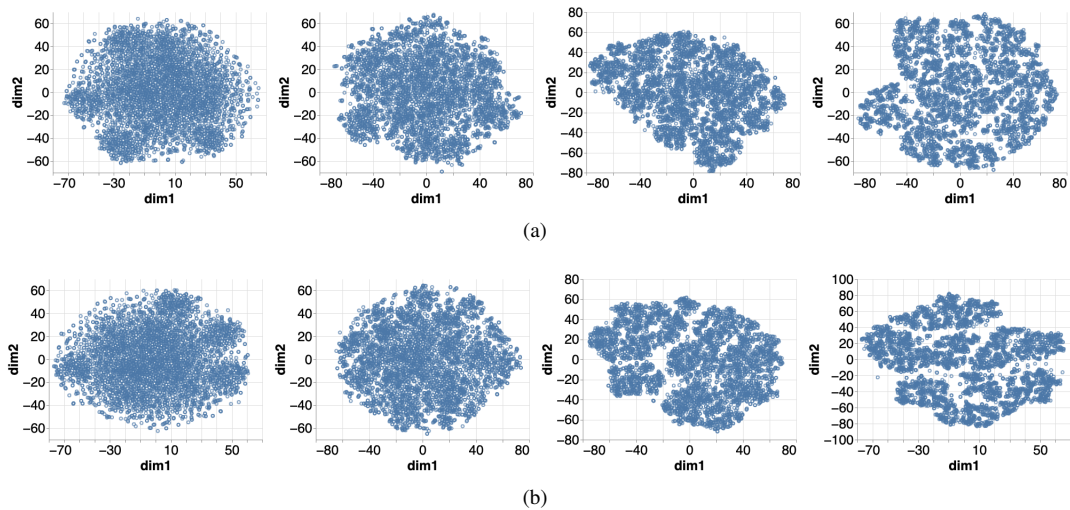

Figure 6: t-SNE representation of the latent space for (a.) FFN model, and (b.) DeepSaDe model for UC4. From left to right, the input space of the first to the last layer is visualized in a two-dimensional space for a test dataset. The latent vectors of the DeepSaDe model exhibit similar behavior as the FFN model, i.e., as the model gets deeper, similar instances can be clustered together by the t-SNE representation. This implies that similar to FFN, DeepSaDe is also able to identify patterns in the data while enforcing hard domain constraints which are always satisfied.

property are highlighted (in orange), while in the right figure, data points that are predicted as either 'pop' or 'rock' by the network are highlighted (in orange). It is shown that the instances that satisfy the input property are clustered together better for DeepSaDe than for the unconstrained model, which implies that enforcing the constraint at the last layer with DeepSaDe enables us to learn the latent space such that enforcing constraints at the last layer becomes easier. Furthermore,

the network trained through DeepSaDe satisfies the constraint for each of these instances by predicting the genre to be either pop' or 'rock,' as shown in the right figure. However, for an unconstrained network, the model still violates some of the constraints since the constraints are not enforced. We selected UC3 to analyze the distribution of the latent space in the context of the input property, which is difficult to do for use cases UC1, UC4, and UC5, where the constraint is being

enforced on all possible predictions.

We perform another analysis for use case UC4 which classifies a collection of MNIST images into respective labels in a multi-label classification task. Here, the constraint is being enforced for all possible outputs of the network, hence it is difficult to see the impact of constraint satisfaction in the t-SNE representation. However, in this experiment, we compare how the t-SNE representation of the Deep-SaDe model compares with the t-SNE representation for the baseline model (which in this case is a feed-forward neural network without any constraint enforcement). The goal here is to demonstrate the validity of DeepSaDe. Figure 6 shows that DeepSaDe exhibits similar behavior in the low dimensional representation of the latent space for each layer when compared with the similar representation of the FFN model. DeepSaDe can identify similar points in the latent space (because t-SNE can cluster data points as the model becomes deeper), which means that the network can learn patterns from the data, similar to the FFN while enforcing hard constraints on the predictions which are always satisfied.