A Solver-Free Framework for Scalable Learning in Neural ILP Architectures

Anonymous Author(s)
Affiliation
Address
email

Abstract

There is a recent focus on designing architectures that have an Integer Linear Programming (ILP) layer within a neural model (referred to as *Neural ILP* in this paper). Neural ILP architectures are suitable for pure reasoning tasks that require data-driven constraint learning or for tasks requiring both perception (neural) and reasoning (ILP). A recent SOTA approach for end-to-end training of Neural ILP explicitly defines gradients through the ILP black box [Paulus et al., 2021] – this trains extremely slowly, owing to a call to the underlying ILP solver for every training data point in a minibatch. In response, we present an alternative training strategy that is *solver-free*, i.e., does not call the ILP solver at all at training time. Neural ILP has a set of trainable hyperplanes (for cost and constraints in ILP), together representing a polyhedron. Our key idea is that the training loss should impose that the final polyhedron separates the positives (all constraints satisfied) from the negatives (at least one violated constraint or a suboptimal cost value), via a soft-margin formulation. While positive example(s) are provided as part of the training data, we devise novel techniques for generating negative samples. Our solution is flexible enough to handle equality as well as inequality constraints. Experiments on several problems, both perceptual as well as symbolic, which require learning the constraints of an ILP, show that our approach has superior performance and scales much better compared to purely neural baselines and other state-of-the-art models that require solver-based training. In particular, we are able to obtain excellent performance in 9×9 Visual Sudoku, to which the other Neural ILP solver is not able to scale.

1 Introduction

There has been a growing interest in the community which focuses on developing neural models for solving combinatorial optimization problems. These problems often require complex reasoning over discrete symbols. Many of these problems can be expressed in the form an underlying Integer Linear Program (ILP). Two different kinds of problem (input) settings have been considered in the literature: (a) purely symbolic and (b) combination of perceptual and symbolic. Solving an n-Queens problem given the partial assignment of queens on the board as input would be an example of the former, and solving a sudoku puzzle given the image of a partially filled board as input would be an example of latter. While the first setting corresponds to a pure reasoning task, second involves a combination of perception and reasoning tasks which need to be solved in a joint fashion. Existing literature proposes various approaches to handle one or both these settings. One line of work proposes purely neural models [Palm et al., 2018, Nandwani et al., 2021, Dong et al., 2019] for solving these tasks representing the underlying constraints and costs implicitly. While standard CNNs are used to solve the perceptual task, neural models such as Graph Neural Networks take care of the reasoning component. In an alternate view, one may want to solve these tasks by explicitly learning

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

the constraints and cost of the underlying ILP. While the perceptual reasoning would still be handled by using modules such as CNN, reasoning is taken care of by having an explicit representation in the form of an ILP layer representing constraints and costs. Such an approach would have the potential advantage of being more interpretable, and also being more accurate if the underlying constraints could be learned in a precise manner. Some recent approaches take this route, and include works by [Paulus et al., 2021, Pogancic et al., 2020, Berthet et al., 2020]. We refer to the latter set of approaches as *neural ILP* architectures. [1]

Learning in neural ILP architectures is complicated by the fact that there is a discrete optimization (in the form of an ILP layer) at the end of the network, which is typically non-differentiable, making the end-to-end learning of the system difficult. One of the possible ways is to instead use an iterative ILP solving algorithm such as a cutting-plane method [Gomory, 2010] that uses a continuous relaxation in each iteration which is shown to be differentiable due to the introduction of continuous variables [Ferber et al., 2020, Wilder et al., 2019]. Most of these works are concerned with learning only the cost function and assume that the constraints are given. Recent work by Paulus et al. [2021] has proposed an approach to directly pass the gradients through a black-box ILP solver. Specifically, they rely on Euclidean distances between the constraint hyperplanes and the current solution obtained by the ILP solver to approximate the gradients. Though this approach improves the quality of results compared to earlier works involving continuous relaxations, scalability gets severely affected since an ILP has to be solved at every learning iteration, making the training extremely slow.

We are interested in answering the following question: is there a way to train a neural ILP architecture in an end-to-end manner, which does not require an access to an underlying solver? Such an approach, if exists, could presumably result in significantly faster training times, resulting in scalability. In response, we propose a novel technique to differentiate through the learnable constraints as well as the learnable cost function of an unknown Integer Linear Program (ILP). During training, our technique doesn't solve the ILP to compute the gradients. Instead, we cast the learning of ILP constraints (and cost) as learning of a polyhedron, consisting of a set of hyperplanes, such that points inside the polyhedron are treated as positive, and points outside as negative. While a positive point needs to be classified as positive by each of the hyperplanes, a negative point needs to be classified as negative by at least one of the hyperplanes. Our formulation incorporates the learning of ILP cost also as learning of one of the hyperplanes in the system. We formulate a novel margin-based loss to learn these hyperplanes in a joint fashion. A covariance based regularizer that minimizes the cosine similarity between all pairs of hyperplanes ensures that learned constraints (hyperplanes) are not redundant. Since the training data comes only with positive examples, i.e., solutions to respective optimization problems, we develop several techniques for sampling the negatives, each of which is central to the effective learning of our hyperplanes in our formulation.

We present several experiments on problems which require learning of ILP constraints and cost, with both symbolic as well as perceptual input. These include solving a sudoku puzzle given the image of a partially filled board [Wang et al., 2019] (perceptual) and ILPs with random constraints (symbolic), Knapsack from sentence description (perceptual), and key-point matching (perceptual) from [Paulus et al., 2021]. Our closest competitor, CombOptNet [Paulus et al., 2021], can not solve even the smallest of the sudoku boards sized 4×4, whereas we can easily scale to 9×9, getting close to 100% accuracy. We are comparable on key-point matching, and obtain significantly better accuracy on random ILPs. We also outperform purely neural baselines (wherever applicable).

2 Related Works

In the first line of work, a neural model such as a Graph Neural Network (GNN) replaces a combinatorial solver altogether and *implicitly* encodes the rules or constraints in their weights, and learns them from data. Nandwani et al. [2022, 2021], Palm et al. [2018] train a Recurrent Relational Network for learning to solve puzzles like sudoku, graph coloring, futoshiki etc; Dong et al. [2019] propose Neural Logic Machines (NLMs) that learn lifted first order rules and experiment with blocks world, reasoning on family trees etc; Selsam et al. [2019], Amizadeh et al. [2019a,b] train a GNN for solving any CSP when constraints are *explicitly* given in a standard form such as CNF, DNF or Boolean Circuits. This is clearly different from us since we are interested in explicit learning the constraints of the underlying optimization problem.

[1] Although these methods can also train a Neural-ILP-Neural architecture, studying this is beyond our scope.

Inverse Optimization [Chan et al., 2021] aims to learn the constraints (cost) of a linear program (LP) from the observed optimal decisions. Recently [Tan et al., 2019, 2020] use the notion of Parameterized Linear Programs (PLP) in which both the cost and the constraints of an LP are parameterized by unknown weights and a feature vector. [Gould et al., 2019] show how to differentiate through continuous constrained optimization problem using the notion of a '*declarative node*' that uses implicit function theorem [Mingari Scarpello and Ritelli, 2002]. Similar to this are other works [Amos and Kolter, 2017, Agrawal et al., 2019] that define methods for differentiating through other continuous problems like convex quadratic programs (QPs) or cone programs. While all these techniques are concerned with learning the constraints (cost) for optimization problems defined over continuous variables, our focus is on learning constraints (cost) for an ILP which involves optimization over discrete variables and can be a significantly harder problem.

In the space of learning cost for an ILP, several approaches have been proposed recently. [Pogancic et al., 2020] define gradient w.r.t. linear cost function that is optimized over a given discrete space. [Rolínek et al., 2020a,b] exploit it for the task of Deep Graph Matching, and for differentiating through rank based metrics such as Average Precision/Recall respectively. [Berthet et al., 2020] replace the black-box discrete solver with its smooth perturbed version during training and exploit 'perturb-and-MAP' [Papandreou and Yuille, 2011] to compute the gradients. On similar lines, [Niepert et al., 2021] also exploit perturb-and-MAP but propose a different noise model for perturbation and also extends to the case when the output of the solver may feed into a downstream neural model. These methods assume the constraints to be given, and only learn the cost for an ILP.

Other methods use relaxations of specific algorithms for the task of learning the constraints and cost for an ILP. Ferber et al. [2020] backprop through the KKT conditions of the LP relaxation created iteratively while using cutting-plane method [Gomory, 2010] for solving a MILP; Wilder et al. [2019] add a quadratic penalty term to the continuous relaxation and use implicit function theorem to backprop through the KKT conditions, as done in Amos and Kolter [2017] for smooth programs; Mandi and Guns [2020] instead add a log-barrier term to get the LP relaxation and differentiate through its homogeneous self-dual formulation linking it to the iterative steps in the interior point method. Wang et al. [2019] uses a low rank SDP relaxation of MAXSAT and defines how the gradients flow during backpropagation. Presumably, these approaches are limited in their application since they rely on specific algorithmic relaxations.

Instead of working with a specific algorithm, some recent works differentiate through a black-box combinatorial solver for the task of learning constraints [Paulus et al., 2021]. The solver is called at every learning step to compute the gradient. This becomes a bottleneck during training, since the constraints are being learned on the go, and the problem could become ill-defined resulting in a larger solver time, severely limiting scalability of such approaches. In contrast, we would like to perform this task in a solver-free manner. [Pan et al., 2020] propose learning of constraints for the task of structured prediction. They represent the linear constraints compactly using a specific two layer Rectifier Network [Pan and Srikumar, 2016]. A significant limitation is that their method can only be used for learning constraints, and not costs. Further, their approach does not do well experimentally on the benchmark domains that we compare with. [Meng and Chang, 2021] propose a non-learning based approach for mining constraints in which for each training data $(\mathbf{c}, \mathbf{y})$, a new constraint is added which essentially imply that no other target can have better cost than the given target $\mathbf{y}$ for the corresponding cost $\mathbf{c}$. We are specifically interested in learning not only the constraints, but also the cost of an ILP from data. Convex Polytope Machines (CPM) [Kantchelian et al., 2014] learns a non linear binary classifier in the form of a polytope from a dataset of positive and negative examples. In contrast, our goal is to learn the cost as well as constraints for an ILP where both the constraints and the cost could be parameterized by another neural network. Also, we do not have access to any negative samples.

3 Differentiable ILP Loss

3.1 Background and Task Description

We are interested in learning how to solve combinatorial optimization problems that can be expressed as an Integer Linear Program (ILP) with equality as well as inequality constraints:

$$\arg\min_{\mathbf{z}} \mathbf{c}^T\mathbf{z} \;\text{ subject to }\; \mathbf{U}\mathbf{z} = \mathbf{v};\; \mathbf{G}\mathbf{z} + \mathbf{h} \geq \mathbf{0};\;\; \mathbf{z} \in \mathbb{Z}^n \tag{1}$$

Here $\mathbf{c} \in \mathbb{R}^n$ represents an n dimensional cost vector, the matrix $\mathbf{U} \in \mathbb{R}^{m_1 \times n}$ and $\mathbf{v} \in \mathbb{R}^{m_1}$ represent m_1 linear equality constraints, and the matrix $\mathbf{G} \in \mathbb{R}^{m_2 \times n}$ and $\mathbf{h} \in \mathbb{R}^{m_2}$ represent m_2 linear inequality constraints, together defining the feasible region.

Without loss of generality, one can replace an equality constraint $\mathbf{u}_i^T \mathbf{z} = \mathbf{v}_i$ by two inequality constraints: $\mathbf{u}_i^T \mathbf{z} \geq \mathbf{v}_i$ and $-\mathbf{u}_i^T \mathbf{z} \geq -\mathbf{v}_i$. Here $\mathbf{u}_i$ and $\mathbf{v}_i$ represent the ith row of the matrix $\mathbf{U}$ and i^{th} element of the vector $\mathbf{v}$ respectively. Using this, one can reduce ILP in eq. (1) to an equivalent form with just inequality constraints:

$$\arg\min_{\mathbf{z}} \mathbf{c}^T \mathbf{z} \;\text{ subject to }\; \mathbf{A}\mathbf{z} + \mathbf{b} \geq \mathbf{0}, \mathbf{z} \in \mathbb{Z}^n \tag{2}$$

Here matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is row-wise concatenation of the matrices $\mathbf{G}, \mathbf{U}$ and $-\mathbf{U}$. Vector $\mathbf{b} \in \mathbb{R}^m$ is a row-wise concatenation of vectors $\mathbf{h}, -\mathbf{v}$ and $\mathbf{v}$, and $m = 2m_1 + m_2$ is the total number of inequality constraints. We represent the ith constraint $\mathbf{a}_i^T \mathbf{z} + \mathbf{b}_i \geq 0$, as $[\mathbf{a}_i | \mathbf{b}_i]$. The integer constraints $\mathbf{z} \in \mathbb{Z}^n$ make the problem NP hard.

Neural ILP Architecture: In a neural ILP architecture, the constraint matrix $\mathbf{A}$, vector $\mathbf{b}$, and the cost vector $\mathbf{c}$ are neural functions $f_{\mathbf{A}}, f_{\mathbf{b}}$, and $f_{\mathbf{c}}$ (of input x) parameterized by learnable $\Theta = (\theta_{\mathbf{A}}, \theta_{\mathbf{b}}, \theta_{\mathbf{c}})$, *i.e.*, $\mathbf{A} = f_{\mathbf{A}}(x; \theta_{\mathbf{A}})$; $\mathbf{b} = f_{\mathbf{b}}(x; \theta_{\mathbf{b}})$; $\mathbf{c} = f_{\mathbf{c}}(x; \theta_{\mathbf{c}})$. This results in a different ILP for each input x. For an input x, the solution given by a neural ILP model, $\mathbf{M}_\Theta(x)$, is nothing but the optima of the following paramterized ILP where $\mathbf{A}, \mathbf{b}, \mathbf{c}$ are replaced by corresponding neural functions $f_{\mathbf{A}}, f_{\mathbf{b}}, f_{\mathbf{c}}$ evaluated at input x:

$$\mathbf{M}_\Theta(x) = \arg\min_{\mathbf{z}} \left(f_{\mathbf{c}}(x; \theta_{\mathbf{c}})\right)^T \mathbf{z} \;\text{ subject to }\; f_{\mathbf{A}}(x; \theta_{\mathbf{A}})\mathbf{z} + f_{\mathbf{b}}(x; \theta_{\mathbf{b}}) \geq \mathbf{0}, \mathbf{z} \in \mathbb{Z}^n \tag{3}$$

Example of visual-sudoku: In $k \times k$ visual-sudoku, the input x is an image of a sudoku puzzle and $\mathbf{y}^* \in \{0, 1\}^n$ is the corresponding solution, represented as a $n = k^3$ dimensional binary vector: the integer in each of the k^2 cells is represented by a k dimensional one-hot binary vector. Function $f_{\mathbf{c}}(x; \theta_{\mathbf{c}})$ parameterizing the cost is nothing but a neural digit classifier that classifies the content of each of the k^2 cells into one of the k classes. The neural functions $f_{\mathbf{A}}$ and $f_{\mathbf{b}}$ are independent of the input x as the constraints are the same for every $k \times k$ sudoku puzzle. Therefore, $f_{\mathbf{A}}(x; \theta_{\mathbf{A}}) = \theta_{\mathbf{A}}$, and $f_{\mathbf{b}}(x; \theta_{\mathbf{b}}) = \theta_{\mathbf{b}}$, where $\theta_{\mathbf{A}}$ is just a learnable matrix of dimension $m \times n$ and $\theta_{\mathbf{b}}$ is a learnable vector of dimension m. See Bartlett et al. [2008] for an ILP formulation of $k \times k$ sudoku with $4k^2$ equality constraints in a $n = k^3$ dimensional binary space $\{0, 1\}^n$.

Learning Task: Given a training dataset $\mathcal{D} = \{(x_s, \mathbf{y}_s^*) \mid s \in \{1 \ldots S\}\}$ with S training samples, the task is to learn the parameters $\Theta = (\theta_{\mathbf{A}}, \theta_{\mathbf{b}}, \theta_{\mathbf{c}})$, such that $\mathbf{y}_s^* = \mathbf{M}_\Theta(x_s)$ for each s. To do so, one needs to define derivatives $\frac{\partial L}{\partial \mathbf{A}}, \frac{\partial L}{\partial \mathbf{b}}$, and $\frac{\partial L}{\partial \mathbf{c}}$ *w.r.t.* $\mathbf{A}, \mathbf{b}$, and $\mathbf{c}$ respectively, of an appropriate loss function L. Once such a derivative is defined, one can easily compute derivatives *w.r.t.*$\theta_{\mathbf{A}}, \theta_{\mathbf{b}}$, and $\theta_{\mathbf{c}}$ using the chain rule: $\frac{\partial L}{\partial \theta_{\mathbf{A}}} = \frac{\partial L}{\partial \mathbf{A}} \frac{\partial f_{\mathbf{A}}(x; \theta_{\mathbf{A}})}{\partial \theta_{\mathbf{A}}}$; $\frac{\partial L}{\partial \theta_{\mathbf{b}}} = \frac{\partial L}{\partial \mathbf{b}} \frac{\partial f_{\mathbf{b}}(x; \theta_{\mathbf{b}})}{\partial \theta_{\mathbf{b}}}$; and $\frac{\partial L}{\partial \theta_{\mathbf{c}}} = \frac{\partial L}{\partial \mathbf{c}} \frac{\partial f_{\mathbf{A}}(x; \theta_{\mathbf{c}})}{\partial \theta_{\mathbf{c}}}$. Hence, in the formulation below, we only worry about computing gradients *w.r.t.* the constraint matrix $\mathbf{A}$, vector $\mathbf{b}$, and the cost vector $\mathbf{c}$.

The existing approaches, *e.g.*, [Paulus et al., 2021], explicitly need access to current model prediction $\mathbf{M}_\Theta(x)$ This requires solving the ILP in eq. (3), making the learning process extremely slow. In contrast, we present a *'solver-free'* framework for computation of an appropriate loss and its derivatives *w.r.t.* $\mathbf{A}, \mathbf{b}$, and $\mathbf{c}$. Our framework does not require solving any ILP while training, thereby making it extremely scalable as compared to existing approaches.

3.2 A Solver-free Framework

Conversion to a constraint satisfaction problem: As a first step, we convert the constraint optimization problem in eq. (2) to an equivalent constraint satisfaction problem by introducing an additional '*cost–constraint*': $\mathbf{c}^T \mathbf{z} \leq \mathbf{c}^T \mathbf{y}^*$, equivalent to $\mathbf{a}_{m+1} = -\mathbf{c}$ and $\mathbf{b}_{m+1} = \mathbf{c}^T \mathbf{y}^*$.
Note that the above cost–constraint separates the solution $\mathbf{y}^*$ of the original ILP in eq. (2) from the other feasible integral points. This is because $\mathbf{y}^*$ must achieve the minimum objective value $\mathbf{c}^T \mathbf{y}^*$ amongst all the feasible integral points and hence no other feasible integral point $\mathbf{z}$ can obtain objective value less than $\mathbf{c}^T \mathbf{y}^*$. The new constraint $[\mathbf{a}_{m+1} | \mathbf{b}_{m+1}]$ together with the original m constraints guarantee that $\mathbf{y}^*$ is the only solution satisfying all of the $(m+1)$ constraints. This results

in the following equivalent linear constraint satisfaction problem:

$$\arg\min_{\mathbf{z}} \mathbf{0}^T\mathbf{z} \;\text{ subject to }\; \mathbf{A}\mathbf{z}+\mathbf{b} \geq \mathbf{0} \;\;;\;\; \mathbf{c}^T\mathbf{z} \leq \mathbf{c}^T\mathbf{y}^* \;\;;\;\; \mathbf{z} \in \mathbb{Z}^n \tag{4}$$

Constructing such an equivalent satisfaction problem requires access to the solution $\mathbf{y}^*$ of the original ILP in eq. (2) and that is already available to us for the training data samples. By rolling up the cost vector $\mathbf{c}$ into an additional constraint with the help of $\mathbf{y}^*$, we have converted our original objective of learning both the cost and constraints to just learning of constraints in eq. (4).

The main intuition behind our framework comes from the observation that each of the m linear constraints defining the feasible region are essentially m linear binary classifiers (hyperplanes) in $\mathbb{R}^n$ separating the ground truth $\mathbf{y}^*$ from the infeasible region. The additional cost–constraint in eq. (4) separates $\mathbf{y}^*$ from other integral samples feasible *w.r.t.* the original m constraints. Learning constraints of an ILP is akin to simultaneously learning m linear binary classifiers separating $\mathbf{y}^*$ from other infeasible points of the original ILP along with learning a classifier for cost–constraint, separating $\mathbf{y}^*$ for other feasible integral points.

For a vector $\mathbf{y}^*$ to lie inside the feasible region, *all* the classifiers need to classify it positively, and hence it acts as a positive data point for all the $m+1$ binary classifiers. In the absence of explicitly provided negative samples, we propose a couple of strategies for sampling them from $\mathbb{Z}^n$ for each ground truth $\mathbf{y}^*_s$. We discuss them in detail in section 3.3. For now, let $\mathcal{N}_{\mathbf{y}^*}$ be the set of all the negative samples generated by all our sampling techniques for a ground truth positive sample $\mathbf{y}^*$. In contrast to a positive point which needs to satisfy <u>*all*</u> the $m+1$ constraints, a negative point becomes infeasible even if it violates <u>*any one*</u> of the $m+1$ constraints. As a result, it would suffice if any one of the $m+1$ classifiers correctly classify it as a negative sample (refer to appendix for an illustration). While learning, one cannot assign a negative sample to any specific classifier as their parameters are being updated continuously. Instead, we make a soft assignment depending upon the Euclidean distance of the negative sample from the hyperplane. With this intuition, we now formally define our ILP–Loss.

Formulating solver-free ILP–Loss: Let $d(\mathbf{z};[\mathbf{a}_i|\mathbf{b}_i]) = \frac{\mathbf{a}_i^T\mathbf{z}+\mathbf{b}_i}{|\mathbf{a}_i|}$ represent the signed Euclidean distance of a point $\mathbf{z} \in \mathbb{R}^n$ from the hyperplane corresponding to the constraint $[\mathbf{a}_i|\mathbf{b}_i]$. We want the signed distance from all the hyperplanes to be positive for the ground truth samples and negative from at least one hyperplane for all the sampled negative points. We operationalize this via a margin based loss function:

$$L(\mathbf{A},\mathbf{b},\mathbf{c},\mathbf{y}^*|\mathcal{N}_{\mathbf{y}^*}) \;=\; \lambda_{pos}L_+ + \lambda_{neg}L_- + \lambda_{cov}L_o \quad \text{where} \tag{5}$$

$$L_+ \;=\; \frac{1}{m}\sum_{i=1}^{m} \max\{0, \mu^+ - d(\mathbf{y}^*;[\mathbf{a}_i|\mathbf{b}_i])\} \tag{6}$$

$$L_- \;=\; \frac{1}{|\mathcal{N}_{\mathbf{y}^*}|}\sum_{\mathbf{y}^- \in \mathcal{N}_{\mathbf{y}^*}} \sum_{i=1}^{m+1} w^i_{\mathbf{y}^-} \max\{0, \mu^- + d(\mathbf{y}^-;[\mathbf{a}_i|\mathbf{b}_i])\} \tag{7}$$

$$L_o \;=\; \sum_{i,j=1;i\neq j}^{m} \frac{\mathbf{a}_i^T\mathbf{a}_j}{|\mathbf{a}_i||\mathbf{a}_j|} \;\simeq\; \left(\sum_{i=1}^{m}\frac{\mathbf{a}_i}{|\mathbf{a}_i|}\right)^2 \tag{8}$$

$$w^i_{\mathbf{y}^-} \;=\; \frac{e^{\left(-d\left(\mathbf{y}^-;[\mathbf{a}_i|\mathbf{b}_i]\right)/\tau\right)}}{\sum_{j=1}^{m+1} e^{(-d(\mathbf{y}^-;[\mathbf{a}_j|\mathbf{b}_j])/\tau)}} \tag{9}$$

We call L_+, L_-, and L_o as the positive, negative and covariance loss respectively. The average in L_+ ensures that a ground truth sample is positively classified by <u>*all*</u> the classifiers. μ^+ and μ^- are the hyperparameters representing the margins for the positive and the negative points respectively. $w^i_{\mathbf{y}^-}$ in L_- represents the soft assignment of $\mathbf{y}^-$ to the ith constraint $[\mathbf{a}_i|\mathbf{b}_i]$ and is computed by temperature annealed softmax over the negative distances in eq. (9). Softmax ensures that the hyperplane which is most confident to classify it as negative gets the maximum weight. When $\mathbf{y}^-$ lies inside the feasible region, then the most confident classifier is the one closest to $\mathbf{y}^-$. To avoid the pathological behaviour of decreasing the loss by changing the weights, we ensure that gradients do not flow through $w^i_{\mathbf{y}^-}$ in eq. (7).

The temperature parameter τ needs to be annealed as the training progresses. A high temperature initially can be seen as '*exploration*' for the right constraint that will be violated by $\mathbf{y}^-$. This is important as the constraints are also being learnt and are almost random initially, so a given negative $\mathbf{y}^-$ should not commit to a particular hyperplane. Additionally, this encourages multiple constraints to be violated for each negative, which leads to a robust set of constraints. As the training progresses, we reduce the temperature τ, ensuring that the most confident classifier with least signed distance gets almost all the weight. If $\mathbf{y}^-$ is correctly classified as a negative with a margin μ^- by *any* classifier *i.e.*, $d(\mathbf{y}^-; [\mathbf{a}_i|\mathbf{b}_i]) \leq -\mu^-$ for some i, then the corresponding negative loss, $\max\{0, \mu^- + d(\mathbf{y}^-; [\mathbf{a}_i|\mathbf{b}_i])\}$, becomes zero, and a low value of τ ensures that it gets all the weight.

The last term, L_o acts as a regularizer and tries to ensure that no two learnt constraints are similar. This is done by minimizing the cosine similarity between all the pairs of constraints. The weights λ_{pos}, λ_{neg}, and λ_{cov} are computed automatically with a multi-loss weighing technique using coefficient of variations as described in Groenendijk et al. [2021]. Intuitively, the loss term with maximum variance over the learning iterations, gets most of the weight.

Other details: See appendix for specific parameterization and initialization of $\mathbf{A}$, $\mathbf{b}$; handling known boundaries; and encoding equality constraints.

3.3 Negative Sampling

A meaningful computation of ILP–Loss in eq. (5) depends crucially on the negative samples $\mathcal{N}^*_{\mathbf{y}}$. Randomly sampling negatives from $\mathbb{Z}^n$ is one plausible strategy, but it may not be efficient from the learning perspective: any point which is far away from any of the classifiers will easily be classified as negative and will not contribute much to the loss function. In response, we propose multiple alternatives for sampling the negative points:

1. **Integral k-hop neighbours:** To generate negatives, we sample the integral neighbours that are at a Hamming Distance of k from $\mathbf{y}^*$. For small k, these form the hardest negatives as they are closest to the positive point. Note that it is possible for a few integral neighbours to be feasible *w.r.t.* the m constraints, but they must have worse cost than the given ground truth $\mathbf{y}^*$. Such samples contribute towards learning the cost parameters $\theta_{\mathbf{c}}$.

2. **Project and sample:** In this method, we project the ground truth $\mathbf{y}^*$ on each of the m hyperplanes and then randomly sample a neighbour of the projection, generating a total of m negatives. Sampling probability in the jth dimension depends on the jth coordinate of the projection: if value of the jth coordinate is $r \notin \mathbb{Z}$, then we sample $floor(r)$ and $ceil(r)$ with probability $ceil(r) - r$ and $r - floor(r)$ respectively. If $r \in \mathbb{Z}$, then we sample r with probability 1. Projection samples are close to the boundary of the currently learnt polyhedron, thus taking the training progress into account. Further, each hyperplane is likely to be assigned to a close-by negative due to projection sampling.

3. **Batch negatives:** In this strategy, we consider every other ground truth $\mathbf{y}^*_{s'}$ $s' \neq s$ in the minibatch as a potential negative sample for $\mathbf{y}^*_s$. We call it as a 'batch–negative'. This is particularly useful for learning the cost parameters $\theta_{\mathbf{c}}$ when the learnable constraints are the same for all the ground truth samples, such as in sudoku. In such cases, a batch–negative $\mathbf{y}^*_{s'}$ being a feasible point of the original ILP formulation, must always satisfy all of the m learnable constraints of the original ILP in eq. (2). Hence, the only way for $\mathbf{y}^*_{s'}$ to be correctly classified as a negative for $\mathbf{y}^*_s$ is by violating the cost constraint in eq. (4), Learning cost parameters $\theta_{\mathbf{c}}$ that result in violation of the cost constraint for every batch–negative helps in ensuring that the ground truth $\mathbf{y}^*_s$ indeed has the minimum cost.

4 Experiments

The goal of our experiments is to evaluate the effectiveness and scalability of our proposed approach compared to the SOTA black-box solver based approach, i.e., CombOptNet [Paulus et al., 2021]. We experiment on 4 datasets: visual sudoku [Wang et al., 2019], and three datasets from [Paulus et al., 2021]: random constraints, knapsack from sentence description and key-point matching. We also compare with an appropriately designed neural baseline for each of our datasets. To measure scalability, in each of the domains, we compare performance of each of the algorithms in terms of time taken, as well as accuracy obtained, for varying problem sizes. For each of the algorithms, we kept a maximum time limit of 12 hours for training. We next describe details of each of these datasets, appropriate baselines, and our results. See the appendix for the details of the ILP solver used in our experiments, the hardware specifications, the hyper-parameters, and various other design choices.

4.1 Visual Sudoku

This task involves learning the rules of the sudoku puzzle. The input x is an image of a sudoku puzzle where a digit is replaced by a random MNIST image of the same class. A $k \times k$ sudoku puzzle can be viewed as an ILP with $4k^2$ equality constraints over k^3 binary variables [Bartlett et al., 2008]. Each of the digit images in the k^2 cells are decoded into a k dimensional real vector, resulting in a k^3 dimensional cost $\mathbf{c} \in \mathbb{R}^n$ where $n = k^3$. The k^3 dimensional binary output vector (solution) $\mathbf{y}^*$ is created by representing each of the k^2 cells of the puzzle by a k dimensional binary vector which takes a value of 1 at i^{th} dimension if and only if the cell is filled by digit i. Here, the cost vector needs to be inferred from the input image. Our objective is to learn the constraint matrix $\mathbf{A} = f_{\mathbf{A}}(x; \theta_{\mathbf{A}}) = \theta_{\mathbf{A}}$ and vector $\mathbf{b} = f_{\mathbf{b}}(x; \theta_{\mathbf{b}}) = \theta_{\mathbf{b}}$, representing the linear constraints of sudoku. The cost vector $\mathbf{c} = f_{\mathbf{c}}(x; \theta_{\mathbf{c}})$ is a function of the input image x, parameterized by $\theta_{\mathbf{c}}$ which needs to be learned.

Dataset: We experiment with 3 different datasets for $k = 4, 6$, and 9. We first build symbolic sudoku puzzles with k digits, and use MNIST to convert a puzzle into an image. For 9×9 sudokus, we use a standard dataset from Kaggle [Park, 2017], for $k = 4$ we use publically available data from [Arcot and Kalluraya, 2019], and for $k = 6$ we use the data generation process described in [Nandwani et al., 2022]. We randomly select $10,000$ samples for training, 1000 samples for validation, and 500 samples for testing for each k. To generate the input images for visual-sudoku, we use the official train and test split of MNIST[Deng, 2012]. Digits in our train and test splits are randomly replaced by images in the MNIST train and test sets respectively, avoiding any leakage.

Baselines: Our neural baseline is Recurrent Relational Networks (RRN) [Palm et al., 2018]: a purely neural approach for reasoning based on Graph Neural Networks. We use the default set of parameters provided in their paper for training RRNs. We note that the RRN baseline uses additional information in the form of graph structure which is not available to our solver. We make this comparison to see how well we perform compared to one of the SOTA techniques for this problem. We also compare against another neuro-symbolic architecture that uses SATNet [Wang et al., 2019] as the reasoning layer. The neural component in all the neuro-symbolic methods is a CNN which decodes images into digits.

Results: Table 1 compares the performance and training time of our method against the different baselines described above. CompOptNet fails miserably on this problem, not being able to complete training for any of the sizes in the stipulated time. The purely neural model gives a decent performance and is competitive with ours for smaller board sizes[2]. But for larger board size of 9×9, we beat RRN based model by a significant margin of about 25 points. While SATNet performs comparable on smaller 4×4 sudoku puzzles, its performance degrades drastically to 17.8% for 9×9 board size. See appendix for a comparison on the dataset used in Wang et al. [2019].

Table 1: Board accuracy and training time for different board sizes of visual sudoku (TO refers to time-out of 12 hours)

	Board Accuracy (In %)			**Training Time (in min.)**		
	4 x 4	**6 x 6**	**9 x 9**	**4 x 4**	**6 x 6**	**9 x 9**
Neural (RRN)	99.4	**97.6**	74.4	120	65	97
CombOptNet	0.0	0.0	0.0	TO	TO	TO
SATNet	98.0	80.8	17.8	79	89	205
ILP–Loss (Ours)	**99.8**	94.7	**99.0**	**2**	**5**	**45**

4.2 Random constraints

This is the synthetic dataset borrowed from Paulus et al.. The training data $\mathcal{D} = \{(\mathbf{c}_s, \mathbf{y}^*_s) | s \in \{1 \cdots S\}\}$ is created by first generating a random polyhedron $\mathcal{P} \subseteq \mathcal{Y}$ by sampling m' hyperplanes $\{[\mathbf{a}'_i | \mathbf{b}'_i] \mid \mathbf{a}'_i, \mathbf{b}'_i \in \mathbb{R}^n, i \in \{1 \cdots m'\}\}$ in an n dimensional bounded continuous space $\mathcal{Y}$. A cost vector $\mathbf{c}_s \in \mathbb{R}^n$ is then randomly sampled and corresponding $\mathbf{y}^*_s$ is obtained as $\mathbf{y}^*_s = \arg\min_{\mathbf{z} \in \mathcal{P}} \mathbf{c}_s^T \mathbf{z}, \mathbf{z} \in \mathbb{Z}^n$. Objective is to learn a constraint matrix $\mathbf{A}$ and vector $\mathbf{b}$ such that for a ground truth $(\mathbf{c}, \mathbf{y}^*)$ pair, $\mathbf{y}^* = \arg\min_{\mathbf{z} \in \mathcal{Y}} \mathbf{c}^T \mathbf{z}$, $\mathbf{z} \in \mathbb{Z}^n$, and $\mathbf{A}\mathbf{z} + \mathbf{b} \geq \mathbf{0}$.

Dataset: Restricting $\mathcal{Y}$ to $[0, 1]^n$ and $[-5, 5]^n$ results in two variations, referred as 'binary' and 'dense' settings, respectively. For both of the output spaces, we experiment with four different settings in $n = 16$ dimensional space by varying the number of ground truth constraints as $m' = 1, 2, 4$, and

[2]For 6×6 sudokus, 4.4% of queries timed out during inference using our learned constraints.

Table 2: Mean $\pm$ std err of the vector accuracy ($\mathbf{M}_\Theta(x) = \mathbf{y}^*$) and training time over the 10 problem instances in each setting. Number of learnable constraints is twice the number of ground truth constraints.

	Binary				**Dense**			
	1	**2**	**4**	**8**	**1**	**2**	**4**	**8**
	Vector Accuracy (mean $\pm$ std err in %)							
CombOptNet	97.0 ± 1.0	96.4 ± 0.3	89.5 ± 1.6	61.2 ± 3.9	85.9 ± 1.0	71.2 ± 2.3	29.6 ± 2.8	2.7 ± 1.0
ILP–Loss (Ours)	**97.8** $\pm$ **0.4**	**96.4** $\pm$ **0.5**	**93.5** $\pm$ **0.8**	**88.6** $\pm$ **3.6**	**96.6** $\pm$ **0.3**	**86.3** $\pm$ **2.3**	**74.0** $\pm$ **5.4**	**41.5** $\pm$ **5.7**
	Training Time (mean $\pm$ std err in minutes)							
CombOptNet	44.5 ± 0.5	43.4 ± 0.5	48.7 ± 1.1	56.5 ± 3.2	38.6 ± 1.5	40.9 ± 0.1	39.6 ± 0.7	61.4 ± 1.6
ILP–Loss (Ours)	**6.9** $\pm$ **2.3**	**15.4** $\pm$ **2.3**	**28.9** $\pm$ **4.8**	**42.0** $\pm$ **7.2**	**12.9** $\pm$ **2.0**	**27.2** $\pm$ **3.7**	**30.3** $\pm$ **4.4**	**35.2** $\pm$ **7.6**

8. For each m', we experiment with all the 10 datasets provided by Paulus et al. and report mean over the 10 models. The training data in each case consists of 1600 $(\mathbf{c}, \mathbf{y})$ pairs and model performance is tested for 1000 cost vectors.

Baseline: Here we compare only against CombOptNet. A neural baseline (an MLP) is shown to perform badly for this symbolic problem in Paulus et al.. Hence we exclude this in our experiments.

Results: Table 2 presents the comparison of the two algorithms in terms of accuracy as well as time taken for training. While we perform marginally better than CombOptNet in terms of training time, our performance is significantly better, on almost all problems in the dense setting, and the larger problems in the binary setting. We are more than 35 accuracy points better compared to CombOptNet on the largest problem instance with 8 constraints in both settings. This result demonstrates that our approach is not only faster in terms of training time, but also results in better solutions even when the baseline converges to a solution. This validates the effectiveness of our framework.

4.3 Knapsack from Sentence Descriptions

This task, also borrowed from Paulus et al., is based on the classical NP-hard Knapsack problem. Each input consists of N sentences describing the price and weight of N items and the objective is to select a subset of items that maximizes their total price while keeping their total weight less than the knapsack's capacity C: $\arg\max \mathbf{p}^T\mathbf{z}$ s.t. $\mathbf{w}^T\mathbf{z} \leq C, \mathbf{z} \in \{0,1\}^N$, where $\mathbf{p}, \mathbf{w} \in \mathbb{R}^N$ are the price and weight vectors, respectively. Each sentence has been converted into a $d = 4096$ dimensional dense embedding using Conneau et al. [2017], so that each input is $N \times d$ dimensional vector $\mathbf{x}$. The corresponding output is $\mathbf{y}^* \in \{0,1\}^N$. The knapsack capacity C is fixed for all instances in a dataset. The task is to learn the parameters $\theta_\mathbf{c}$, $\theta_\mathbf{A}$ and $\theta_\mathbf{b}$ of a neural network that extracts the cost and the constraints from $\mathbf{x}$. Note that here both the cost and the constraints need to be inferred from the input.

Table 3: Accuracy and train-time for various knapsack datasets with different number of items in each instance. For $N = 25, 30$, TO represents timeout of 12 hours and we evaluate using the latest snapshot of the model obtained within 12 hours of total training.

	Accuracy (in %)					**Training Time (in mins.)**				
	10	**15**	**20**	**25**	**30**	**10**	**15**	**20**	**25**	**30**
CombOptNet	64.8	46.0	27.8	4.2	0.0	46.3	91.4	156.2	TO	TO
ILP–Loss (Ours)	**72.8**	**56.6**	**47.0**	**37.2**	**27.0**	**20.1**	**43.8**	**112.1**	**161.6**	**102.2**

Dataset and Methodology: The original dataset in Paulus et al. has 4500 train and 500 test instances with fixed $N = 10$ items, extracted from a corpus containing $50,000$ sentences and their embeddings. In addition to experimenting with the original dataset, to demonstrate the scalability of our method, we also bootstrap new datasets with $N = 15, 20, 25$ and 30, by randomly selecting sentences from the original corpus. Each bootstrapped dataset also has 4500 train and 500 test instances. Following Paulus et al., we use an MLP to extract $\mathbf{c}$ and $\mathbf{A}$ from N d-dimensional sentence embeddings, and keep $\mathbf{b} = 1$. We keep number of constraints as $m = 4$, the setting which achieves best performance in Paulus et al..

Results: Table 3 presents a comparison between the training time and accuracy of our method against CombOptNet. We significantly outperform the baseline, both in terms of accuracy and train-times across all the datasets. Note that our relative gain increases significantly with increasing problem size.

CombOptNet fails to complete even a single epoch in the stipulated time and results in 0% accuracy on the largest problem.

4.4 Keypoint Matching

Here we experiment on a real world task of matching key points between two images of different objects of same type Rolínek et al. [2020b]. The input x consists of a pair of images (I_1, I_2) of same type along with a shuffled list of coordinates (pixels) of k keypoints in both images. The task is to match the same keypoints in the two images with each other. The ground truth constraint enforces a bijection between the keypoints in the two images. The output $\mathbf{y}$ is represented as a $k \times k$ binary permutation matrix, *i.e.*, entries in a row or column should sum to 1. The cost $\mathbf{c}$ is a $k \times k$ sized vector parameterized by $\theta_{\mathbf{c}}$. The goal is to learn $\mathbf{A}$, $\mathbf{b}$ and $\theta_{\mathbf{c}}$.

Dataset: We experiment with image pairs in SPair-17k [Min et al., 2019] dataset used for the task of keypoint matching in [Rolínek et al., 2020b]. Since the neural ILP models can deal with a fixed size output space, we artificially create 4 datasets for 4,5,6 and 7 keypoints from the original dataset. While generating samples for k keypoints dataset, we randomly sample any k annotated pairs from input image pairs that have more than k keypoints annotated. See appendix for details on dataset size.

Baselines: We compare against a strong neural baseline which is same as the backbone model parameterizing the k^2 dimensional cost vector in [Rolínek et al., 2020b]. It is trained by minimizing BCE loss between negative learnt cost $(-\mathbf{c})$ and target $\mathbf{y}^*$. We create an additional baseline by doing constrained inference with ground truth constraints and learnt cost ('Neural + CI') in table 4.

Results: Table 4 presents the percentage of keypoints matched correctly by different models. Both CombOptNet and our method achieve similar performance. Note that we obtain a gain of 2 points over the purely neural baseline without the knowledge of constraints. This is almost same as the gain obtained by constrained inference with known constraints on top of neural model. For this task, we did not observe any gain in the training time over CombOptNet. This is expected as the bottleneck here is the backbone neural model parameterizing the cost function. Even though our method spends less time in each iteration, it takes more number of iterations to train, balancing out the gain obtained by speeding up each iteration.

Table 4: Pointwise accuracy and train-times for 4 datasets with varying number of keypoints. Neural+CI is the ILP inference with the known constraints over the cost learnt by neural model.

	4	**5**	**6**	**7**	**Avg.**
Neural	81.98	78.93	76.06	74.68	77.91
Neural + CI	83.32	80.28	77.68	76.66	79.49
CombOptNet	**83.63**	80.91	77.29	76.28	79.54
ILP–Loss (Ours)	83.50	**80.98**	**78.62**	**76.80**	**79.98**
	Training Times (in min)				
Neural	108	**12**	**17**	**41**	**44**
CombOptNet	**89**	98	50	57	74
ILP–Loss (Ours)	155	77	86	100	105

5 Conclusion and Future Work

We have presented a solver–free framework for scalable training of a neural ILP architecture. Our method learns the linear constraints by viewing them as linear binary classifiers, separating the positive points (inside the polyhedron) from the negative points (outside the polyhedron). While given ground truth acts as positives, we propose multiple strategies for sampling negatives. A simple trick using the available ground truth outputs in the training data, converts the cost vector into a constraint, enabling us to learn the cost vector and constraints in a similar fashion.

Future work involves extending our method to *neural-ILP-neural* architectures, i.e., where the output of ILP is an input to a downstream neural model (see appendix for a detailed discussion). Second, a neural ILP model works with a fixed dimensional output space, even though the constraints for the same underlying problem are the same in first order logic, *e.g.*, constraints for $k \times k$ sudoku puzzles remain the same in first order irrespective of k. Creating neural ILP models that can parameterize the constraints on the basis of the size of the input (or learnt) cost vector is a potential direction for future work. Lastly, the constraints learnt in a neural ILP architecture might not be interpretable even if the ground truth constraints are. In future, we would like to develop methods that distill the learnt constraints into a human-interpretable form.

References

Ananye Agarwal, Pradeep Shenoy, and Mausam. End-to-end neuro-symbolic architecture for image-to-image reasoning tasks. *CoRR*, abs/2106.03121, 2021. URL https://arxiv.org/abs/2106.03121.

Akshay Agrawal, Brandon Amos, Shane T. Barratt, Stephen P. Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 9558–9570, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/9ce3c52fc54362e22053399d3181c638-Abstract.html.

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019a. URL https://openreview.net/forum?id=BJxgz2R9t7.

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. PDP: A general neural framework for learning constraint satisfaction solvers. *CoRR*, abs/1903.01969, 2019b. URL http://arxiv.org/abs/1903.01969.

Brandon Amos and J. Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 136–145. PMLR, 2017. URL http://proceedings.mlr.press/v70/amos17a.html.

Vaibhav Arcot and Samarth Kalluraya. Data set of 4x4 sudoku puzzles and solutions. https://github.com/Black-Phoenix/4x4-Sudoku-Dataset, 2019.

Andrew Bartlett, Timothy Chartier, Amy Langville, and Timothy Rankin. An integer programming model for the sudoku problem. 04 2008.

Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9508–9519. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/6bb56208f672af0dd65451f869fedfd9-Paper.pdf.

Timothy C. Y. Chan, Rafid Mahmood, and Ian Yihang Zhu. Inverse optimization: Theory and applications. 2021.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 670–680. Association for Computational Linguistics, 2017. doi: 10.18653/v1/d17-1070. URL https://doi.org/10.18653/v1/d17-1070.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=B1xY-hRctX.

Aaron M. Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY,*

USA, February 7-12, 2020, pages 1504–1511. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/5509.

Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs *and* an algorithm for the mixed integer problem. In Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 77–103. Springer, 2010. doi: 10.1007/978-3-540-68279-0_4. URL https://doi.org/10.1007/978-3-540-68279-0_4.

Stephen Gould, Richard Hartley, and Dylan Campbell. Deep declarative networks: A new hope. *CoRR*, abs/1909.04866, 2019. URL http://arxiv.org/abs/1909.04866.

Rick Groenendijk, Sezer Karaoglu, Theo Gevers, and Thomas Mensink. Multi-loss weighting with coefficient of variations. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*, pages 1468–1477. IEEE, 2021. doi: 10.1109/WACV48630.2021.00151. URL https://doi.org/10.1109/WACV48630.2021.00151.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

Alex Kantchelian, Michael Carl Tschantz, Ling Huang, Peter L. Bartlett, Anthony D. Joseph, and J. Doug Tygar. Large-margin convex polytope machine. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3248–3256, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/b147a61c1d07c1c999560f62add6dbc7-Abstract.html.

Jayanta Mandi and Tias Guns. Interior point solving for lp-based prediction+optimisation. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/51311013e51adebc3c34d2cc591fefee-Abstract.html.

Tao Meng and Kai-Wei Chang. An integer linear programming framework for mining constraints from data. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 7619–7631. PMLR, 2021. URL http://proceedings.mlr.press/v139/meng21a.html.

Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. Spair-71k: A large-scale benchmark for semantic correspondence. *CoRR*, abs/1908.10543, 2019. URL http://arxiv.org/abs/1908.10543.

Giovanni Mingari Scarpello and Daniele Ritelli. A historical outline of the theorem of implicit functions. *Divulgaciones Matemáticas*, 10, 01 2002.

Yatin Nandwani, Deepanshu Jindal, Mausam, and Parag Singla. Neural learning of one-of-many solutions for combinatorial problems in structured output spaces. In *International Conference on Learning Representations (ICLR)*, 2021.

Yatin Nandwani, Vidit Jain, Mausam, and Parag Singla. Neural models for output-space invariance in combinatorial problems. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2202.03229.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit mle: Backpropagating through discrete exponential family distributions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 14567–14579. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/7a430339c10c642c4b2251756fd1b484-Paper.pdf.

Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 3372–3382, 2018. URL http://papers.nips.cc/paper/7597-recurrent-relational-networks.

Xingyuan Pan and Vivek Srikumar. Expressiveness of rectifier networks. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2427–2435. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/panb16.html.

Xingyuan Pan, Maitrey Mehta, and Vivek Srikumar. Learning constraints for structured prediction using rectifier networks. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4843–4858. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.438. URL https://doi.org/10.18653/v1/2020.acl-main.438.

George Papandreou and Alan L. Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. In Dimitris N. Metaxas, Long Quan, Alberto Sanfeliu, and Luc Van Gool, editors, *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 193–200. IEEE Computer Society, 2011. doi: 10.1109/ICCV.2011.6126242. URL https://doi.org/10.1109/ICCV.2011.6126242.

Kyubyong Park. 1 million sudoku games. https://www.kaggle.com/datasets/bryanpark/sudoku, 2017.

Anselm Paulus, Michal Rolínek, Vít Musil, Brandon Amos, and Georg Martius. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8443–8453. PMLR, 2021. URL http://proceedings.mlr.press/v139/paulus21a.html.

Marin Vlastelica Pogancic, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BkevoJSYPB.

Michal Rolínek, Vít Musil, Anselm Paulus, Marin Vlastelica P., Claudio Michaelis, and Georg Martius. Optimizing rank-based metrics with blackbox differentiation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*, pages 7617–7627. Computer Vision Foundation / IEEE, 2020a. doi: 10.1109/CVPR42600.2020.00764. URL https://openaccess.thecvf.com/content_CVPR_2020/html/Rolinek_Optimizing_Rank-Based_Metrics_With_Blackbox_Differentiation_CVPR_2020_paper.html.

Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XXVIII*, volume 12373 of *Lecture Notes in Computer Science*, pages 407–424. Springer, 2020b. doi: 10.1007/978-3-030-58604-1_25. URL https://doi.org/10.1007/978-3-030-58604-1_25.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HJMC_iA5tm.

Yingcong Tan, Andrew Delong, and Daria Terekhov. Deep inverse optimization. In Louis-Martin Rousseau and Kostas Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019,*

Thessaloniki, Greece, June 4-7, 2019, Proceedings, volume 11494 of *Lecture Notes in Computer Science*, pages 540–556. Springer, 2019. doi: 10.1007/978-3-030-19212-9_36. URL https://doi.org/10.1007/978-3-030-19212-9_36.

Yingcong Tan, Daria Terekhov, and Andrew Delong. Learning linear programs from optimal decisions. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e44e875c12109e4fa3716c05008048b2-Abstract.html.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 2019. URL http://proceedings.mlr.press/v97/wang19e.html.

Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1658–1665. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33011658. URL https://doi.org/10.1609/aaai.v33i01.33011658.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] . In Conclusion and future works section 5
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] We will make the code and data available on acceptance
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Details are in experiments section 4 under 'Training Methodology' and in the appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In one of the experiments on random constraints.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] In the experiments section 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Datasets are cited in the respective experiment sections.
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

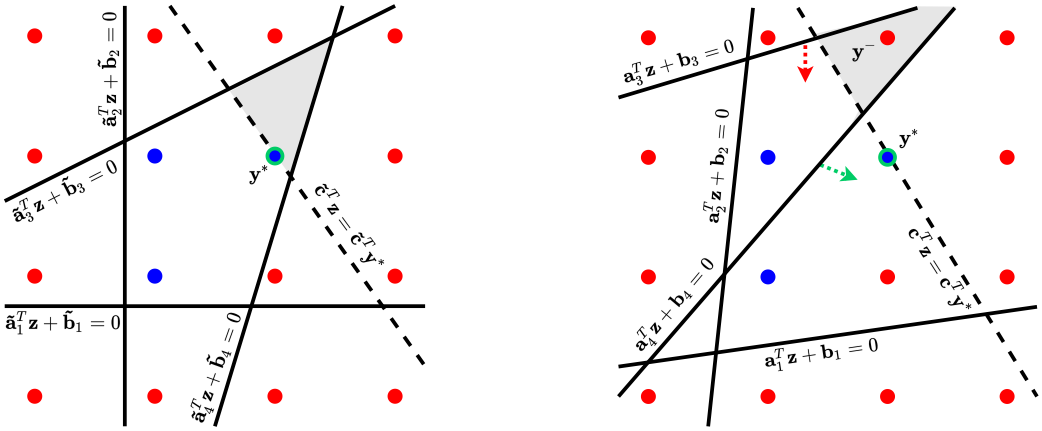
(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

3 Differentiable ILP Loss

3.2 A Solver-free Framework



(a) Ground Truth ILP

(b) Intermediate Learnt ILP

Figure 1: An illustration of our framework. Figure on the left shows 4 ground truth constraints that need to be learnt. Blue dots are the only feasible integral points w.r.t. the 4 constraints. Shaded area containing only the dot with green border is the feasible region after we add the cost constraint (dashed line). Figure on the right shows an intermediate scenario while learning. The green-bordered dot (positive) is outside the intermediate 4th constraint and the red dot (negative) is inside the intermediate 3rd constraint. Positive and negative losses encourage the 4th and the 3rd hyperplanes to move in the direction shown by the green and red dotted arrows respectively.

In fig. 1, we consider a 2-dimensional ILP with 4 ground truth constraints $\tilde{\mathbf{a}_\mathbf{i}}^T\mathbf{z}+\tilde{\mathbf{b}_\mathbf{i}} \geq 0, i = 1, 2, 3, 4$, a cost vector $\tilde{\mathbf{c}}$, and target solution $\mathbf{y}^*$. The dots represent points in $\mathbb{Z}^2$. The solid lines represent hyperplanes (here lines) corresponding to the constraints, and the dashed line represents our cost-constraint. The shaded area in fig. 1a is the feasible region of the constraint satisfaction problem:

$$\arg\min_{\mathbf{z}} \mathbf{0}^T\mathbf{z} \;\text{ subject to }\; \tilde{\mathbf{A}}\mathbf{z}+\tilde{\mathbf{b}} \geq \mathbf{0} \;\;;\;\; \tilde{\mathbf{c}}^T\mathbf{z} \leq \tilde{\mathbf{c}}^T\mathbf{y}^* \;\;;\;\; \mathbf{z} \in \mathbb{Z}^2$$

Figure 1a shows the ground truth ILP. The signs of $\tilde{\mathbf{a}_\mathbf{i}}$'s and $\tilde{\mathbf{b}}_i$'s are such that the closed polyhedron (here polygon) containing the blue points forms the feasible region $\tilde{\mathbf{A}}\mathbf{z}+\tilde{\mathbf{b}} \geq \mathbf{0}$. The green point is optimal w.r.t. the cost $\tilde{\mathbf{c}}$. The red points are infeasible, *i.e.*, $\tilde{\mathbf{A}}\mathbf{z}+\tilde{\mathbf{b}} < \mathbf{0}$ for red points. Note that the target solution $\mathbf{y}^*$ (point with green border) is the only integral point in the shaded region. The other blue points violate the cost constraint $\tilde{\mathbf{c}}^T\mathbf{z} \leq \tilde{\mathbf{c}}^T\mathbf{y}^*$, and the red points violate at least one of the four feasibility constraints $\tilde{\mathbf{a}_\mathbf{i}}^T\mathbf{z}+\tilde{\mathbf{b}}_i \geq 0, i \in \{1 \cdots 4\}$.

Figure 1b shows a possible situation during learning. For simplicity, consider temperature close to zero ($\tau \approx 0$), so that only the closest hyperplane contributes to the negative loss for a negative sample. Also consider margins close to zero ($\mu^+ \approx 0, \mu^- \approx 0$). In fig. 1b, the point with the green border is outside the shaded region, whereas one red point is inside. The ground truth $\mathbf{y}^*$ is on the wrong side of only the fourth hyperplane, *i.e.*, $\mathbf{a}_4^T\mathbf{y}^* + \mathbf{b}_4 < 0$, and hence it is the only contributor to the positive loss, *i.e.*, $L_+ = -\frac{1}{4}d(\mathbf{y}^*; [\mathbf{a}_4|\mathbf{b}_4])$, where $d(\mathbf{y}^*; [\mathbf{a}_4|\mathbf{b}_4]) = \frac{\mathbf{a}_4^T\mathbf{y}^*+\mathbf{b}_4}{|\mathbf{a}_4|} < 0$. The red negative point denoted as $\mathbf{y}^-$ (inside the shaded region) being closest to the third hyperplane contributes $d(\mathbf{y}^-; [\mathbf{a}_3|\mathbf{b}_3]) = \frac{\mathbf{a}_3^T\mathbf{y}^-+\mathbf{b}_3}{|\mathbf{a}_3|} > 0$ to the negative loss L_-. The green and red dotted arrows indicate the directions of movement of the constraint hyperplanes on weight update.

3.2 A Solver-free Framework

Other details:

Parameterization of equality constraints: Recall that we replace an equality $\mathbf{u}_i^T\mathbf{z} = \mathbf{v}_i$ by two inequality constraints: $\mathbf{v}_i \leq \mathbf{u}_i^T\mathbf{z} \leq \mathbf{v}_i$. In practice, to enhance learnability, we add a small margin of ϵ on both sides: $\mathbf{v}_i - \epsilon \leq \mathbf{u}_i^T\mathbf{z} \leq \mathbf{v}_i + \epsilon$. We pick an ϵ small enough so that the probability of the new feasible region to include an infeasible integral point is negligible.
Known boundaries: In many cases, the boundary conditions on the output variables are known, *i.e.*, $\mathbf{z} \in \mathcal{Y} = [l, u]^n$, where l and u are the lower and upper bounds on each dimension of $\mathbf{z}$. We handle this by adding known boundary constraints: $l \leq \mathbf{z}_i \leq u, \forall i \in \{1 \cdots n\}$.
Over-parameterization of constraints: As done in Paulus et al., we also over-parameterize each constraint hyperplane by an additional learnable offset vector $\mathbf{o}_i$ which can be viewed as its own local origin. Radius $\mathbf{r}_i$ represents its distance from its own origin $\mathbf{o}_i$, resulting in the following hyperplane in the base coordinate system: $\mathbf{a}_i^T\mathbf{z} + \mathbf{r}_i - \mathbf{o}_i^T\mathbf{a}_i/|\mathbf{a}_i| \geq 0$.
Initialization of A: The way we initialize the constraints may have an impact on the learnability. While [Paulus et al., 2021] propose to sample each entry of $\mathbf{a}_i$ uniformly (and independently) between $[-0.5, 0.5]$, we also experiment with a standard Gaussian initialization. The latter results in initial hyperplanes with their normal directions ($\mathbf{a}_i$'s) uniformly sampled from a unit hyper-sphere. In expectation, such initialization achieves minima of L_o that measures total pairwise covariance.

4 Experiments

Details of the ILP Solver and the hardware used for experiments: To solve the learnt ILPs, we use Gurobi ILP solver [Gurobi Optimization, LLC, 2022] available under 'Named-user academic license'. All our experiments were run on 11 GB 'GeForce GTX 1080 Ti' GPUs installed on a machine with 2.60GHz Intel(R) Xeon(R) Gold 6142 CPU. For each of the algorithms, we kept a maximum time limit of 12 hours for training.

4.1 Visual Sudoku

Hyperparameters and other design choices: (a) # of learnable constraints: For $k = 4$ and 6, we keep $m = (n + 1)$ as the number of learnable constraints where $n = k^3$ is the number of binary variables. For $k = 9$, we learn $m = (n + 1)/2$ equality constraints, which is equivalent to learning $(n + 1)$ inequality constraints with parameter tying. (b) Margin: We find that a margin of 0.01 works well across domains and problem sizes. (c) Temperature: we start with a temperature $\tau = 1$ and anneal it by a factor of 0.1 whenever the performance on a small held out set plateaus. (d) Early Stopping: we early stop the training based on validation set performance, with a timeout of 12 hours for each experiment. We bypass the validation bottleneck of solving the ILPs from scratch by providing the gold solutions as hints when invoking Gurobi. (e) Negative Sampling: For sampling neighbors, we select all the n one hop neighbors, and an equal number of randomly selected 2,3 and 4 hop neighbors, resulting in a total of $4n$ neighbors. (f) Initialization: we initialize $\mathbf{a}_i$ from a standard Gaussian distribution for CombOptNet and our method.

Comparison with SATNet on their dataset: Wang et al. use a different set of 9×9 puzzles for training and testing sudoku and report 63.2% accuracy, different from what it obtains on our dataset. Hence we trained both SATNet and our model on the dataset available on SATNet's Github repo.

Table 5: Keypoint Matching: Number of train and test samples for datasets with different keypoints

Num Keypoints	**4**	**5**	**6**	**7**
#Test	10,474	9,308	7,910	6,580
#Train	43,916	37,790	31,782	26,312

Interestingly, on their dataset which we believe to be easier (as shown by performance numbers), our run of SATNet achieves 71.0% board accuracy whereas our method achieves 98.3%.

4.2 Random Constraints

Hyperparameters and other design choices: we keep number of learnable constraints as twice the number of ground truth constraints, *i.e.*, $m = 2m'$, as Paulus et al. report best performance in most of the settings with it. Here we initialize $\mathbf{a}_i$ uniformly between $[-0.5, 0.5]$. Rest of the hyperparamters are set as in the case of sudoku.

4.4 Keypoint Matching

Hyperparameters: For each k, the number of learnable constraints is set to $2k$: same as the number of ground truth constraints. In our method, for keypoints 5,6 and 7, we initialize the backbone cost parameters $\theta_{\mathbf{c}}$ with the one obtained by training it on 4 keypoints. We start with a temperature $\tau = 0.5$ and anneal it by a factor of 0.2 at 10th and 30th epoch. Here we initialize $\mathbf{a}_i$ uniformly between $[-0.5, 0.5]$. Rest of the hyperparmeters are same as those used for sudoku.

Dataset details: See table 5 for the number of train and test samples in the four datasets created for $4, 5, 6$, and 7 keypoints.

5 Future Work

Discussion on training Neural-ILP-Neural architectures: In the current formulation, availability of the solution to the ground truth ILP is important for our solver-free approach to work. Specifically, it is required to: 1.) convert the constrained optimization problem to a constraint satisfaction problem by including the cost-constraint eq. (4), and 2.) to calculate the positive loss eq. (6). However, in a Neural-ILP-Neural architecture, the intermediate supervision for only the Neural-ILP part (*i.e.*, solution of the ground truth ILP) is not available.

On the other hand, solver based methods such as CombOptNet, do not require access to the solution of the ground-truth ILP. Instead, they rely on the solution of the current intermediate ILP (during learning) to compute the gradients of the loss and thus their approach is not solver free. We note that even though in princple they can train Neural-ILP-Neural architectures, their experiments are only in the Neural-ILP settings.

Extending our current work for Neural-ILP-Neural architectures is an important direction of future work. One plausible approach could be to train an auxiliary inverse network that converts a given output of Neural-ILP-Neural architecture to a predicted symbolic target of Neural-ILP component. This predicted target can be used as a proxy to the ground truth solution of the Neural-ILP part. Similar ideas of using an inverse network have been explored in [Agarwal et al., 2021], albeit under the setting where ILP is known and only the neural encoder and decoder needs to be learnt.