DL2: Training and Querying Neural Networks with Logic

Marc Fischer [1] **Mislav Balunović** [1] **Dana Drachsler-Cohen** [1] **Timon Gehr** [1] **Ce Zhang** [1] **Martin Vechev** [1]

Abstract

We present DL2, a system for training and querying neural networks with logical constraints. Using DL2, one can declaratively specify domain knowledge constraints to be enforced during training, as well as pose queries on the model to find inputs that satisfy a set of constraints. DL2 works by translating logical constraints into a loss function with desirable mathematical properties. The loss is then minimized with standard gradient-based methods. We evaluate DL2 by training networks with interesting constraints in unsupervised, semi-supervised and supervised settings. Our experimental evaluation demonstrates that DL2 is more expressive than prior approaches combining logic and neural networks, and its loss functions are better suited for optimization. Further, we show that for a number of queries, DL2 can find the desired inputs in seconds (even for large models such as ResNet-50 on ImageNet).

1. Introduction

With the success of neural networks across a wide range of application domains, an important emerging challenge is creating better, more flexible mechanisms for experts to interact with the network. For example, one may wish the network to capture certain background knowledge not easily available as labeled training data or to inquire how the network's decision changes under particular domain-specific inputs. Fundamentally, to cleanly capture such interactions, one needs some form of logical reasoning and a way to combine this reasoning with the neural network's training and decision making procedures. Indeed, recent work (Hu et al., 2016; Xu et al., 2018) has started investigating novel combinations of neural and logical reasoning. However, while promising, as we show later, these approaches lack generality and cannot capture important use cases.

[1]Department of Computer Science, ETH Zurich, Switzerland. Correspondence to: Marc Fischer <marc.fischer@inf.ethz.ch>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

We introduce a new method and system, called DL2[2] (acronym for Deep Learning with Differentiable Logic), which can be used to: (i) query networks for inputs meeting logical constraints, and (ii) train networks to meet logical specifications, all in a *declarative* way. Our constraint language can express rich combinations of comparisons over inputs, neurons and outputs of neural networks using negations, conjunctions, and disjunctions. Thanks to its expressiveness, DL2 enables users to easily incorporate domain knowledge during training and to interact with the network using queries in order to inquire about its decision making.

Technically, DL2 translates logical constraints into non-negative loss functions with two key properties: (i) the loss is zero exactly if the constraints are satisfied, and (ii) the loss is differentiable almost everywhere. Combined, these properties enable us to train and query with constraints by minimizing a loss with off-the-shelf optimizers.

Training with DL2 To make optimization tractable, we extract constraints on inputs that capture certain kinds of convex sets and use them as optimization constraints instead of including them in the optimization goal. We then optimize with projected gradient descent (PGD), which has already been shown successful in training with robustness constraints (Madry et al., 2018). The expressiveness of DL2 along with tractable optimization with PGD enables us to train with new, interesting constraints. For example, we can declaratively express constraints over quantities which are not explicitly computed by the network, such as

$$p_\theta(\boldsymbol{x})_{\mathit{people}} < \epsilon \vee p_\theta(\boldsymbol{x})_{\mathit{people}} > 1 - \epsilon.$$

This constraint on the output activations of a CIFAR-100 classifier p_θ says that for a network input $\boldsymbol{x}$, the probability of *people*, denoted $p_\theta(\boldsymbol{x})_{\mathit{people}}$, is either very small or very large. As CIFAR-100 does not have a class *people*, we define it as a function of other output activations:

$$p_\theta(\boldsymbol{x})_{\mathit{people}} = p_\theta(\boldsymbol{x})_{\mathit{baby}} + p_\theta(\boldsymbol{x})_{\mathit{boy}} + p_\theta(\boldsymbol{x})_{\mathit{woman}} + \cdots$$

We show that DL2 can make misclassifications less severe (e.g., a boy is misclassified as a man instead of as a bicycle) and it can even increase the accuracy of CIFAR-100 networks in the semi-supervised setting.

[2]`https://github.com/eth-sri/dl2`

Beyond classification tasks, DL2 can capture constraints arising in regression tasks. For example, GalaxyGAN (Schawinski et al., 2017), a generator of galaxy images, requires the network to respect constraints imposed by the underlying physical systems, e.g., flux: the sum of input activations should equal the sum of output activations. Instead of hardcoding such a constraint into the network in an ad hoc manner, this can be expressed with DL2 declaratively as $sum(\boldsymbol{x}) = sum(\text{GalaxyGAN}(\boldsymbol{x}))$.

Global Training An important feature of DL2 is its ability to train with constraints that place restrictions on inputs *outside the training set*. Most prior work on training with constraints (e.g., Xu et al., 2018) focuses on the given training set to *locally train* the network to meet the constraints. With DL2, we can query for inputs *outside* the training set which violate the constraints, and use them to *globally train* the network. Prior work that trained on examples outside the training set was tailored either to specific constraints (Madry et al., 2018) or network types (Minervini et al., 2017). Our approach splits the task of global training between: (i) an optimizer, which trains the network to meet the constraints for the given inputs, and (ii) an adversary, which provides the optimizer with new inputs that aim to violate the constraints. To illustrate, consider the following Lipschitz condition:

$$\forall \boldsymbol{z} \in B_\epsilon(\boldsymbol{x}), \boldsymbol{z}' \in B_\epsilon(\boldsymbol{x}').\ \|f(\boldsymbol{z}) - f(\boldsymbol{z}')\|_2 < L\|\boldsymbol{z} - \boldsymbol{z}'\|_2$$

Here, for two inputs from the training set ($\boldsymbol{x}$ and $\boldsymbol{x}'$), any pair of points in their respective ϵ-neighborhoods ($\boldsymbol{z}$ and $\boldsymbol{z}'$) must satisfy the condition. This constraint is inspired by recent works (e.g., Gouk et al., 2018; Balan et al., 2017) which have shown that neural networks are more stable if they satisfy the Lipschitz condition.

Querying with DL2 We also designed an SQL-like language which enables users to interact with the model by posing declarative queries. For example, consider the recent work of Song et al. (2018) which shows how to generate adversarial examples with AC-GANs (Odena et al., 2017). The generator is used to create images from a certain class (e.g., 1) which fool a classifier (to classify as, e.g., 7). With DL2, this can be phrased as the following query:

```
find n[100]
where n in [-1, 1],
      class(M_NN1(M_ACGAN_G(n, 1))) = 7
return M_ACGAN_G(n, 1)
```

This query looks for an input $\text{n} \in \mathbb{R}^{100}$ to the generator satisfying two constraints: its entries are between -1 and 1 (a domain constraint) and it results in the generator producing an image which it believes to be classified as 1 (enforced by `M_ACGAN_G(n, 1)`), but is classified by the network (`M_NN1`) as 7. DL2 automatically translates this query to a DL2 loss and optimizes it with an off-the-shelf optimizer (L-BFGS-B) to find solutions. Our language captures various prior works at the declarative level, including finding neurons responsible for a given prediction (Olah et al., 2018), inputs that differentiate two networks (Pei et al., 2017), and adversarial examples (e.g., Szegedy et al., 2014).

Main Contributions The main contributions are:

- An approach for training and querying neural networks with logical constraints, which translates constraints into a loss function with desirable properties.
- A training procedure which extracts constraints on inputs that capture convex sets and includes them as PGD constraints, making optimization tractable.
- A declarative language for querying neural network inputs, outputs, and internal neurons. Queries are compiled into a loss and optimized with L-BFGS-B.
- An extensive evaluation demonstrating that DL2 is effective for querying and training neural networks. Among other experimental results, DL2 is shown to successfully train networks with constraints on inputs outside the training set.

2. Related Work

Constraints and Continuous Reasoning Several works integrate logical constraints and continuous reasoning. The closest ones to our setting are Probabilistic Soft Logic (PSL) (Kimmig et al., 2012; Hu et al., 2016), Xu et al. (2018), XSAT (Fu & Su, 2016), and Bach et al. (2017). We discuss these works in comparison to DL2 in Section 3.1.

Adversarial Examples and Training Adversarial example generation (Pei et al., 2017; Goodfellow et al., 2015) can be seen as a fixed query on the network, while adversarial training (Madry et al., 2018) aims to enforce a specific constraint. Both can be expressed in DL2. Most works aiming to train networks with logic impose soft constraints, often using additional loss terms (Pathak et al., 2015; Xu et al., 2018); Márquez-Neila et al. (2017) show that hard constraints have no empirical advantage over soft constraints. Similar to our training algorithm (Section 4), Minervini & Riedel (2018) find adversarial examples violating logical rules and incorporate them into a network's training. Unlike our approach, theirs is specific to Natural Language Inference and uses discrete search to find counterexamples.

Knowledge Bases and Neural-Symbolic Approaches Combinations of neural networks and symbolic methods (Besold et al., 2017) as well as embedding logical relations into vector spaces for completion and querying have been studied extensively. Rocktäschel et al. (2015) translates logical rules into a loss in order to derive an embedding for logical relations. Other works extend this approach to knowledge graphs (Guo et al., 2016) and scale to larger numbers of implication rules (Demeester et al., 2016). Guu et al. (2015) performs knowledge base queries by translating logical path queries into numerical computation over a learned model. Rocktäschel & Riedel (2017) introduces Neural Theorem Provers, which are neural networks that learn to prove basic theorems over (incomplete) knowledge bases. Yang et al. (2017) introduces Neural Logic Programming, a model able to learn logical rules for reasoning in a differentiable manner. Evans & Grefenstette (2018) show a similar approach, while thoroughly investigating underlying loss translations and showing robustness to noisy data.

3. From Logical Constraints to Loss

We now present our constraint language and show how to translate logical constraints into a (non-negative) loss.

Logical Language Our language of logical constraints consists of boolean combinations of comparisons between terms. A term t is defined over variables $\boldsymbol{x}$ and neural network parameters θ. In our translation, terms are translated to themselves. Therefore, a term can be any real-valued function that might appear as a subexpression of a loss function, such as a constant or an output activation $p_\theta(\boldsymbol{x})_i$ of a neural network. We require terms to be differentiable almost everywhere, in both $\boldsymbol{x}$ and θ.

Two terms can be combined to form comparison constraints. For two terms t and t', we can obtain constraints $t = t'$, $t \leq t'$, $t \neq t'$ and $t < t'$.

A constraint φ is either a comparison constraint, a conjunction $\varphi' \wedge \varphi''$ or a disjunction $\varphi' \vee \varphi''$ of two constraints φ' and φ'', or a negation $\neg\varphi$ of a constraint φ.

We write $\varphi(\boldsymbol{x}, \theta)$ to evaluate a constraint at values $\boldsymbol{x}$ and θ.

Translation into Loss To each constraint φ, we associate a corresponding non-negative loss function $\mathcal{L}(\varphi)$. We construct $\mathcal{L}(\varphi)$ such that it is differentiable in $\boldsymbol{x}$ and θ almost everywhere and such that we have $\mathcal{L}(\varphi)(\boldsymbol{x}, \theta) = 0$ if and only if $\varphi(\boldsymbol{x}, \theta)$ is satisfied.

We define $\mathcal{L}(\varphi)$ recursively, based on the different possible shapes of φ. We first consider the case where φ is a comparison constraint $t \leq t'$ or $t \neq t'$:

$$\begin{aligned} \mathcal{L}(t \leq t') &:= \max(t - t', 0), \\ \mathcal{L}(t \neq t') &:= \xi \cdot [t = t']. \end{aligned}$$

Here, $\xi > 0$ is a parameter to our translation and $[t = t']$ is an indicator function. Note that our definition of $\mathcal{L}(t \neq t')$ is canonical, but there are other possible choices for $\mathcal{L}(t \leq t')$.

Comparison constraints $t = t'$ and $t < t'$ are rewritten into logically equivalent constraints before translation into loss:

$$\begin{aligned} \mathcal{L}(t = t') &:= \mathcal{L}(t \leq t' \wedge t' \leq t), \\ \mathcal{L}(t < t') &:= \mathcal{L}(t \leq t' \wedge t \neq t'). \end{aligned}$$

We next consider the case where φ is a boolean combination of constraints $\varphi' \wedge \varphi''$ or $\varphi' \vee \varphi''$:

$$\begin{aligned} \mathcal{L}(\varphi' \wedge \varphi'') &:= \mathcal{L}(\varphi') + \mathcal{L}(\varphi''), \\ \mathcal{L}(\varphi' \vee \varphi'') &:= \mathcal{L}(\varphi') \cdot \mathcal{L}(\varphi''). \end{aligned}$$

Note that $\mathcal{L}(\varphi' \wedge \varphi'') = 0$ if and only if $\mathcal{L}(\varphi') = 0$ *and* $\mathcal{L}(\varphi'') = 0$, which by construction is true if φ' and φ'' are satisfied, and similarly $\mathcal{L}(\varphi' \vee \varphi'') = 0$ if and only if $\mathcal{L}(\varphi') = 0$ *or* $\mathcal{L}(\varphi'') = 0$.

If φ is a negation $\neg\psi$, it is rewritten into a logically equivalent constraint before translation into loss:

$$\begin{aligned} \mathcal{L}(\neg(t = t')) &:= \mathcal{L}(t \neq t'), \\ \mathcal{L}(\neg(t \leq t')) &:= \mathcal{L}(t' < t), \\ \mathcal{L}(\neg(t \neq t')) &:= \mathcal{L}(t = t'), \\ \mathcal{L}(\neg(t < t')) &:= \mathcal{L}(t' \leq t), \\ \mathcal{L}(\neg(\psi' \wedge \psi'')) &:= \mathcal{L}(\neg\psi' \vee \neg\psi''), \\ \mathcal{L}(\neg(\psi' \vee \psi'')) &:= \mathcal{L}(\neg\psi' \wedge \neg\psi''), \\ \mathcal{L}(\neg(\neg\varphi')) &:= \mathcal{L}(\varphi'). \end{aligned}$$

By construction, we obtain the following theorem:

Theorem 1 *$\mathcal{L}(\varphi) = 0$ if and only if φ is satisfied.*

3.1. Comparison of DL2 with Prior Works

We next discuss the works most closely related to ours, and contrast them with DL2. Probabilistic Soft Logic (PSL) (Kimmig et al., 2012) translates logical constraints into continuous almost-everywhere differentiable loss functions over $[0, 1]$. However, using this loss to find satisfying assignments with gradient methods can be futile, as the gradient may be zero. To illustrate, consider the toy example $\varphi(\boldsymbol{x}) := (\boldsymbol{x} = \binom{1}{1})$. PSL translates this formula into the loss $\mathcal{L}_{\text{PSL}}(\varphi)(\boldsymbol{x}) = \max\{x_0 + x_1 - 1, 0\}$ (it assumes $x_0, x_1 \in [0, 1]$). Assuming optimization starts from $\boldsymbol{x} = \binom{0.2}{0.2}$ (or any pair of numbers with $x_0 + x_1 - 1 \leq 0$), the gradient is $\nabla_{\boldsymbol{x}} \mathcal{L}_{\text{PSL}}(\varphi)(\boldsymbol{x}) = \binom{0}{0}$, which means the optimization cannot continue from this point, even though $\boldsymbol{x}$ is not a satisfying assignment to φ. In contrast, with our translation, we obtain $\mathcal{L}(\varphi)(\boldsymbol{x}) = |x_0 - 1| + |x_1 - 1|$, for which the gradient for the same $\boldsymbol{x}$ is $\nabla_{\boldsymbol{x}} \mathcal{L}(\varphi)(\boldsymbol{x}) = \binom{-1}{-1}$.

Evans & Grefenstette (2018) also analyse different standard choices (t-norms) encodings for boolean combinations and find multiplication to be best suited for gradient descent.

Hu et al. (2016) build on PSL and present a teacher-student framework which distills rules into the training phase. The idea is to formulate rule satisfaction as a convex problem with a closed-form solution. However, this formulation is restricted to rules over inputs and output classes, and cannot express rules constraining the numerical values of output activations. In contrast, DL2 can express such constraints, e.g., $p_1 > p_2$, which requires that the output activation for class 1 is greater than for class 2. Also, the convexity and existence of a closed-form solution stem from the linearity of the rules in the network's output, meaning that non-linear constraints (e.g., Lipschitz condition, expressible with DL2) are fundamentally beyond the reach of this method. The work of Xu et al. (2018) similarly is restricted to constraints over output probabilities and is intractable for large constraints in many variables. Unlike DL2, both of these works do not support constraints for regression tasks.

XSAT (Fu & Su, 2016) reduces the satisfiability of floating-point formulas to numerical optimization. To this end, XSAT also translates logical constraints into numerical loss. However, its atomic constraints are translated into a discrete, nowhere-differentiable loss based on floating-point representation, unamenable to gradient-based optimization.

Bach et al. (2017) introduce $\max(\ell(\boldsymbol{x}), 0)$ as a soft version of the hard linear constraint $\ell(\boldsymbol{x}) \leq 0$. DL2 uses the same approach for this type of constraint. However, DL2 is more general: it allows arbitrary boolean combinations of possibly non-linear constraints over variables with domain $\mathbb{R}$, while Bach et al. (2017) only consider conjunctions of linear constraints over variables with domain $[0, 1]$.

4. Constrained Neural Networks

In this section, we present our method for training neural networks with constraints.

Training with Constraints For a distribution $\mathcal{D}$, a set $\mathbb{A}$ and a constraint φ, we consider the maximization problem

$$\arg\max_{\theta} \Pr_{\boldsymbol{x}\sim\mathcal{D}} \left[\forall \boldsymbol{z} \in \mathbb{A}.\ \varphi(\boldsymbol{x}, \boldsymbol{z}, \theta)\right].$$

In other words, we want to find neural network weights θ such that the probability that the constraint $\varphi(\boldsymbol{x}, \boldsymbol{z}, \theta)$ is satisfied for all $\boldsymbol{z}$ is as large as possible. (We write $\varphi(\boldsymbol{x}, \boldsymbol{z}, \theta)$ for $\varphi(\boldsymbol{y}, \theta)$, where $\boldsymbol{y}$ is the concatenation of $\boldsymbol{x}$ and $\boldsymbol{z}$.) In practice, $\boldsymbol{x}$ usually describes one or more independent data set samples, and the set $\mathbb{A}$ is $\mathbb{R}^n$ for some n.

Formulation as Nested Optimization We can equivalently minimize the probability that there is a counterexample violating the constraint φ:

$$\arg\min_{\theta} \Pr_{\boldsymbol{x}\sim\mathcal{D}} \left[\exists \boldsymbol{z} \in \mathbb{A}.\ \neg\varphi(\boldsymbol{x}, \boldsymbol{z}, \theta)\right].$$

If it exists, a counterexample can be found by maximizing the indicator function $[\neg\varphi(\boldsymbol{x}, \boldsymbol{z}, \theta)]$:

$$\arg\min_{\theta} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}} \left[\max_{\boldsymbol{z}\in\mathbb{A}} [\neg\varphi(\boldsymbol{x}, \boldsymbol{z}, \theta)]\right].$$

Assume that for each given $\boldsymbol{x}$ and θ, we can find an optimal solution $\boldsymbol{z}^*(\boldsymbol{x}, \theta)$ to the inner maximization problem:

$$\boldsymbol{z}^*(\boldsymbol{x}, \theta) = \arg\max_{\boldsymbol{z}\in\mathbb{A}} [\neg\varphi(\boldsymbol{x}, \boldsymbol{z}, \theta)]. \tag{1}$$

Using $\boldsymbol{z}^*(\boldsymbol{x}, \theta)$, we can rephrase the original problem as

$$\arg\max_{\theta} \Pr_{\boldsymbol{x}\sim\mathcal{D}} [\varphi(\boldsymbol{x}, \boldsymbol{z}^*(\boldsymbol{x}, \theta), \theta)]. \tag{2}$$

The advantage of this formulation is that it splits the problem into two subproblems. The optimization can now be seen as a game between an optimizer that proposes neural network weights (2) and an adversary that finds counterexamples (1).

Approximate Solutions based on Loss We approximately solve (1) and (2) by translating logical constraints into loss functions, as shown in Section 3. Using Theorem 1, we can show that the inner maximization problem (1) can be solved by *minimizing* the translated loss $\mathcal{L}(\neg\varphi)$:

$$\boldsymbol{z}^*(\boldsymbol{x}, \theta) = \arg\min_{\boldsymbol{z}\in\mathbb{A}} \mathcal{L}(\neg\varphi)(\boldsymbol{x}, \boldsymbol{z}, \theta). \tag{3}$$

For a given $\boldsymbol{z}^*(\boldsymbol{x}, \theta)$, we approximate the outer optimization problem (2) as

$$\arg\min_{\theta} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{T}} [\mathcal{L}(\varphi)(\boldsymbol{x}, \boldsymbol{z}^*(\boldsymbol{x}, \theta), \theta)]. \tag{4}$$

Here, $\boldsymbol{x}$ is an uniform sample from the training set $\mathcal{T}$, which approximates $\mathcal{D}$. We optimize the loss using Adam (Kingma & Ba, 2015).

Constrained Optimization The loss in (3) can sometimes be difficult to optimize. To illustrate, assume that the random samples are input-label pairs $(\boldsymbol{x}, y)$ and consider the constraint

$$\varphi((\boldsymbol{x}, y), \boldsymbol{z}, \theta) = \|\boldsymbol{x} - \boldsymbol{z}\|_\infty \leq \epsilon \implies \text{logit}_\theta(\boldsymbol{z})_y > \delta,$$

where we write $\text{logit}_\theta(\boldsymbol{z})_y$ for the pre-softmax activation of the neural network parameterized by weights θ for class y when given input z. Our translation of this constraint to a loss produces $\mathcal{L}(\neg\varphi)((\boldsymbol{x}, y), \boldsymbol{z}, \theta)$ equal to

$$\max(0, ||\boldsymbol{x} - \boldsymbol{z}||_\infty - \epsilon) + \max(0, \text{logit}_\theta(\boldsymbol{z})_y - \delta).$$

This function is difficult to minimize because the magnitude of the two terms is different. This causes first-order methods to optimize only a single term in an overly greedy

```
find i[32, 32, 3]
where i in [0, 255],
    class(NN(i)) = 9,
    ‖i - deer‖∞ < 25,
    ‖i - deer‖∞ > 5
```

(a) Adversarial example.

```
find i[32, 32, 3]
where i in [0, 255],
    class(NN(i)) = 4,
    ‖i - deer‖∞ < 25,
    NN(i).l₁[0, 1, 1, 31] = 0
```

(b) Neuron deactivated.

```
find i[28, 28]
where i in [0, 1],
    class(NN1(i)) = 8,
    class(NN2(i)) = 9,
    i[0:9,:] = nine[0:9,:]
```

(c) Diffing networks.

Figure 1. DL2 queries enable to declaratively search for inputs satisfying constraints over networks.

manner, as reported by Carlini & Wagner (2017). However, some constraints have a closed-form analytical solution, e.g., $\max(0, ||\boldsymbol{x} - \boldsymbol{z}||_\infty - \epsilon)$ can be minimized by projecting $\boldsymbol{z}$ into the L_∞ ball with radius ϵ around $\boldsymbol{x}$. We identify logical constraints which restrict the variables $\boldsymbol{z}$ to convex sets that have an efficient algorithm for projection, e.g., line segments, L_2, L_∞ or L_1 balls (Duchi et al., 2008). We exclude such constraints from $\neg\varphi$ and add them as constraints of the optimization. We thus rewrite (3) as

$$\boldsymbol{z}^*(\boldsymbol{x}, \theta) = \underset{\boldsymbol{z} \in \mathbb{B}}{\arg\min}\, \mathcal{L}(\psi)(\boldsymbol{x}, \boldsymbol{z}, \theta). \tag{5}$$

Here, $\mathbb{B}$ is the respective convex set and ψ is $\neg\varphi$ without the respective constraints. To solve (5) in our setting, we employ Projected Gradient Descent (PGD) which has been shown to have strong performance in the case of adversarial training with L_∞ balls (Madry et al., 2018).

Algorithm 1 Training with constraints.

```
Input: Training set $\mathcal{T}$, network parameters $\theta$,
       constraint $\varphi(\boldsymbol{x}, \boldsymbol{z}, \theta)$
Extract convex set $\mathbb{B}$ and constraint $\psi$ from constraint $\varphi$.
for epoch = 1 to $n_{\text{epochs}}$ do
    Sample mini-batch of vectors $\boldsymbol{x} \sim \mathcal{T}$.
    Using PGD, compute
    $\boldsymbol{z}^* \approx \arg\min_{\boldsymbol{z} \in \mathbb{B}} \mathcal{L}(\psi)(\boldsymbol{x}, \boldsymbol{z}, \theta)$ for each $\boldsymbol{x}$.
    Perform GD update with $\nabla_\theta \mathcal{L}(\varphi)(\boldsymbol{x}, \boldsymbol{z}^*, \theta)$.
end for
```

Training procedure Algorithm 1 shows our training procedure. We first form a mini-batch of random samples from the training set $\mathcal{T}$. And then an approximate solution for (3) using the formulation in (5). This solution is given to the optimizer, which improves the network weights to optimize (4). Note that if $\boldsymbol{z}$ is 0-dimensional, the adversary becomes trivial and the loss can be computed directly.

5. Querying Networks

Building on DL2, we design a declarative language for querying neural networks. Interestingly, many questions investigated by prior work can now be phrased as DL2 queries: neurons responsible for a prediction (Olah et al., 2018), inputs that differentiate networks (Pei et al., 2017), and adversarial examples (e.g., Szegedy et al., 2014). We support the following class of queries:

$$\begin{aligned}
&\textbf{find } \boldsymbol{z}_1[m_1], \ldots, \boldsymbol{z}_k[m_k] \\
&\textbf{where } \varphi(\boldsymbol{z}^1, \ldots, \boldsymbol{z}^k) \\
&[\textbf{init } \boldsymbol{z}_1 = \boldsymbol{c}_1, \ldots, \boldsymbol{z}_k = \boldsymbol{c}_k] \\
&[\textbf{return } \boldsymbol{t}(\bar{z})]
\end{aligned}$$

Here, the **find** clause defines a number of variables with a shape given in parentheses. The **where** clause defines the constraint. The **init** clause defines initial values for (part or all of) the variables, and the **return** clause defines a target term to compute at the end of search; if missing, $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_k$ are returned. Neural networks and constants can be defined outside of the queries. They are then automatically loaded at query time. The constraints are equally expressive as the fragment described in Section 3, but our language supports additional features for improved convenience. For example, the user can specify and manipulate tensors. In queries, we write comma (`,`) for conjunction ($\wedge$); **in** for box constraints and **class** for returning the network's output label on a given input, i.e., the class with the maximum probability.

The constraint **class** `(NN(x)) = y` is equivalent to $\bigwedge_{i=1, i \neq \mathtt{y}}^{k} p(x)_i < p(x)_{\mathtt{y}}$, where $p(x)_i$ denotes the probability which `NN` returns for class i out of k classes.

Examples Figure 1 shows a few interesting queries. The first two are defined over networks trained for CIFAR-10, while the last is for MNIST. The first query attempts to find an adversarial example `i` of shape $(32, 32, 3)$, classified as a truck (class 9) where the distance of `i` to a given deer image (`deer`) is between 5 and 25, with respect to the infinity norm. The second query attempts to find an `i` which is classified as a deer, while a specific neuron in a convolutional layer (accessed by indices) is deactivated. The last query attempts to find an `i` classified differently by two networks where part of `i` is fixed to pixels of the image `nine`.

Solving Queries As with training, we compile the constraints into a loss. The loss is minimized either until it reaches zero or until timeout. Unlike training, we perform optimization with L-BFGS-B, which is slower but more sophisticated than standard gradient descent. We can afford this because the optimized loss function does not depend on an entire batch of training samples.

Query Optimization Here, we discuss how loss compilation can be optimized for L-BFGS-B. While our translation is defined for arbitrarily large constraints, in general, it is hard to optimize for a loss with many terms. Thus, we make the loss smaller by extracting box constraints out of the expression. The loss is then compiled from the remaining constraints. Extracted box constraints are passed to the L-BFGS-B solver which is then used to find the minimum of the loss. This enables us to exclude a dominant part of φ from the loss, making our loss amenable to optimization.

To illustrate this benefit, consider the query in Figure 1a. Its box constraint, `i` **in** `[0,255]`, is syntactic sugar for a conjunction with $2 \cdot 32 \cdot 32 \cdot 3 = 6,144$ atomic constraints (two for each variable, i.e., for every index j, we have $i_j \geq 0$ and $i_j \leq 255$). In contrast, the second constraint consists of 9 atomic constraints (one for each possible class different from 9), and the third and fourth constraints are already atomic. By excluding the box constraints from the loss, the obtained loss consists of only 11 terms, making it more amenable to gradient-based optimization. If a solution is not found after a certain time, we restart L-BFGS-B and reinitialize the variables using MCMC sampling.

To prevent numerical issues, constraints using the **class** keyword are implemented using logits rather than probabilities.

Further, we translate L_∞-constraints such as $\|\boldsymbol{a}\|_\infty < b$ into $\bigwedge_i |\boldsymbol{a}_i| < b$. This type of constraint often appears in adversarial example queries. The original formulation only provides gradient information for the largest element of $\boldsymbol{a}$, while the new one provides information to potentially all. This allows for much faster optimization of these types of queries. The ultimate loss is very similar to the one in (Carlini & Wagner, 2017), which employs a similar trick.

6. Experimental Evaluation

We now present a thorough experimental evaluation showing the effectiveness of DL2 for querying and training neural networks with logical constraints. Our system is implemented in PyTorch (Paszke et al., 2017) and evaluated on an Nvidia GTX 1080 Ti and Intel Core i7-7700K with 4.20 GHz.

6.1. Training with DL2

We evaluated DL2 on various tasks (supervised, semi-supervised and unsupervised learning) across four datasets: MNIST, FASHION (Xiao et al., 2017), CIFAR-10, and CIFAR-100 (Krizhevsky & Hinton, 2009). In all experiments, we added a cross-entropy term to our loss, to optimize for high prediction accuracy. For each experiment, we describe the additional logical constraints.

Table 1. Semi-supervised training.

Method	Accuracy (%)	Constraint Accuracy (%)
Baseline	45.20	74.80
DL2	**47.40**	**90.85**

Table 2. Unsupervised learning.

Approach	MSE
Supervised (regression)	*0.0895*
Unsupervised (baseline)	0.6680
Unsupervised (with DL2)	**0.1030**

Semi-supervised Learning For semi-supervised learning, we focus on the CIFAR-100 dataset, and split the training set into labeled, unlabeled and validation set in the ratio 20/60/20. In the spirit of the experiments of Xu et al. (2018), we consider the constraint which requires that the probabilities of *groups of classes* have either very high probability or very low probability. A group consists of classes of a similar type (e.g., the classes *baby*, *boy*, *girl*, *man*, and *woman* are part of the *people* group), and the group's probability is the sum of its classes' probabilities. Formally, our constraint consists of 20 groups and its structure is

$$(p_{\text{people}} < \epsilon \vee p_{\text{people}} > 1-\epsilon) \wedge \ldots \wedge (p_{\text{trees}} < \epsilon \vee p_{\text{trees}} > 1-\epsilon),$$

for a small ϵ. Each group includes classes that belong to a similar family, for example $p_{\text{people}} = p_{\text{baby}} + p_{\text{boy}} + p_{\text{girl}} + p_{\text{man}} + p_{\text{woman}}$. The intuition is that we want to constrain the neural network to put (almost) all probability mass into one group (e.g. people). Note that this constraint is beyond the reach of prior works (for reasons discussed in Section 3.1).

We train a network with the normal cross-entropy loss on the labeled data and the DL2 encoding of the constraint on the unlabeled data (weighted by a parameter λ). For further implementation details, see Appendix A.

Table 1 shows the prediction and constraint accuracies on the test set of our approach and a baseline that trains only with cross-entropy. We found that training with our loss yields a significantly higher constraint accuracy. Further, the information from the DL2 constraint on the unlabeled data allows the model to obtain 2.2% higher prediction accuracy.

Unsupervised Learning We next consider a regression task in an unsupervised setting. The task is training an MLP (multilayer perceptron) to predict the minimum distance from a source node to every node in an unweighted graph $G = (V, E)$. Assuming the source node is 0, the minimum distance is a function with certain properties, which can be

Table 3. Supervised learning, P/C is prediction/constraint accuracy.

		MNIST		FASHION		CIFAR-10	
		Baseline	DL2	Baseline	DL2	Baseline	DL2
Robustness$^{\text{T}}$	P	99.57	98.26	92.33	88.13	92.11	89.46
	C	97.90	99.41	94.50	98.36	91.39	95.46
Robustness$^{\text{G}}$	P	99.56	99.60	92.56	92.93	92.18	92.18
	C	00.00	99.96	00.00	99.91	00.00	99.95
Lipschitz$^{\text{T}}$	P	99.54	97.90	92.66	89.08	91.39	90.57
	C	09.65	99.90	05.62	99.78	05.00	99.41
Lipschitz$^{\text{G}}$	P	99.62	99.56	92.79	81.10	92.11	92.05
	C	00.00	100.00	00.00	98.64	00.00	99.55
C-similarity$^{\text{T}}$	P	-	-	-	-	91.69	91.91
	C	-	-	-	-	93.67	99.68
C-similarity$^{\text{G}}$	P	-	-	-	-	91.89	88.99
	C	-	-	-	-	50.47	99.42
Segment$^{\text{G}}$	P	98.27	97.60	88.58	87.77	-	-
	C	17.58	42.19	21.24	53.16	-	-

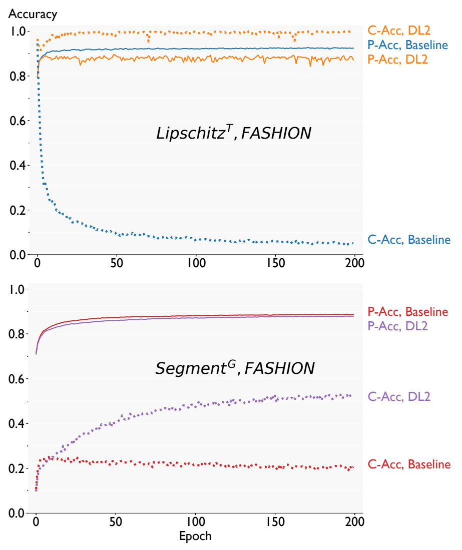


Figure 2. P/C Accuracy during training.

encoded into the following logical constraint:

$$d(0)=0 \wedge \bigwedge_{v\in V} d(v)\geq 0 \wedge \left(\bigvee_{\substack{(v,v')\in E\\ v'\neq 0}} d(v')=d(v)+1 \right) \wedge \left(\bigwedge_{(v,v')\in E} d(v')\leq d(v)+1 \right).$$

We train the model in an unsupervised fashion with the DL2 loss. We generate random connected graphs with 15 vertices and split them into training (300), validation (150) and test (150) sets. As an unsupervised baseline, we consider a model which always predicts $d(v) = 1$. We also train a supervised model with the mean squared error (MSE) loss. Table 2 shows the MSE of each approach. Results indicate that our approach obtains an error very close to the supervised model, despite not using any labels.

Supervised Learning We consider two types of constraints for supervised learning: *global constraints*, which have (possibly more than one) universally quantified variable $\boldsymbol{z}$, and *training set constraints*, where all variables refer to samples from the training set (no quantified variable $\boldsymbol{z}$). Note that no prior work applies to *global constraints*, in general. Furthermore, because of limitations of their encoding, as explained in Section 3.1, prior work is not able to handle the complex *training set constraints* considered in our experiments (e.g., comparisons between output activations). We write random samples as $\boldsymbol{x}$ and y, to denote inputs from the training set with their corresponding label.

For local robustness (Szegedy et al., 2014), the training set constraint says that if two random inputs from the dataset are close (their distance is less than a given ϵ_1, with respect to the L_2 norm), then the KL-divergence of their respective output activations is smaller than ϵ_2:

$$\|\boldsymbol{x}-\boldsymbol{x}'\|_2 < \epsilon_1 \implies \mathrm{KL}(p_\theta(\boldsymbol{x})||p_\theta(\boldsymbol{x}')) < \epsilon_2 \qquad (\text{Robustness}^{\text{T}})$$

Our global robustness constraint requires that for any input $\boldsymbol{x}$ with a classification y, inputs in its ϵ neighborhood which are valid images (pixels are between 0 and 1), have a high probability for y. For numerical stability, instead of directly checking the probability, we check that the corresponding log-probability is larger than a given threshold δ:

$$\forall \boldsymbol{z} \in B_\epsilon(\boldsymbol{x}) \cap [0,1]^d.\ \log p_\theta(\boldsymbol{z})_y > \delta \qquad (\text{Robustness}^{\text{G}})$$

Here, $B_\epsilon(\boldsymbol{x})$ is an L_∞-ball around $\boldsymbol{x}$ with radius ϵ.

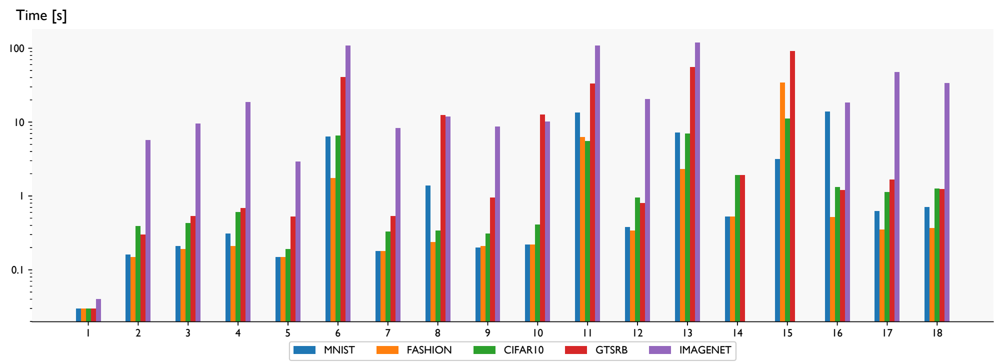
Similarly, we have two definitions for the Lipschitz condition. The training set constraint requires that for random input pairs from the training set, the distance between their output logits is less than the Lipschitz constant L times the distance between the inputs:

$$\|\mathrm{logit}_\theta(\boldsymbol{x}) - \mathrm{logit}_\theta(\boldsymbol{x}')\|_2 < L\|\boldsymbol{x}-\boldsymbol{x}'\|_2 \qquad (\text{Lipschitz}^{\text{T}})$$

The global version poses the same constraint for valid images in the neighborhood of $\boldsymbol{x}$ and $\boldsymbol{x}'$:

$$\forall \boldsymbol{z} \in B_\epsilon(\boldsymbol{x}) \cap [0,1]^d, \boldsymbol{z}' \in B_\epsilon(\boldsymbol{x}') \cap [0,1]^d.$$
$$\|\mathrm{logit}_\theta(\boldsymbol{z}) - \mathrm{logit}_\theta(\boldsymbol{z}')\|_2 < L\|\boldsymbol{z}-\boldsymbol{z}'\|_2 \qquad (\text{Lipschitz}^{\text{G}})$$

We also consider *C-similarity*, a training set constraint providing additional domain knowledge to CIFAR-10 networks. The constraint requires that inputs classified as a *car* have a



```
find i[100]
where i in [-1,1],
class(N1(G(i)))=c1,
N1(G(i)).p[c1]> 0.3,
class(N2(G(i)))=c2,
N2(G(i)).p[c2]> 0.3
```

Figure 3. (left) The average execution time per query in seconds. Unsuccessful queries reached a timeout of 120s. (right) The template for query 15. N_1, N_2 denote a classifier neural network, c_1, c_2 are classes ($c_1 \neq c_2$), and G is a generator neural network.

higher activation for the label *truck* than for the label *dog*:

$$y = \text{car} \implies \text{logit}_\theta(\boldsymbol{x})_{\text{truck}} > \text{logit}_\theta(\boldsymbol{x})_{\text{dog}} + \delta \qquad (\text{C-similarity}^{\text{T}})$$

The global constraint is similar but applied for valid images in the ϵ-neighborhood of $\boldsymbol{x}$:

$$\forall \boldsymbol{z} \in B_\epsilon(\boldsymbol{x}) \cap [0,1]^d.\, y = \text{car} \implies \text{logit}_\theta(\boldsymbol{z})_{\text{truck}} > \text{logit}_\theta(\boldsymbol{z})_{\text{dog}} + \delta \qquad (\text{C-similarity}^{\text{G}})$$

Finally, we consider a *Segment* constraint which requires that if an input $\boldsymbol{z}$ is on the line between two inputs $\boldsymbol{x}$ and $\boldsymbol{x}'$ with interpolation parameter $\lambda \in [0,1]$, then its output probabilities are on the line between the output probabilities with the same interpolation parameter λ:

$$\forall \boldsymbol{z}.\, \boldsymbol{z} = \lambda \cdot \boldsymbol{x} + (1-\lambda) \cdot \boldsymbol{x}' \implies \mathcal{H}(p_\theta(\boldsymbol{z}), \lambda \cdot p_\theta(\boldsymbol{x}) + (1-\lambda) \cdot p_\theta(\boldsymbol{x}')) < \delta \quad (\text{Segment}^{\text{G}})$$

Here, $\mathcal{H}$ denotes cross-entropy. We note that the constraints for C-similarity and Segment$^{\text{G}}$ are shortened versions of the evaluated ones. The full constraints are provided in Appendix A.

Table 3 shows the prediction accuracy (P) and the constraint accuracy (C) when training with (i) cross-entropy only (CE) and (ii) CE and the constraint. Results indicate that DL2 can significantly improve constraint accuracy (from 0% to 99% for Lipschitz$^{\text{G}}$), while prediction accuracy slightly decreases. This decrease is expected in light of a recent work by Tsipras et al. (2018), which shows that adversarial robustness comes with decreased prediction accuracy. We suspect that we observe a similar phenomenon here. Figure 2 shows how the prediction and constraint accuracies change during training, for two constraints, on the FASHION dataset.

In terms of run-time overhead, training set constraints (T) only incur a small overhead per epoch (roughly 2x), while global constraints (G) incur a higher overhead (roughly 10x) as the projection steps can be expensive. Detailed run-times are provided in Table 4, Appendix A.

6.2. Querying with DL2

Next, we evaluate DL2 on the task of querying with constraints. We considered five image datasets, each with different well-established neural network architectures. For each, we considered at least two classifiers; and for some a generator and a discriminator (trained using GAN, Goodfellow et al., 2014) — Table 6, Appendix B, provides details about these networks. Our benchmark consists of 18 template queries (provided in Appendix B), which are instantiated with different networks, classes, and images. Figure 3 (right) shows a template query, which attempts to search for an input to a generator G such that its output is classified into two different classes $\mathbf{c}_1$ and $\mathbf{c}_2$ by classifiers N_1 and N_2. For each dataset, we instantiate each query template into 10 queries. The queries run until a correct solution is found or until a 2 minute timeout is reached. Figure 3 (left) shows the average run-times of each query template and a dataset. The results show that our system usually finds solutions, and usually completes within 40 seconds. It is unknown whether queries that were not solved in fact do have a solution. We conclude that successful executions terminate relatively quickly and that DL2 scales well to multiple large networks (e.g., for ImageNet). Appendix B and Appendix C provide further details and experiments.

7. Conclusion

We presented DL2, a method and system for training and querying neural networks with logical constraints. DL2 supports expressive logical constraints and provides translation rules into a differentiable-almost-everywhere loss, which is zero exactly for those inputs that satisfy the constraints. To make training tractable, we handle input constraints which capture convex sets through PGD. We also introduce a declarative language for querying networks utilzing the translation from logic to loss. Experimental results indicate that DL2 can be used effectively for both training and querying neural networks with additional constraints.

References

Bach, S. H., Broecheler, M., Huang, B., and Getoor, L. Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 18:109:1–109:67, 2017. URL http://jmlr.org/papers/v18/15-631.html.

Balan, R., Singh, M., and Zou, D. Lipschitz properties for deep convolutional networks. *CoRR*, abs/1701.05217, 2017. URL http://arxiv.org/abs/1701.05217.

Besold, T. R., d'Avila Garcez, A. S., Bader, S., Bowman, H., Domingos, P. M., Hitzler, P., Kühnberger, K., Lamb, L. C., Lowd, D., Lima, P. M. V., de Penning, L., Pinkas, G., Poon, H., and Zaverucha, G. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR*, abs/1711.03902, 2017. URL http://arxiv.org/abs/1711.03902.

Carlini, N. and Wagner, D. A. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pp. 39–57. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.49. URL https://doi.org/10.1109/SP.2017.49.

Demeester, T., Rocktäschel, T., and Riedel, S. Lifted rule injection for relation embeddings. In Su, J., Carreras, X., and Duh, K. (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pp. 1389–1399. The Association for Computational Linguistics, 2016. URL http://aclweb.org/anthology/D/D16/D16-1146.pdf.

Duchi, J. C., Shalev-Shwartz, S., Singer, Y., and Chandra, T. Efficient projections onto the l_1-ball for learning in high dimensions. In Cohen, W. W., McCallum, A., and Roweis, S. T. (eds.), *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pp. 272–279. ACM, 2008. doi: 10.1145/1390156.1390191. URL https://doi.org/10.1145/1390156.1390191.

Evans, R. and Grefenstette, E. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018. doi: 10.1613/jair.5714. URL https://doi.org/10.1613/jair.5714.

Fu, Z. and Su, Z. Xsat: A fast floating-point satisfiability solver. In Chaudhuri, S. and Farzan, A. (eds.), *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, volume 9780 of *Lecture Notes in Computer Science*, pp. 187–209. Springer, 2016. doi: 10.1007/978-3-319-41540-6_11. URL https://doi.org/10.1007/978-3-319-41540-6_11.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 2672–2680, 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

Gouk, H., Frank, E., Pfahringer, B., and Cree, M. J. Regularisation of neural networks by enforcing lipschitz continuity. *CoRR*, abs/1804.04368, 2018. URL http://arxiv.org/abs/1804.04368.

Guo, S., Wang, Q., Wang, L., Wang, B., and Guo, L. Jointly embedding knowledge graphs and logical rules. In Su, J., Carreras, X., and Duh, K. (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pp. 192–202. The Association for Computational Linguistics, 2016. URL http://aclweb.org/anthology/D/D16/D16-1019.pdf.

Guu, K., Miller, J., and Liang, P. Traversing knowledge graphs in vector space. In Màrquez, L., Callison-Burch, C., Su, J., Pighin, D., and Marton, Y. (eds.), *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pp. 318–327. The Association for Computational Linguistics, 2015. URL http://aclweb.org/anthology/D/D15/D15-1038.pdf.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90.

Hu, Z., Ma, X., Liu, Z., Hovy, E. H., and Xing, E. P. Harnessing deep neural networks with logic rules. In *Proceedings* *of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics, 2016. URL http://aclweb.org/anthology/P/P16/P16-1228.pdf.

Kimmig, A., Bach, S., Broecheler, M., Huang, B., and Getoor, L. A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*, pp. 1–4, 2012. URL http://stephenbach.net/files/kimmig-probprog12.pdf.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. 2015. URL http://arxiv.org/abs/1412.6980.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, 2009.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

Márquez-Neila, P., Salzmann, M., and Fua, P. Imposing hard constraints on deep networks: Promises and limitations. *CoRR*, abs/1706.02025, 2017. URL http://arxiv.org/abs/1706.02025.

Minervini, P. and Riedel, S. Adversarially regularising neural NLI models to integrate logical background knowledge. In Korhonen, A. and Titov, I. (eds.), *Proceedings of the 22nd Conference on Computational Natural Language Learning, CoNLL 2018, Brussels, Belgium, October 31 - November 1, 2018*, pp. 65–74. Association for Computational Linguistics, 2018. URL https://aclanthology.info/papers/K18-1007/k18-1007.

Minervini, P., Demeester, T., Rocktäschel, T., and Riedel, S. Adversarial sets for regularising neural link predictors. 2017. URL http://auai.org/uai2017/proceedings/papers/306.pdf.

Odena, A., Olah, C., and Shlens, J. Conditional image synthesis with auxiliary classifier gans. 70:2642–2651, 2017. URL http://proceedings.mlr.press/v70/odena17a.html.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. The building blocks of interpretability. *Distill*, 2018. doi: 10.23915/distill.00010. https://distill.pub/2018/building-blocks.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. Technical report, 2017.

Pathak, D., Krähenbühl, P., and Darrell, T. Constrained convolutional neural networks for weakly supervised segmentation. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pp. 1796–1804. IEEE Computer Society, 2015. doi: 10.1109/ICCV.2015.209. URL https://doi.org/10.1109/ICCV.2015.209.

Pei, K., Cao, Y., Yang, J., and Jana, S. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pp. 1–18, 2017. doi: 10.1145/3132747.3132785. URL http://doi.acm.org/10.1145/3132747.3132785.

Rocktäschel, T. and Riedel, S. End-to-end differentiable proving. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 3791–3803, 2017. URL http://papers.nips.cc/paper/6969-end-to-end-differentiable-proving.

Rocktäschel, T., Singh, S., and Riedel, S. Injecting logical background knowledge into embeddings for relation extraction. In Mihalcea, R., Chai, J. Y., and Sarkar, A. (eds.), *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pp. 1119–1129. The Association for Computational Linguistics, 2015. URL http://aclweb.org/anthology/N/N15/N15-1118.pdf.

Schawinski, K., Zhang, C., Zhang, H., Fowler, L., and Krishnan Santhanam, G. Galaxygan: Generative adversarial networks for recovery of galaxy features. *Astrophysics Source Code Library*, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

Song, Y., Shu, R., Kushman, N., and Ermon, S. Generative adversarial examples. *CoRR*, abs/1805.07894, 2018. URL http://arxiv.org/abs/1805.07894.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6199.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., and Madry, A. There is no free lunch in adversarial robustness (but there are unexpected benefits). *CoRR*, abs/1805.12152, 2018. URL http://arxiv.org/abs/1805.12152.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Xu, J., Zhang, Z., Friedman, T., Liang, Y., and den Broeck, G. V. A semantic loss function for deep learning with symbolic knowledge. In Dy, J. G. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 5498–5507. PMLR, 2018. URL http://proceedings.mlr.press/v80/xu18h.html.

Yang, F., Yang, Z., and Cohen, W. W. Differentiable learning of logical rules for knowledge base reasoning. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 2316–2325, 2017. URL http://papers.nips.cc/paper/6826-differentiable-learning-of-logical-rules-for-knowledge-base-reasoning.

Supplemental Material for DL2: Training and Querying Neural Networks with Logic

Here, we describe implementation details and hyperparameters for our experiments. In all experiments, we used $\xi = 1$, but we found the value not to have a large impact.

A. Details for Training Experiments

Supervised Learning For our experiments with supervised learning, we used a batch size of 128 and the Adam optimizer with learning rate 0.0001. All experiments were run for 200 epochs and the reported values in Table 3 are for the model with the highest constraint accuracy times prediction accuracy within the last 25 epochs. The average time for one epoch is given in Table 4. All other parameters are listed in Table 5. Additionally, for the CIFAR-10 experiments, we used data augmentation with random cropping and random horizontal flipping. Experiments with the Segment constraints were done by first embedding images in 40-dimensional space using PCA. In this lower dimensional space, it is sensible to consider linear interpolation between images which is not the case otherwise. This experiment was not performed for CIFAR-10 because we did not observe good prediction accuracy with baseline model using lower dimensional embedding. This is likely because the dimensionality of CIFAR-10's images is much higher than that of MNIST or FASHION.

We used ResNet-18 (He et al., 2016) for the experiments on CIFAR-10 and convolutional neural network (CNN) with 6 convolutional and 2 linear layers for MNIST and FASHION (trained with batchnorm after each convolutional layer). The layer dimensions of the CNN are (1, 32, 5x5) - (32, 32, 5x5) - (32, 64, 3x3) - (64, 64, 3x3) - (64, 128, 3x3) - (128, 128, 1x1) - 100 - 10 where (in, out, kernel-size) denotes a convolutional layer and a number n denotes a linear layer with n neurons.

We have implemented the Segment$^{\text{G}}$ constraint as

$$\begin{aligned}\forall \boldsymbol{z}.\ (&(\boldsymbol{z} = \lambda \cdot \boldsymbol{x} + (1-\lambda)\cdot \boldsymbol{x}')\\ &\wedge (\|\boldsymbol{x} - \boldsymbol{x}'\|_2 < \epsilon)\\ &\wedge (y \neq y')) \implies\\ &H(p_\theta(\boldsymbol{z}), \lambda \cdot p_\theta(\boldsymbol{x}) + (1-\lambda)\cdot p_\theta(\boldsymbol{x}')) < \delta.\end{aligned}$$

Which in addition to the constraint in the paper, only considers training samples that are close together ($\|\boldsymbol{x} - \boldsymbol{x}'\|_2 < \epsilon$) and that have different labels $(y \neq y')$.

For the C-Similarity constraint we use the formulation as in the body of the paper, but we not only apply it to the car-truck-dog labels, but also deer-horse-ship, plane-ship-frog, dog-cat-truck and cat-dog-car.

Semi-supervised Learning For this experiment, we use the VGG-16 architecture (Simonyan & Zisserman, 2014) and optimize the loss using Adam with learning rate 0.001 and a batch size of 128. For each labeled batch, we also sampled one unlabeled batch and combined the losses before back-propagating. We trained for 1600 epochs, although stable results were observed after around 400 epochs. We used $\lambda = 0.6$ as the weighting factor for the DL2 loss.

Unsupervised Learning Our model is the multilayer perceptron with $N \cdot N$ input neurons, three hidden layers with 1000 neurons each and an output layer of N neurons. N is the number of vertices in the graph, in our case 15. The input is the adjacency matrix of the graph and the output is the distance for each node. The network uses ReLU activations and dropout of 0.3 after each hidden layer. The network is optimized using Adam with learning rate 0.0001.

B. Additional Details for Section 6.2

Here, we provide statistics on the networks and queries used in the experiments of Section 6.2. Table 6 summarizes the networks that we used, their architecture, and accuracy. Each row shows the dataset, the type of the network (classifier, generator, or discriminator), the network signature, and the architecture of the network. For example, the first row describes a classifier that takes as input images of size 28×28 pixels, each ranging between 0–1, and returns a probability distribution over ten classes.

Figure 4 shows the query templates, where the template parts are colored in blue. In the queries, we use the following names for the template parts. Networks are named `N`, `N`$_1$`,` `N`$_2$ for classifiers, `G` for generator, and `D` for discriminator. A variable denoting an input image (e.g., `deer` in Figure 1a) is `var`. Classes are `c`, `c`$_1$`,` `c`$_2$ or `c`$_v$ to refer to the (known) class of a variable `var`. Masks are de-

Table 4. Average time per epoch in seconds for the supervised training experiments in Table 3.

	MNIST		**FASHION**		**CIFAR-10**	
	Baseline	DL2	Baseline	DL2	Baseline	DL2
Robustness^T	7.34	13.11	7.24	12.58	40.04	100.97
Robustness^G	14.23	866.19	13.9	533.41	40.78	418.62
Lipschitz^T	7.37	12.45	7.53	12.13	40.63	75.72
Lipschitz^G	12.2	67.79	7.58	401.4	40.02	287.64
C-similarity^T	-	-	-	-	40.13	83.85
C-similarity^G	-	-	-	-	39.87	423.82
Segment^G	15.16	101.55	13.5	30.62	-	-

Table 5. Hyperparameters used for supervised learning experiment

	MNIST, FASHION			CIFAR-10		
	λ	PGD Iterations	Params	λ	PGD Iterations	Params
Robustness^T	0.2	-	$\epsilon_1 = 7.8, \epsilon_2 = 2.9$	0.04	-	$\epsilon_1 = 13.8, \epsilon_2 = 0.9$
Robustness^G	0.2	50	$\epsilon = 0.3, \delta = 0.52$	0.1	7	$\epsilon = 0.3, \delta = 0.52$
Lipschitz^T	0.1	-	$L = 1.0$	0.1	-	$L = 1.0$
Lipschitz^G	0.2	50	$L_{\text{mnist}} = 0.1, L_{\text{fashion}} = 0.3, \epsilon = 0.3$	0.1	5	$L = 1.0, \epsilon = 0.3$
Classes^T	-	-	-	0.2	-	$\delta = 0.01$
Classes^G	-	-	-	0.2	10	$\delta = 1.0, \epsilon = 0.3$
Segment^G	0.01	5	$\epsilon = 100, \delta = 1.5$	-	-	-

noted by `mask` and their complementary mask by `nm`. Pixel constraints (e.g., `i` **in** `[0,255]`) are `pix_con`, and their range (e.g., `[0,255]`) is `range`. The shape of a variable (e.g., 28, 28 for MNIST) is `shape`, and a constant denoting a distance is `dist`.

The tested queries enable us to study various aspects of the DL2 system. The first query runs a neural network on a given input, which allows us to gauge the overheads of our system (without the optimizer). This includes parsing, query setup time and the call to PyTorch to run the neural network. Queries 2-4 look for inputs satisfying constraints without an **init** clause, while queries 5-11 look for adversarial examples. Queries 12-18 aim to find inputs classified differently by the two networks, whereas queries 14-15 leverage generators and discriminators for this task. Table 7 shows the average run-time and success rate of different queries.

C. Additional Query Experiments

Here, we provide further experiments to investigate the scalability and run-time behavior of DL2. For all experiments, we used the same hyperparameters as in Section 6.2.

Experiment I: Number of Variables We first study the run-time behavior of DL2 as a function of the number of variables. For this, we consider a simple toy query:

```
find i[c]
where 1000 < sum(i), sum(i) < 1001
return i
```

with different integer values for c. We execute this query for a wide range of c values – 10 times for each value – and we report the average run-time in Figure 5a. All runs completed successfully and returned a correct solution. We observe constant run-time behavior for up to 2^{13} variables, and a linear run-time in the number of variables afterwards.

Experiment II: Opposing Constraints We next study the impact of (almost) opposing constraints. For this, we consider another toy query:

```
find i[1]
where i[0] < −c ∨ c < i[0]
return i
```

for an integer c. This query requires optimizing two opposing terms until one of them is fulfilled. The larger c, the more opposed the two objectives are – in the extreme case, for $c \to \infty$, we obtain an unsatisfiable objective. All runs completed successfully and returned a correct solution. Figure 5b shows the average run-time over 10 runs for different values of c.

Experiment III: Scaling to Many Constraints Lastly, we study the scaling of DL2 as a function of the number of

Table 6. The datasets and networks used to evaluate DL2. The reported accuracy is top-1 accuracy and it was either computed by the TorchVision library ($^{\#}$), or by us (†).

Dataset	Type	Network	Architecture	Accuracy
MNIST	C	`M_NN1`: $[0,1]^{28\times 28} \mapsto [0,1]^{10}$	small CNN, PyTorch Examples	$0.990^{\dagger}$
	C	`M_NN2`: $[0,1]^{28\times 28} \mapsto [0,1]^{10}$	small FFNN, PyTorch Examples	$0.979^{\dagger}$
	G	`M_G`: $[-1,1]^{100} \mapsto [0,1]^{28\times 28}$	DC-GAN	-
	D	`M_D`: $[0,1]^{28\times 28} \mapsto [0,1]$	DC-GAN	-
	G	`M_ACGAN_G`: $[-1,1]^{100} \times \{0,\dots,9\} \mapsto [0,1]^{28\times 28}$	AC-GAN	-
	D	`M_ACGAN_D`: $[0,1]^{28\times 28} \mapsto [0,1] \times [0,1]^{10}$	AC-GAN	-
Fashion MNIST	C	`FM_NN1` : $[0,1]^{28\times 28} \mapsto [0,1]^{10}$	same as `M_NN1`	$0.876^{\dagger}$
	C	`FM_NN2` : $[0,1]^{28\times 28} \mapsto [0,1]^{10}$	same as `M_NN2`	$0.860^{\dagger}$
	G	`FM_G`: $[-1,1]^{100} \mapsto [0,1]^{28\times 28}$	DC-GAN	-
	D	`FM_D` : $[0,1]^{28\times 28} \mapsto [0,1]$	DC-GAN	-
CIFAR	C	`C_NN1` : $[0,1]^{32\times 32\times 3} \mapsto [0,1]^{10}$	VGG-16	$0.936^{\dagger}$
	C	`C_NN2` : $[0,1]^{32\times 32\times 3} \mapsto [0,1]^{10}$	Resnet-18	$0.950^{\dagger}$
	G	`C_G` : $[-1,1]^{100} \mapsto [0,1]^{32\times 32\times 3}$	DC-GAN	-
	D	`C_D` : $[0,1]^{32\times 32\times 3} \mapsto [0,1]$	DC-GAN	-
GTSRB	C	`G_NN1` : $[0,1]^{32\times 32\times 3} \mapsto [0,1]^{10}$	VGG-16	$0.985^{\dagger}$
	C	`G_NN2` : $[0,1]^{32\times 32\times 3} \mapsto [0,1]^{10}$	Resnet-18	$0.995^{\dagger}$
	G	`G_G` : $[-1,1]^{100} \mapsto [0,1]^{32\times 32\times 3}$	DC-GAN	-
	D	`G_D` : $[0,1]^{32\times 32\times 3} \mapsto [0,1]$	DC-GAN	-
ImageNet	C	`I_V16` : $[0,1]^{224\times 224\times 3} \mapsto [0,1]^{1000}$	VGG-16 from PytTorch Vision	$0.716^{\#}$
	C	`I_V19` : $[0,1]^{224\times 224\times 3} \mapsto [0,1]^{1000}$	VGG-19 from PytTorch Vision	$0.7248^{\#}$
	C	`I_R50` : $[0,1]^{224\times 224\times 3} \mapsto [0,1]^{1000}$	ResNet-50 from PytTorch Vision	$0.764^{\#}$

constraints. For this, we consider the following query which looks for an adversarial example:

```
find p[28, 28]
where class(M_NN1(clamp(p + M_nine,
                          0, 1))) = c
return clamp(p + M_nine, 0, 1)
```

The query looks for an adversarial perturbation p to a given image of a nine (`M_nine`) such that the resulting image gets classifies as class c. The query returns the found perturbation and the resulting image. The `clamp(I, a, b)` operation takes an input `I` and cuts its values such that they are between a and b.

Additionally, we impose constraints on the rows and columns of the image. For a row i, we want to enforce that the values of the perturbation vector are increasing from left to right:

```
p[i, 0] < p[i, 1],
p[i, 1] < p[i, 2],
p[i, 2] < p[i, 3], ...
```

For one row this yields 27 constraints. We further consider a similar constraint for a column j:

```
p[0, j] < p[1, j],
p[1, j] < p[2, j],
p[2, j] < p[3, j] ...
```

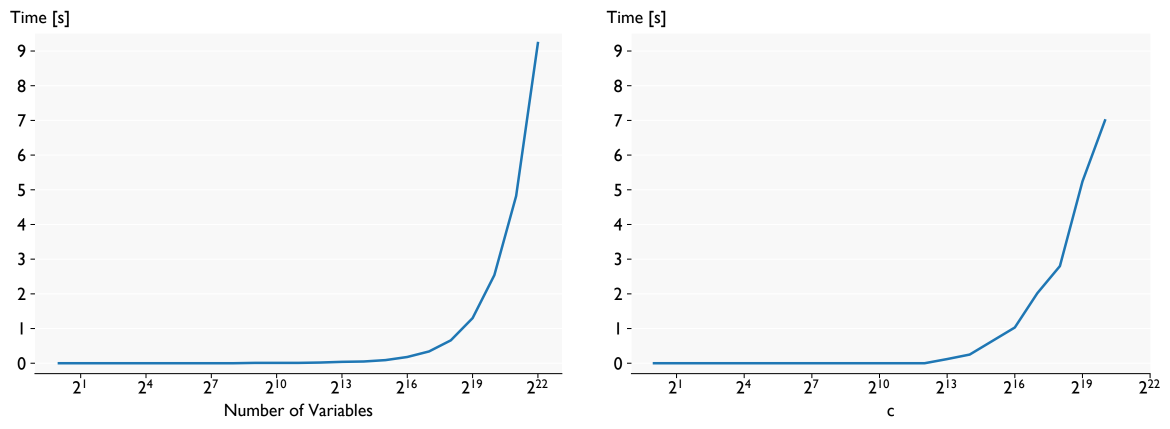
We apply these constraints on the first k rows and columns of the image independently and jointly. We vary the value of k, and for each we execute the query over all possible target classes $c \in \{0, \dots, 8\}$. We report the average time in Table 8. Most queries could be solved. For **Row & Column constraints** where $k = 20$ and $k = 28$, only 6 out of 9 and 1 out of 9 could be solved receptively. These queries hit the 300 s timeout. Figure 6 shows a resulting image.

	Query
1	**eval** N(var)
2	**find** i[shape] **where** **c**(N(i))=c
3	**find** i[shape] **where** **c**(N(i))=c, pix_con
4	**find** i[shape] **where** **c**(N(i))=c, N(i).p[c] > 0.8, pix_con
5	**find** i[shape] **where** **c**(N(i))=c **init** i=var
6	**find** i[shape] **where** **c**(N(i))=c, pix_con, $\|$i - var$\|_\infty$ < dist **init** i=var
7	**find** i[shape] **where** **c**(N(i))=c, pix_con **init** i=var
8	**find** i[shape] **where** **c**(N(i))=c, i[mask] **in** range, i[nm]=var[nm] **init** i=var
9	**find** i[shape] **where** **c**(N(i))=c, pix_con, N(i).p[c] > 0.8 **init** i=var
10	**find** i[shape] **where** **c**(N(i))=c, pix_con, N(i).p[c] > 0.8, N(i).p[c_v] < 0.1 **init** i=var
11	**find** i[shape] **where** **c**(N(i))=c, pix_con, N(i).p[c] > 0.8, N(i).p[c_v] < 0.1, $\|i-$var$\|_\infty$ < dist **init** i=var
12	**find** i[shape] **where** pix_con, **c**(N_2(i))=c_2, **c**(N_1(i))=c_1
13	**find** i[shape] **where** pix_con, **c**(N_2(i))=c, $\|$i - var$\|_\infty$ < dist, **c**(N_1(i))=c_v, **init** i=var
14	**find** i[shape] **where** **c**(N_1(i))=c_1, **c**(N_2(i))=c_2, N_1(i).p[c_1] > 0.5, N_2(i).p[c_1] < 0.1, N_2(i).p[c_2] > 0.5, N_1(i).p[c_2] < 0.1, pix_con,D(i) < 0.1
15	**find** i[100] **where** i **in** [-1,1], **c**(N_1(G(i)))=c_1, N_1(G(i)).p[c_1]> 0.3, **c**(N_2(G(i)))=c_2, N_2(G(i)).p[c_2]> 0.3
16	**find** i[shape] **where** **c**(N_1(i))=c_v, **c**(N_2(i))=c, i[mask] **in** range, i[nm]=var[nm] **init** i=var
17	**find** i[shape] **where** **c**(N_1(i))=c_1, **c**(N_2(i))=c_2, N_1(i).p[c_1] > 0.5, N_1(i).p[c_2] < 0.1, pix_con
18	**find** i[shape] **where** **c**(N_1(i))=c_1, **c**(N_2(i))=c_2, N_1(i).p[c_1] > 0.6, N_1(i).p[c_2] < 0.1, N_2(i).p[c_2] > 0.6, N_2(i).p[c_1] < 0.1, pix_con

Figure 4. The template queries used to evaluate DL2. N_1, N_2 denote classifier neural networks, var an example input from the corresponding dataset, c, c_1, c_2 are classes and $c_1 \neq c_2$, pix_con is a box constraint for pixels to be in the proper input range, D is a discriminator neural network and G is a generator neural network.

Table 7. Results for queries: (#✓) number of completed instances (out of 10), ⏱ is the average running time in seconds, and ⏱✓the average running time of successful runs (in seconds).

	MNIST			FASHION			CIFAR-10			GTSRB			ImageNet		
Nr.	#✓	⏱	⏱✓	#✓	⏱	⏱✓	#✓	⏱	⏱✓	#✓	⏱	⏱✓	#✓	⏱	⏱✓
1	10	0.03	0.03	10	0.03	0.03	10	0.03	0.03	10	0.03	0.03	10	0.04	0.04
2	10	0.16	0.16	10	0.15	0.15	10	0.39	0.39	10	0.30	0.30	10	5.71	5.71
3	10	0.21	0.21	10	0.19	0.19	10	0.43	0.43	10	0.54	0.54	10	9.58	9.58
4	10	0.31	0.31	10	0.21	0.21	10	0.61	0.61	10	0.69	0.69	10	18.81	18.81
5	10	0.15	0.15	10	0.15	0.15	10	0.19	0.19	10	0.53	0.53	10	2.95	2.95
6	10	6.38	6.38	10	1.76	1.76	10	6.59	6.59	9	40.94	32.15	1	109.20	12.01
7	10	0.18	0.18	10	0.18	0.18	10	0.33	0.33	10	0.54	0.54	10	8.30	8.30
8	10	1.39	1.39	10	0.24	0.24	10	0.34	0.34	9	12.52	0.57	10	11.83	11.83
9	10	0.20	0.20	10	0.21	0.21	10	0.31	0.31	10	0.96	0.96	10	8.75	8.75
10	10	0.22	0.22	10	0.22	0.22	10	0.41	0.41	9	12.71	0.79	10	10.21	10.21
11	9	13.57	1.74	10	6.32	6.32	10	5.58	5.58	9	33.18	23.53	1	109.22	12.17
12	10	0.38	0.38	10	0.34	0.34	10	0.96	0.96	10	0.80	0.80	10	20.52	20.52
13	10	7.23	7.23	10	2.33	2.33	10	7.04	7.04	8	55.68	39.61	0	120.00	0.00
14	10	0.53	0.53	10	0.53	0.53	10	1.92	1.92	10	1.93	1.93	-	-	-
15	10	3.17	3.17	8	34.53	13.16	10	11.15	11.15	4	91.90	49.74	-	-	-
16	9	13.95	2.16	10	0.52	0.52	10	1.33	1.33	10	1.21	1.21	10	18.39	18.39
17	10	0.63	0.63	10	0.35	0.35	10	1.14	1.14	10	1.67	1.67	9	47.43	39.37
18	10	0.71	0.71	10	0.37	0.37	10	1.27	1.27	10	1.24	1.24	10	33.69	33.69



(a) Run-time for experiment 1. Runs up to 2^{13} variables take less than 0.01 s and don't show increase with number of variables.

(b) Run-time for experiment 2. All runs up to $c = 2^{12}$ take less than 0.01 s.

Figure 5. Experimental results for Experiments 1 and 2. Results are averaged over 10 runs with different random seed. All runs succeed.

Table 8. Run-times for additional constraints on adversarial perturbation. #✓is the number of successful runs out of 9 and ⏱✓is the average run-time over the successful runs in seconds. k row or column constraints corresponds to 27 individual constraints in DL2 each. Thus, the right most column adds 756 constraints for the first two settings and 1512 for the last.

Row constraints

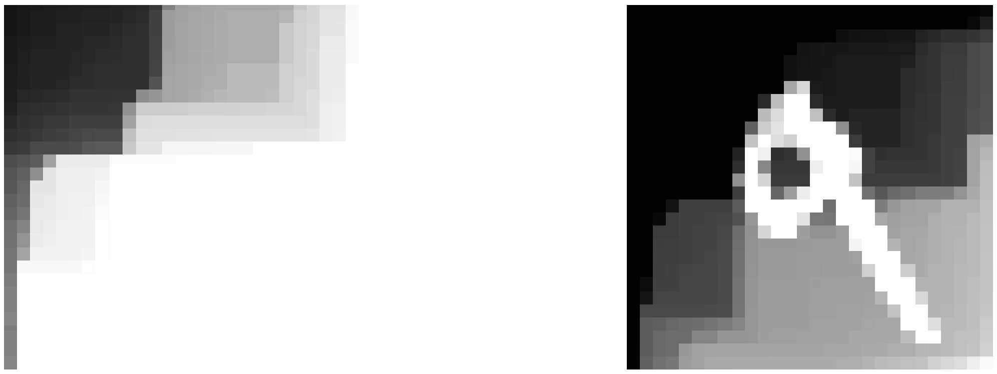
k	0	1	3	5	10	20	28
⏱✓ [s]	0.02	1.24	3.25	6.36	24.82	67.25	101.49
#✓	9	9	9	9	9	9	9

Column constraints

k	0	1	3	5	10	20	28
⏱✓ [s]	0.02	1.00	3.22	5.83	21.44	71.87	270.83
#✓	9	9	9	9	9	9	9

Row & Column constraints

k	0	1	3	5	10	20	28
⏱✓ [s]	0.02	1.89	6.14	11.06	89.02	213.30	270.83
#✓	9	9	9	9	9	6	1



(a) The found perturbation p, scaled such that -0.3 corresponds to black and 0.3 to white.

(b) The resulting image.

Figure 6. Found results for the full 28 row & column constraints and target class 6.