Beyond NaN: Resiliency of Optimization Layers in The Face of Infeasibility

Wai Tuck Wong[1], **Sarah Kinsey**[2], **Ramesha Karunasena**[1], **Thanh H. Nguyen**[2], **Arunesh Sinha**[3]
[1]Singapore Management University
[2]University of Oregon
[3]Rutgers University
wt.wong.2020@msc.smu.edu.sg, sarahevekinsey@gmail.com, rameshak@smu.edu.sg, thanhhng@cs.uoregon.edu, arunesh.sinha@rutgers.edu

Abstract

Prior work has successfully incorporated optimization layers as the last layer in neural networks for various problems, thereby allowing joint learning and planning in one neural network forward pass. In this work, we identify a weakness in such a set-up where inputs to the optimization layer lead to undefined output of the neural network. Such undefined decision outputs can lead to possible catastrophic outcomes in critical real time applications. We show that an adversary can cause such failures by forcing rank deficiency on the matrix fed to the optimization layer which results in the optimization failing to produce a solution. We provide a defense for the failure cases by controlling the condition number of the input matrix. We study the problem in the settings of synthetic data, Jigsaw Sudoku, and in speed planning for autonomous driving. We show that our proposed defense effectively prevents the framework from failing with undefined output. Finally, we surface a number of edge cases which lead to serious bugs in popular optimization solvers which can be abused as well.

Introduction

There is a recent trend of incorporating optimization and equation solvers as the *final layer* in a neural network, where the *penultimate layer* outputs parameters of the optimization or the equation set that is to be solved (Amos and Kolter 2017; Donti, Amos, and Kolter 2017; Agrawal et al. 2019; Wilder, Dilkina, and Tambe 2019; Wang et al. 2019; Perrault et al. 2020; Li et al. 2020; Paulus et al. 2021). The learning and optimizing is performed jointly by differentiating through the optimization layer, which by now is incorporated into standard libraries. Novel applications of this method have appeared for decision focused learning, solving games, clustering after learning, with deployment in real world autonomous driving (Xiao et al. 2022) and scheduling (Wang et al. 2022). In this work, we explore a novel attack vector that is applicable for this setting, but we note that the core concepts in this attack can be applied to other settings as well. While a lot of work exists in attacks on machine learning, in contrast, we focus on a new attack that forces the decision output to be meaningless via specially crafted inputs. The failure of the decision system to produce meaningful output can lead to catastrophic outcomes in critical domains such as autonomous driving where decisions are needed in real time. Also, such inputs when present in training data lead to abrupt failure of training. Our work *exploits the failure conditions of the optimization layer* of the joint network in order to induce such failure. This vulnerability has not been exploited in prior literature.

First, we present a *numerical instability attack*. Typically, an optimization solver or an equation set solver takes in parameters θ as input. In the joint network, this parameter θ is output by the learning layers and feeds into the last optimization layer (see Fig. 1). At its core, the issue lies in using functions which are prone to numerical stability issues in its parameters (see appendix). Most optimization or equation solvers critically depend on the matrix A—part of the parameter θ—to be sufficiently far from a singular matrix to solve the problem. Our attack proceeds by searching for input(s) that cause the matrix A to become singular. The instability produces *NaNs*—undefined values in floating-point arithmetic—which may result in undesired behavior in downstream systems that consume them. We perform this search via gradient descent and test three different ways of finding a singular matrix in neighborhood of A; only one of which works consistently in practice.

Second, to tackle the numerical instability attack, we propose a novel powerful defense via an efficiently computable intermediate layer in the neural network. This layer utilizes the singular value decomposition (SVD) of the matrix A and, if needed, approximates A closely with a matrix A' that has bounded condition number; the bound is a hyperparameter. Large condition number implies closeness to singularity, hence the bounded condition number guarantees numerical stability in the forward pass through the optimization (or equation) solver. Surprisingly, we find that the training performance with our defense in place surpasses the performance of the undefended model, even in the absence of attack, perhaps due to more stable gradients.

Finally, we show the efficacy of our attack and defense in (1) a synthetic data problem designed in Amos and Kolter (2017) and (2) a variant of the Sudoku experiment used in Amos and Kolter (2017) and (3) an autonomous driving scenario from Liu, Zhan, and Tomizuka (2017), where failures can occur even without attacks and how augmenting with our defense prevent these failures. Lastly, we identify

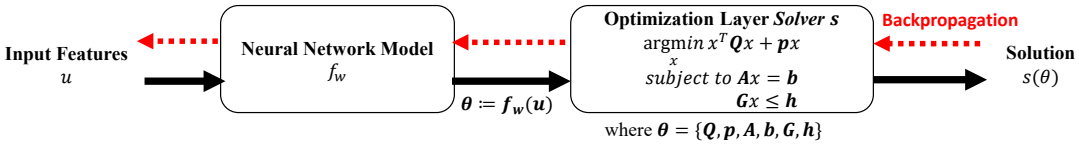


Figure 1: Optimization layers in neural networks. The neural network takes input u. Some parameters (Q, p, A, b, G, h) of the optimization then depend on the output $\theta = f_w(u)$.

other sources of failure in these optimization layers by invoking edge cases in the solver (see appendix). We list serious bugs in the solvers that we encountered.

Background, Notation, and Related Work

Matrix Concepts and Notation. The identity matrix is denoted as I and the matrix dimensions are given by subscripts, e.g., $I_{m\times m}$. The *pseudoinverse* (Laub 2004) of any matrix A is denoted by A^+; if A is invertible then $A^+ = A^{-1}$.The *condition number* (Belsley, Kuh, and Welsch 1980) of non-singular matrix A is defined as $\kappa(A) = \|A^+\|\,\|A\|$ for any matrix norm. We use $\kappa_2(A)$ when the norm used is 2-operator norm and $\kappa_F(A)$ when the norm used is the Frobenius norm. The (thin) SVD of a matrix A is given by $A = U\Sigma V^T$ where U, V have orthogonal columns ($U^TU = I = V^TV$) and Σ is a diagonal matrix with non-negative entries. If A is of dimension $m \times n$, then U, Σ, V^T are of dimension $m \times r$, $r \times r$, $r \times n$ respectively. The diagonal entries of Σ denoted as $\sigma_i = \Sigma_{i,i}$ are the singular values of the matrix A; singular values are always non-negative. The condition number directly depends on the largest and smallest singular value as follows: $\kappa_2(A) = \sigma_{\max}/\sigma_{\min}$. Also, $\|A\|_2 = \sigma_{\max}$. $\text{tr}(A)$ denotes the trace of a matrix.

Embedding Optimization in Neural Networks. Embedding a solver (for optimization or a set of equations) is essentially a composition of a standard neural network f_w and the solver s, where w represents weights. The function f_w takes in input u and produces parameters θ for the problem that the solver s solves. The solver layer takes θ as input and produces a solution $s(\theta)$. The composition $s \circ f_w$ can be jointly trained by differentiating through the solver s (see Fig. 1). The main enabler of this technique is efficient differentiation of the solver function s. Prior work has shown how to differentiate through solver s where s is a convex optimization problem (Amos and Kolter 2017), linear equation solver (Etmann, Ke, and Schönlieb 2020), clustering algorithm (Wilder et al. 2019), and game solver (Li et al. 2020). Such joint networks have been shown to provide better solution over separate learning and solving (Perrault et al. 2020).

Many applications of optimization layers focus on training the network end-to-end with the final output representing some decision of the overall AI system, typically called *decision focused learning* (Donti, Amos, and Kolter 2017; Wilder, Dilkina, and Tambe 2019; Wang et al. 2022). Though this has performed well in certain settings, such networks have not been investigated in terms of robustness.

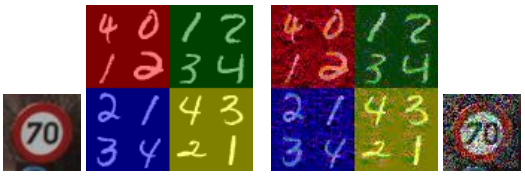
Adversarial Machine Learning. There is a huge body of work on adversarial learning and robustness that studies vulnerabilities of machine learning algorithms, summarized in many surveys and papers (Goodfellow, Shlens, and Szegedy 2015; Szegedy et al. 2014; Akhtar and Mian 2018; Biggio and Roli 2018; He, Li, and Song 2018; Papernot et al. 2018; Li, Bradshaw, and Sharma 2019; Anil, Lucas, and Grosse 2019; Tramer et al. 2020). Our work is different from prior work as our attack targets the *stability* of the optimization solver that is embedded as a layer in the neural network and our defense stops the attack by preventing singularity. To the best of our knowledge, our work is the first work to explore this aspect.

Robustness against Numerical Stability in Optimization. Repairing is an approach proposed in recent work (Barratt, Angeris, and Boyd 2021) to compute the closest solvable optimization when the input generic convex optimization is infeasible. While possessing the same goal as our defense, this repairing approach is computationally prohibitive for use in neural networks as the repairing requires solving tens to hundreds of convex optimization problems just to repair a single problem instance. Optimization layers are considered slow even with just one optimization in the forward pass (Amos and Kolter 2017; Agrawal et al. 2019; Wang et al. 2020), hence multiple optimizations to repair the core optimization in every forward pass is not practical for neural networks. Our defense is computationally cheap due to the targeted adjustment of specific parameters of the optimization.

Pre-conditioning (Wathen 2015) is a standard approach in optimization that helps the solver deal with ill-conditioned matrices better than without pre-conditioning. However, *even with preconditioning, solvers cannot handle specially crafted singular input matrices*. Our defense does not allow any input that the solver cannot handle.

Methodology

Threat Model: We are given a trained neural network which is a composition of two functions f_w and s, where w represents neural network weights and w is known to the adversary (i.e., the adversary has whitebox access to the model). The function f_w takes in input u and produces $\theta = f_w(u)$. θ defines some of the parameters to our solver (Fig. 1). In this paper, we analyze a specific component of θ which corresponds to the intermediate matrix A (in $Ax = b$). For example, if θ only consists of A, it can be formed by reshaping θ, where the i, j entry of A is $\theta_{i,j}$. The solver layer takes A as input and produces a solution $s(A)$. The attacker's goal is to craft any input u^* such that $s(f_w(u^*))$ fails to evaluate successfully due to issues in evaluating s stemming from numerical instability, effectively causing a denial of service. Note that the *existence* of any such input u^* is problematic and we allow latitude to the attacker to produce *any* such



(a) Original Images (b) Attack Images

Figure 2: Left shows original image u, right shows $u' = u + \delta$ which is semantically close. All attacks were found using AllZeroRowCol with a upper bound on the perturbations.

input as long as syntactical properties are maintained, e.g., bounding image pixel values in 0 to 1. In this setting, an attacker can also craft an attack input that is close to some original input if needed (Fig. 2), e.g., when they need to foil a human in the loop defense. Even in this worst case scenario of allowing the attacker to provide any input, our proposed defense prevents NaNs in all cases.

We emphasize the distinction between the goal of our attack inputs and that of adversarial examples. In traditional adversarial examples, small perturbations to the input image is sought in order to show the surprising effect that two images that appear the same to the human eye are assigned different class labels, but these misclassified labels can still be consumed by downstream systems. In contrast, in our work, the *surprise* is the *existence* of inputs that cause a complete failure in the outcome of the system, which to our knowledge have not been previously studied. Here, we show the existence of specially crafted inputs, which may be semantically close to a valid input, that evaluate to outputs that cause a complete denial of service, i.e., NaNs are produced, leading to undefined behavior in the system. A naive remediation of a default safe action for NaN outputs can fail in complex domains (e.g., autonomous driving) which have context-dependent safe actions (e.g., the safest action on a highway with a speed-limit road sign depends on various conditions such as speed of the car in front, need to change lane, etc.). It is thus impossible to provide a rule-based safe default action since there can be infinitely many contexts.

Numerical Instability Attack

In our attack, we seek to find an input u^* that evaluates to a rank deficient intermediate matrix A (Fig. 1). For any $m \times n$ matrix A, A is rank-deficient if its rank is strictly less than $\min(m, n)$. A rank deficient matrix is also singular, hence the system of equations $Ax = b$ ($b \neq 0$) produces undefined values (NaN) when solved directly or as constraints in an optimization. Even matrices close enough to singularity can still produce errors due to the limited precision of computers. Depending on the neural network f_w (Fig. 1), finding u that produces an arbitrary singular A (e.g., $0_{m\times n}$) is not always possible (see appendix). Our approach is guided by the following known result

Proposition 1 (Demko (1986))**.** *For any matrix A, the distance to closest singular matrix is* $\min_B\{\|A - B\|_2 : B \text{ is singular}\} = \|A\|_2 / \kappa_2(A) = \sigma_{\min}$

Algorithm 1 Numerical instability attack

Input: input features u, loss function ℓ, victim model f_w
Parameters: learning rate α
Output: attack input u^*

```
Let u* = u.
while κ2(f_w(u*)) ≠ ∞ do
    l = ℓ(f_w(u*)) {ℓ is a technique dependent loss}
    Update u* based on α, δl/δu*, ℓ
end while
return u*
```

Thus, increasing the condition number of A moves A closer to singularity; at singularity $\kappa_2(A)$ is ∞. Following Alg. 1, we start with a given u producing a well-conditioned matrix A and aim to obtain u^* producing singular A' in the *vicinity* of A using three approaches: AllZeroRowCol, ZeroSingularValue, and ConditionGrad.

AllZeroRowCol: An approach to obtain a rank-deficient matrix A' from A is to zero out a row (resp. column) in case $m < n$ (resp. $m > n$) in A. Then, we use A' as a target matrix for which a gradient descent-based search is performed to find an input u^*, that yields $A' = f_w(u^*)$. In our experiments, we choose the first row/column to zero out, though choosing other rows/columns is equally effective.

ZeroSingularValue: From Prop. 1, A' is a *closest singular matrix* if $\|A - A'\|_2 = \sigma_{\min}$. An approach to obtain this rank-deficient matrix A' from A is to perform the SVD $A = U\Sigma V^T$, then zero out the smallest singular value in Σ to get Σ', and then construct $A' = U\Sigma' V^T$. It follows from the construction that $\|A - A'\|_2 = \sigma_{\min}$. Then, using A' as a target matrix a gradient descent-based search is performed to find u^* that yields $A' = f_w(u^*)$. In theory, since A' is a *closest singular matrix* it should be easier to find by gradient descent compared to AllZeroRowCol. However, this approach fails in practice because precision errors make A' non-singular even though Σ' has a zero singular value.

ConditionGrad: From Prop. 1, we can also use gradient descent to find u^* such that the matrix A has a very high condition number. The overall gradient we seek is $\frac{\partial \log \kappa_2(A)}{\partial u}$, where we use log as condition numbers can be large. Following chain rule, we get $\frac{\partial \log \kappa_2(A)}{\partial u} = \frac{1}{\kappa_2(A)} \frac{\partial \kappa_2(A)}{\partial \theta} \frac{\partial \theta}{\partial u}$. Since $\theta = f_w(u)$, the third term is simply the gradient through the neural network. The second term can be obtained component wise in θ as $\frac{\partial \kappa_2(A)}{\partial \theta_{i,j}}$ for all i, j. The following result provides a closed form formula for the same (see proof in appendix).

Lemma 1. *Let $A \in \mathbb{R}^{m\times n}$ with thin SVD $A = U\Sigma V^T$ and $\sigma_{\max} = \sigma_1 \geq \ldots \geq \sigma_r = \sigma_{\min}$ for $r = \min(m, n)$. Then, $\frac{\partial \kappa_2(A)}{\partial \theta_{i,j}}$ is given by* $\operatorname{tr}\left(\frac{\partial(||A^+||_2 * ||A||_2)}{\partial A} \cdot \frac{\partial A}{\partial \theta_{i,j}}\right)$ *where*

$$\frac{\partial(||A^+||_2 * ||A||_2)}{\partial A} = B^T - (A^+ C A^+)^T + (A^+)^T A^+ C(I - A^+ A) + (I - AA^+) C A^+ (A^+)^T$$

with $B = ||A^+||_2 V e_1 e_1^T U^T$, $C = ||A||_2 U e_r e_r^T V^T$ and e_i is the unit vector with one in the i^{th} position.

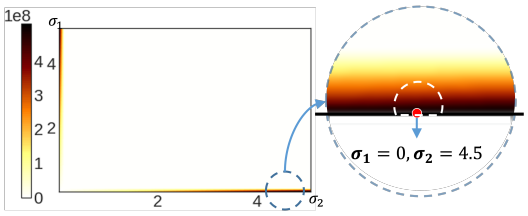


Figure 3: Left is a heatmap of condition numbers for 2D singular value space (σ_1, σ_2) of 2×2 matrices (high condition number near axes, as one of σ_1 or σ_2 approaches 0). Right is an enlarged version of the smaller dashed circle. A reconstructed A using $\boldsymbol{\sigma}_1, \boldsymbol{\sigma}_2$ should be singular, but precision loss makes the singular values of A almost never 0 and they lie in the white dashed circle.

Algorithm 2 Numerical instability defense

Input: model f_w, input features u
Parameter: condition number bound B
Output: well-conditioned A'

Let $A' = A = f_w(u) = U\Sigma V^T$.
if $\kappa_2(A) > B$ **then**
  For all i, let $\Sigma'_{i,i} = \min(\sigma_i, \sigma_{max}/B)$
  $A' = U\Sigma' V^T$
end if
return A'.

ConditionGrad still works less consistently than AllZeroRowCol. This is mainly because the gradient descent often saturates at a condition number that is high but not large enough for instability.

A low-dimension illustration of the approaches is in Fig. 3, which shows the 2D space of the two singular values σ_1, σ_2 of all 2×2 matrices. The condition number $(\sigma_{\max}/\sigma_{\min})$ is ∞ only on the axes and is difficult to reach in ConditionGrad. The illustration also shows why AllZeroRowCol works more consistently than ZeroSingularValue as recovering a matrix from Σ' involves multiplication which leads to loss of singularity (more so in high dimension) whereas AllZeroRowCol directly obtains a singular matrix. This is reflected in our experiments later.

We note that simple approaches such as attempting to use gradient descent or other existing approaches to directly maximize model output to very high values fails due to saturation (see results in appendix). Further, the optimization output and ill-conditioning of A can have no relation at all:

Lemma 2. *For an optimization* $\min_{\{x|Ax=b\}} f(x)$ *with* f *convex, the solution value (if it exists) can be made arbitrarily large by changing* $\theta = \{A, b\}$ *while keeping* A *well-conditioned.*

Lemma 2 implies that A can remain well-conditioned even though output $\min_{\{x|Ax=b\}} f(x)$ is large. Thus, specifically targeting to directly obtain a singular matrix A is important for a successful NaN attack (proof in appendix).

Defense Against Numerical Instability

First, we note that our goal is to fix the instability in the optimization used in the final layer, which is distinctly different from the general problem of instability of training neural networks (Colbrook, Antun, and Hansen 2022). Next, we discuss defense for *square matrices* A. For symmetric square matrices, the condition number can be stated in term of eigenvalues: $\kappa_2(A) = \frac{|\lambda|_{\max}}{|\lambda|_{\min}}$ where $|\lambda|_{\max}$ is the largest eigenvalue by magnitude. A typical heuristic to avoid numerical instability for square matrices is to add ηI for some small η (Haber and Ruthotto 2017). However, this approach *only works for square* positive semi-definite (PSD) matrices. If some eigenvalue of A happens to $-\eta$ then this heuristic actually makes the resultant matrix non-invertible (i.e., infinite condition number). Besides, clearly this heuristic *does not apply for non-square matrices*.

As a consequence, we propose a differentiable technique (Alg. 2) that directly *guarantees* the condition number of *any* intermediate matrix to be a bounded by a hyperparameter B. In the forward pass, we perform a SVD of $A = U\Sigma V^T$; the computation steps in SVD are differentiable and the matrix Σ gives the singular values σ_i's. Recall that the condition number $\kappa_2 = \sigma_{\max}/\sigma_{\min}$. The condition number can be controlled by clamping the σ_i's to a minimum value $\sigma_{\max}/B$ to obtain a modified Σ'. Then, we recover the approximate $A' = U\Sigma' V^T$. We present the following proposition (proof in appendix).

Proposition 2. *For the approximate* A' *obtained from* A *as described above and* x' *a solution for* $A'x = b$*, the following hold: (1)* $\|A' - A\|_2 \leq \sigma_{\max}/B$ *and (2)* $\frac{||x^* - x'||_2}{||x'||_2} \leq \kappa_2(A)/B$ *for some solution* x^* *of* $Ax = b$*.*

The second item (2) shows that approximation of the solution obtained from the solver depends on $\kappa_2(A)$, which can be large if $\kappa_2(A)$ is close to infinity. This error estimate can be provided to downstream systems which can be used in the decision on whether to use the solver's output.

Experiments

We showcase our attacks and defenses on three different domains: (i) synthetic data modelling an assignment problem, (ii) decision-focused solving of Jigsaw Sudoku puzzles, and (iii) real world speed profile planning for autonomous driving. We compare the success rate of the three attack methods, namely AllZeroRowCol, ZeroSingularValue and ConditionGrad. We show that the defense is effective by comparing the condition numbers of the constraint matrix A during test time. We also show that the attack fails with the defense for varying values of B: 2, 10, 100, and 200. Further, we augment our model with the defense during training time and show that it effectively prevents NaNs in training while not sacrificing performance compared to the *original* model. We discuss the results in detail at the end in Section .

For all experiments, we used the `qpth` batched QP solver as the optimization layer (Amos and Kolter 2017) and PyTorch 1.8.1 for SVD. For the synthetic data, we ran the experiments on a cloud instance (16 vCPUs, 104 GB memory) on CPU. For other settings, we ran the experiments on a server (Intel(R) Xeon(R) Gold 5218R CPU, 2x Quadro RTX 6000, 128GB RAM) on the GPU.

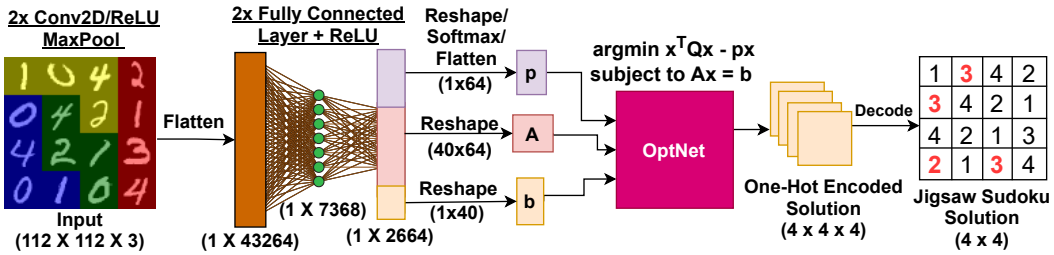


Figure 4: The Jigsaw Sudoku network architecture. The input is an image of the Jigsaw Sudoku puzzle (0 indicates blank cell that needs to be filled with a value in $\{1, 2, 3, 4\}$) and the output is the solution to the puzzle given the constraint that no two numbers in the same colored region are the same. The solution to the blank cells given by the neural network is indicated in red.

Synthetic Data

The setting used follows prior work (Amos and Kolter 2017) to test varying constraint matrix sizes used in optimization. We interpret this prior abstract problem as an assignment problem under constraints, where inputs are assigned to bins with constraints that are learnable. The model learns parameters in the network to best match the bin assignment in the data. The input features of input u are generated from the Gaussian distribution and assigned one out of n bins uniformly. Bin assignment is a constrained maximization optimization, where only the constraint affects binning; thus, the objective is arbitrarily set to $\|x\|_2$ with the constraint $Ax = b$, where A, b are learned and $x \in \mathbb{R}^n$ gives the bin assignment. Here, A has the size $m \times n$, where m is the number of equalities and n is the number of bins. A softmax layer at the end enforces an assignment constraint.

Experimental Setup: For the training of the network, in each of the randomly seeded training run, we draw 30 input feature vectors $u \in \mathbb{R}^{500}$ from the Gaussian distribution and assign them uniformly to n bins. We do the same for the test set comprising of 10 test samples. We ran the training over 1000 epochs using the Adam optimizer (Kingma and Ba 2015) with a fixed learning rate of $1\mathrm{e}{-3}$. For the attack experiments, we ran each of the attacks for 5000 epochs on 30 input samples drawn from the Gaussian distribution on each of the 10 models that were trained. An attack is marked successful if any of the modified inputs produces a NaN. For the training of the defended models, we varied the hyperparameter B. The models are evaluated using *cross-entropy loss* against the true bin in which the sample was assigned. The test loss is averaged over 10 randomly seeded runs.

Results: In this setting where an attacker can arbitrarily change the input vector at test time, we report the success rate of each of the attack methods in Table 1 and the loss results of models trained with the defense in Table 2 for the non-square matrix $A \in \mathbb{R}^{40\times50}$ case and square matrix $A \in \mathbb{R}^{50\times50}$ case. We see our methods are broadly applicable to all matrices as both the attack and the defense achieve their goals regardless of the shape of the matrix. Further, test performance in the *baseline* ηI defense (with $\eta = 10^{-8}$, applicable *only* for square matrices) in the $A \in \mathbb{R}^{50\times50}$ case is worse than *both* the original and our proposed defense when $B = 200$, with a higher loss at 4.86 ± 1.74 (see appendix).

Jigsaw Sudoku

Sudoku is a constraint satisfaction problem, where the goal is to find numbers to put into cells on a board (typically 9×9) with the constraint that no two numbers in a row, column, or square are the same. In prior work (Amos and Kolter 2017), optimization layers were used to learn constraints and obtain solutions satisfying those constraints on a simpler 4×4 board. We note that in the above setting, the constraints (A, b) are fixed and do not vary with the input Sudoku instances and hence, our test time attack does not apply in this case. Instead, we consider a popular variant of the 4×4 Sudoku—Jigsaw Sudoku—where constraints are not just on the rows and columns, but also on other geometric shapes made from four contiguous cells. In this setting, the constraints now vary with input puzzle instances. We represent each Sudoku puzzle as an image (see Fig. 4) and mark each constraint on contiguous shapes with a different color.

The network (Fig. 4) has to (i) *infer* the one-hot encoded representation of the Sudoku problem p (a $4 \times 4 \times 4$ tensor with a one-hot encoding for known entries and zeros for unknown entries); (ii) *infer* the constraints to apply (A, b in $Ax = b$); and (iii) *solve* the optimization task to output the solution that satisfies the constraints of the puzzle — all these steps have to be derived just from the image of the Jigsaw Sudoku puzzle. A small $Q = 0.1I$ ensures strict positive definiteness and convexity of the quadratic program.

Experimental Setup: We generated 24000 puzzles of the form shown in Fig. 4 by composing and modifying images in the MNIST dataset (Lecun et al. 1998) using a modified generator from (Amos and Kolter 2017) (details in appendix). We trained the model architecture in Fig. 4 on 20000 Jigsaw Sudoku images, utilizing Adadelta (Zeiler 2012) with a learning rate of 1, batch size of 500, training over 20 epochs, minimizing the MSE loss against the actual solution to the puzzle. We then test the model on 4000 different held-out puzzles. We repeat the experiments with 30 random seeds for each configuration. For the defense, we restrict the condition number by applying our defense in Section over several values of the hyperparameter B. For all models, we measure MSE loss and *accuracy* which is the percentage of cells with the correct label in the solution produced by the network. For all attack methods, we apply a model-tuned learning rate and optimize for the attack loss for a given image for 500 epochs

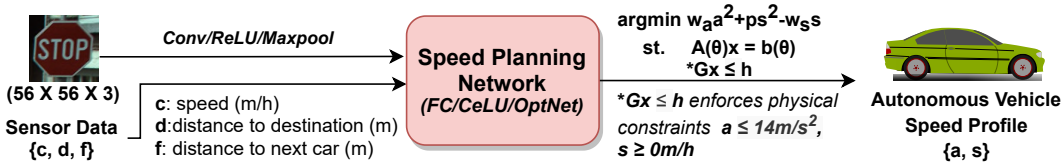


Figure 5: Autonomous vehicle speed planning architecture with CeLU (Barron 2017) activated layers.

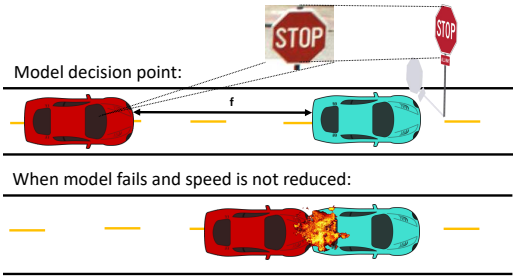


Figure 6: The car determines the optimal speed profile based on the signage and distance f from the car ahead. When the optimization fails, the car exhibits undefined behavior and may assume an unsafe profile, leading to a crash.

until we generate an image that causes failure. We then repeat this for 30 test images.

Results: In this setting, the attack may only modify the input image, which is constrained as a tensor with pixel values in the range $[0, 1]$. Even with these constraints, AllZeroRowCol consistently finds an input that results in NaNs in the output (Table 1), showing the effectiveness of our attack. Looking at the difference in loss (Fig. 7(b)) and condition number of the matrix A (Fig. 7(c)) for the defended and original undefended network, we see the efficacy of our defense in controlling the condition number and preventing the NaN outputs during test time. Finally, plotting the change in training loss over the epochs for $B = 100$ and the original model in Fig. 7(a), we see virtually no difference in epochs to convergence when the defense is applied in training time. We note the observations above apply for all experimental settings for all reported values of B, see appendix for details.

Autonomous Vehicle Speed Planning

In autonomous driving, a layered framework with separate path planning and speed profile generation is often used due to advantages in computational complexity (Gu et al. 2015). Here, we focus on speed profile generation, where constrained optimization is employed to maximize comfort of the passengers while ensuring their safety and adhering to physical limitations of the vehicle (Ziegler et al. 2014). We consider the scenario where a traffic sign is observed and the autonomous vehicle has to make a decision on the acceleration and target speed of the vehicle as shown in Fig. 6.

The autonomous vehicle seeks to make the optimal decision in speed planning taking into account the constraints presented. The learning problem involves identifying the traffic sign and inferring the rules to apply based on the current aggregate state of the autonomous vehicle collected from sensors. We provide as input u an image of the traffic sign along with the state of the vehicle, defined as $V = \{c, d, f\}$, where $c, d, f \in \mathbb{R}_{\geq 0}$, where, c is the current speed of the vehicle (in meters per hour), d is the distance to the destination (in meters), and f is the distance to the vehicle ahead (in meters). Similar to (Liu, Zhan, and Tomizuka 2017), we aim to minimize discomfort a^2 ,where acceleration $a \in \mathbb{R}$, and maximize the target speed $s \in \mathbb{R}_{\geq 0}$ using the quadratic program shown in Fig. 5, where $w_a, w_s \in \mathbb{R}$ are tunable weights on the speed and acceleration, and $p \in \mathbb{R}_{\geq 0}$ is a small penalty term to ensure the problem is a quadratic program. When input u is fed into the network, θ is the output of the network right before the optimization layer, and $A(\theta)$ and $b(\theta)$ depend on θ. These equalities encode rules that will apply based on the traffic sign observed, e.g. a *Stop* sign would signal to the vehicle to set its target speed s to 0. We encode physical constraints of the autonomous vehicle (e.g., maximum acceleration, positivity constraints on speed) in G and h which do not depend on θ.

Experimental Setup: To generate the input, we utilize 5 traffic sign classes of the BelgiumTS dataset (Timofte, Zimmermann, and Van Gool 2011) for the images. These traffic signs require an immediate change in speed/acceleration (e.g. Stop, Yield). We combine the image with the current state of the vehicle $V = \{c, d, f\}$ at the decision point and generate 10000 training samples and 1000 test samples for use in our holdout set. We enforce the following constraints through the matrix G, h: all acceleration is at most $14m/s^2$, and speed must be positive. We setup the network as in Fig. 5 and employ the Adam optimizer (Kingma and Ba 2015) with a learning rate of $1e{-}4$, run the experiments for 30 epochs, and average the result over 30 random seeds. The models were evaluated using loss functions which penalizes different aspects of the decision: $\mathcal{L}_{safety}$ loss due to impact of collision with vehicle ahead, $\mathcal{L}_{comfort}$ loss due to discomfort from acceleration, $\mathcal{L}_{distance}$ loss due to slowing down and $\mathcal{L}_{violation}$ loss from violating the traffic sign. The $\mathcal{L}_{total}$ loss which is a weighted sum of the above losses is reported (details in appendix). For the attack, we apply a model-tuned learning rate and optimize for various attack losses for 500 epochs over 30 random images from the test set.

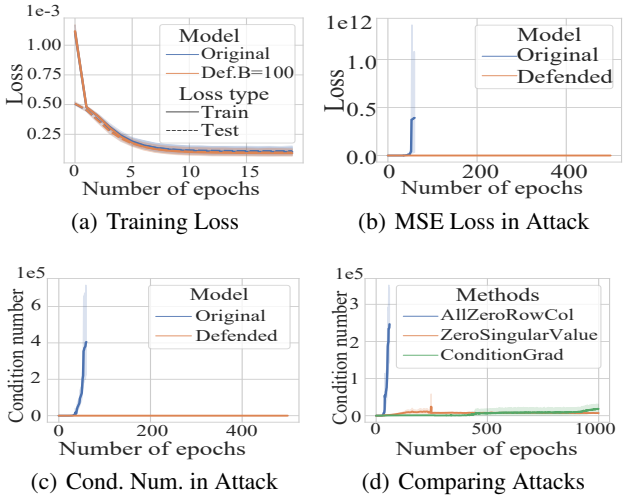
Results: Even in this restricted and complex setting where we only allow the attacker to modify the images and not the state of the vehicle, the test time attack success rate is still high for AllZeroRowCol (see Table 1). We note that this setting is analogous to the real world where attackers can easily control the environment but not the sensor inputs of

	AllZeroRowCol	ZeroSingularValue	ConditionGrad
Synthetic (m=40, n=50)	100.00	0.67	85.33
Synthetic (m=50, n=50)	98.00	0.00	0.00
Jigsaw Sudoku	100.00	6.67	53.33
Speed Planning	100.00	0.00	0.00
Defense (B=2,10,100,200)	**0.00**	**0.00**	**0.00**

Table 1: Comparison of attack success (% of Successful NaNs) for all methods and datasets. Last row shows defense.

	Synthetic Data (CE)		**Jigsaw Sudoku**		**Speed Planning**
	m=40, n=50	**m=50, n=50**	**Test Loss**	**Test Acc.**	**Test Loss**
Original	24.99 ± 2.03	4.43 ± 0.93	1.09 ± 1.03	0.91 ± 0.20	7928.76 ± 38565
B=2	$\mathbf{9.14 \pm 0.77}$	6.41 ± 0.58	0.93 ± 0.73	0.94 ± 0.15	$\mathbf{6.64 \pm 3.52}$
B=10	11.67 ± 1.26	11.53 ± 0.71	$\mathbf{0.82 \pm 0.53}$	$\mathbf{0.96 \pm 0.09}$	83.3 ± 404.8
B=100	23.36 ± 2.08	6.36 ± 1.50	0.96 ± 0.87	0.93 ± 0.16	117.8 ± 443
B=200	23.27 ± 1.75	$\mathbf{3.93 \pm 0.07}$	0.99 ± 0.87	0.93 ± 0.16	1381.9 ± 6905.5

Table 2: Performance of different models trained with defense in place. The loss is on a held-out validation set. For Synthetic Data, loss is cross-entropy (CE). For Jigsaw Sudoku, loss is MSE in order of $\times 10^{-4}$. For Speed Planning, loss is $\mathcal{L}_{total}$.



(a) Training Loss (b) MSE Loss in Attack

(c) Cond. Num. in Attack (d) Comparing Attacks

Figure 7: Effect of attacks and defense in Jigsaw Sudoku

the vehicle or the physical constraints of the car (the attacker has no control over $Gx \leq h$). We report the overall loss $\mathcal{L}_{total}$ when training with defense in Table 2.

Discussion of Results

Efficacy of the attack methods: The simplest AllZeroRowCol attack was the most effective for all experiment settings (see Table 1) for a range of activation functions—ReLU, CeLU, and also tanh (see appendix). This is surprising, given that the theoretically principled ZeroSingularValue worked the least consistently empirically, and even the theoretically motivated ConditionGrad attack worked inconsistently. This highlights the difficulty of transferring theoretical results to the real world, especially with limited numerical precision.

Stabilizing effect of defense: All attacks are thwarted by our defense, showing the effectiveness of controlling the condition number. We also observe that by tuning the B hyperparameter, we are able to train with the defense without any tradeoffs in terms of learning or accuracy (Table 2), and across all domains, we find that it adds less overhead (a constant factor less than 2) than the actual optimization. In fact, at certain values of B, we achieve lower loss and higher accuracy compared to the original undefended model. We conjecture that this occurs when the true matrix exists within the space of the bounded condition number, and the low number makes the gradients of the optimization layer stable.

On achieving general trustworthiness: Further auditing library functions, we noted several related issues which can be abused (some found by us, others by practitioners as benign flaws). Attackers can exploit these to produce solutions that violate the constraints of the optimization or even produce incorrect results when the input is a singular matrix (see appendix). Careful audits should be performed on the implementation, down to potential edge cases in the data types.

Conclusion

Our work scratches the surface of a new class of vulnerabilities that underlie neural networks, where rogue inputs trigger edge cases that are not handled in the underlying math or engineering of the layers, which lead to undefined behavior. We showed methods of constructing inputs that force singularity on the input matrix of equality constraints for optimization layers, and proposed a guaranteed defense via controlling the condition number. We hope that our work raises awareness of this new class of problems so the community can band together to resolve them with novel solutions.

References

Agrawal, A.; Amos, B.; Barratt, S.; Boyd, S.; Diamond, S.; and Kolter, J. Z. 2019. Differentiable Convex Optimization Layers. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Akhtar, N.; and Mian, A. 2018. Threat of adversarial attacks on deep learning in computer vision: A survey. *Ieee Access*, 6: 14410–14430.

Amos, B.; and Kolter, J. Z. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, 136–145.

Anil, C.; Lucas, J.; and Grosse, R. 2019. Sorting out Lipschitz function approximation. In *International Conference on Machine Learning*, 291–301. PMLR.

Barratt, S.; Angeris, G.; and Boyd, S. 2021. Automatic repair of convex optimization problems. *Optimization and Engineering*, 22: 247–259.

Barron, J. T. 2017. Continuously Differentiable Exponential Linear Units. arXiv:1704.07483.

Belsley, D. A.; Kuh, E.; and Welsch, R. E. 1980. The condition number. *Regression diagnostics: Identifying influential data and sources of collinearity*, 100: 104.

Biggio, B.; and Roli, F. 2018. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84: 317–331.

Colbrook, M. J.; Antun, V.; and Hansen, A. C. 2022. The difficulty of computing stable and accurate neural networks: On the barriers of deep learning and Smale's 18th problem. *Proceedings of the National Academy of Sciences*, 119(12): e2107151119.

Demko, S. 1986. Condition numbers of rectangular systems and bounds for generalized inverses. *Linear Algebra and its Applications*, 78: 199–206.

Donti, P. L.; Amos, B.; and Kolter, J. Z. 2017. Task-based end-to-end model learning in stochastic optimization. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 5490–5500.

Etmann, C.; Ke, R.; and Schönlieb, C.-B. 2020. iUNets: Fully invertible U-Nets with learnable up-and downsampling. *arXiv preprint arXiv:2005.05220*.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations*.

Gu, T.; Atwood, J.; Dong, C.; Dolan, J. M.; and Lee, J.-W. 2015. Tunable and stable real-time trajectory planning for urban autonomous driving. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 250–256.

Haber, E.; and Ruthotto, L. 2017. Stable Architectures for Deep Neural Networks. *Inverse Problems*, 34.

He, W.; Li, B.; and Song, D. 2018. Decision boundary analysis of adversarial examples. In *International Conference on Learning Representations*.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. 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*.

Laub, A. J. 2004. *Matrix Analysis For Scientists And Engineers*. Society for Industrial and Applied Mathematics. ISBN 0898715768.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Li, J.; Yu, J.; Nie, Y.; and Wang, Z. 2020. End-to-End Learning and Intervention in Games. *Advances in Neural Information Processing Systems*, 33.

Li, Y.; Bradshaw, J.; and Sharma, Y. 2019. Are generative classifiers more robust to adversarial attacks? In *International Conference on Machine Learning*, 3804–3814. PMLR.

Liu, C.; Zhan, W.; and Tomizuka, M. 2017. Speed profile planning in dynamic environments via temporal optimization. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, 154–159.

Papernot, N.; McDaniel, P.; Sinha, A.; and Wellman, M. P. 2018. Sok: Security and privacy in machine learning. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, 399–414. IEEE.

Paulus, A.; Rolínek, M.; Musil, V.; Amos, B.; and Martius, G. 2021. Comboptnet: Fit the right np-hard problem by learning integer programming constraints. In *International Conference on Machine Learning*, 8443–8453. PMLR.

Perrault, A.; Wilder, B.; Ewing, E.; Mate, A.; Dilkina, B.; and Tambe, M. 2020. End-to-End Game-Focused Learning of Adversary Behavior in Security Games. In *AAAI*, volume 34, 1378–1386.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations*.

Timofte, R.; Zimmermann, K.; and Van Gool, L. 2011. Multi-view Traffic Sign Detection, Recognition, and 3D Localisation. *Machine Vision and Applications*.

Tramer, F.; Carlini, N.; Brendel, W.; and Madry, A. 2020. On Adaptive Attacks to Adversarial Example Defenses. *Advances in Neural Information Processing Systems*, 33.

Wang, K.; Verma, S.; Mate, A.; Shah, S.; Taneja, A.; Madhiwalla, N.; Hegde, A.; and Tambe, M. 2022. Decision-focused learning in restless multi-armed bandits with application to maternal and child care domain. *arXiv preprint arXiv:2202.00916*.

Wang, K.; Wilder, B.; Perrault, A.; and Tambe, M. 2020. Automatically Learning Compact Quality-aware Surrogates for Optimization Problems. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 9586–9596. Curran Associates, Inc.

Wang, P.-W.; Donti, P.; Wilder, B.; and Kolter, Z. 2019. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, 6545–6554. PMLR.

Wathen, A. J. 2015. Preconditioning. *Acta Numerica*, 24.

Wilder, B.; Dilkina, B.; and Tambe, M. 2019. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *AAAI*, volume 33, 1658–1665.

Wilder, B.; Ewing, E.; Dilkina, B.; and Tambe, M. 2019. End to end learning and optimization on graphs. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Xiao, W.; Wang, T.-H.; Chahine, M.; Amini, A.; Hasani, R.; and Rus, D. 2022. Differentiable control barrier functions for vision-based end-to-end autonomous driving. *arXiv preprint arXiv:2203.02401*.

Zeiler, M. D. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR*, abs/1212.5701.

Ziegler, J.; Bender, P.; Dang, T.; and Stiller, C. 2014. Trajectory planning for Bertha — A local, continuous method. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, 450–457.