Activation-Descent Regularization for Input Optimization of ReLU Networks

Hongzhan Yu [1] **Sicun Gao** [1]

Abstract

We present a new approach for input optimization of ReLU networks that explicitly takes into account the effect of changes in activation patterns. We analyze local optimization steps in both the input space and the space of activation patterns to propose methods with superior local descent properties. To accomplish this, we convert the discrete space of activation patterns into differentiable representations and propose regularization terms that improve each descent step. Our experiments demonstrate the effectiveness of the proposed input-optimization methods for improving the state-of-the-art in various areas, such as adversarial learning, generative modeling, and reinforcement learning.

1. Introduction

Many problems in deep learning involves optimizing the inputs of neural networks, instead of the model parameters. Examples include finding effective adversarial attacks (Madry et al., 2017; Wong & Kolter, 2018; Ilyas et al., 2019; Xu et al., 2020; Zhang et al., 2022), optimizing latent variables in generative models (Bojanowski et al., 2017; Zadeh et al., 2019), finding actions that maximizes Q-values in reinforcement learning (Lillicrap et al., 2015; Fujimoto et al., 2018; 2019), and verification of neural networks (Bunel et al., 2018; Cohen et al., 2019; Shi et al., 2023). Such input optimization problems typically involve lower-dimensional problems than the standard parameter optimization problem in network training, but the loss landscapes can be more complex with a stronger need for dealing with the combinatorial nature of it (Agrawal et al., 2019; Gurumurthy et al., 2021).

The standard approach for input optimization is to follow the gradient of the objective function over the input variables. However, this approach can be problematic, especially when dealing with ReLU networks, as the ReLU activation function is discontinuous and non-differentiable, resulting in a gradient of zero for all negative pre-activation values. This means that the steepest descent direction only accounts for descent within the decision boundaries of a specific ReLU activation pattern. In deep neural networks with a large number of activation patterns, the region of inputs for each pattern is typically small. As a result, following the gradient direction can lead to the input jumping over to a different activation pattern with vastly different numerical properties, resulting in significant deviation from the desired objective direction, even if the step sizes are chosen to be very small.

We develop new methods for input optimization that take into account the impact of changes in activation patterns on the output. To accomplish this, we adopt a dual perspective by considering the local optimization steps both from the input space and the space of activation patterns. We convert the original discrete space of activation patterns into differentiable representations to obtain descent directions in the activation pattern space that can be continuously tuned. This allows us to introduce new regularization terms for local descent in the input space that encourages the input change to be aligned with the descent directions in the activation space. We then form the Lagrangian of the original objective with the regularizers and perform primal-dual descent. The overall procedures can thus achieve better local descent properties than gradient descent and also various forms of randomly perturbed gradient methods.

Through our experiments[1], we show that the proposed methods can significantly improve input optimization. We use benchmarks from several applications, such as optimizing adversarial attacks to neural image classifiers, reconstructing target images with generative models, and deep reinforcement learning with better action selection.

2. Related Work

Input optimization in deep learning. Many problems in deep learning can be formulated as input optimization problems. One canonical application is to construct adversarial attacks (Goodfellow et al., 2014; Madry et al., 2017; Wong

[1]Department of Computer Science & Engineering, University of California San Diego, USA. Correspondence to: Hongzhan Yu <hoy021@ucsd.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

[1]Codes are available at github.com/hoy021/ADR-GD.

& Kolter, 2018; Ilyas et al., 2019; Xu et al., 2020). This centralizes around optimizing an objective that leads to an incorrect label prediction by a classifier, while constraining the perturbed input to be within a bounded region around the original input. Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014), and its multi-step variant Projected Gradient Descent (PGD) (Madry et al., 2017; Croce & Hein, 2019) are the two most popular strategies, both of which utilize the signs of input gradients in constructing adversarial examples. However, they both heavily rely on the approach of standard gradient descent applied to non-convex objectives. Generally speaking, many important inverse problems (Bora et al., 2017; Rick Chang et al., 2017; Ongie et al., 2020) and auxiliary tasks (Pattanaik et al., 2017; Amos et al., 2018; Oikarinen et al., 2021; Yang et al., 2022) require solving input optimization problems. (Bora et al., 2017) and (Rick Chang et al., 2017) showed that one can use a pre-trained generative model as a prior to solve a variety of problems such as image reconstruction, denoising and inpainting, by optimizing on the latent space of generative models. (Amos et al., 2018) proposed to train differential Model Predictive Control (MPC) as a policy class for reinforcement learning, and construct controls by optimizing the cost and dynamics function both parameterized by ReLU networks. (Huang et al., 2017) and (Ilahi et al., 2021) demonstrate the effectiveness of white-box adversarial attacks on neural network policies. (Yang et al., 2024) employs adversarial optimization to generate critical samples for improving the learning of neural Lyapunov-like functions (Chang et al., 2019; Taylor et al., 2020; Yu et al., 2023). Those adversarial attacks are typically generated by optimizing on the input of neural networks with FGSM or PGD. Meanwhile, recent works (Amos et al., 2017; Makkuva et al., 2020) propose new network designs which yield convex network output with respect to its inputs. They allow the potentials of employing more sophisticated optimization algorithms in solving input optimization, but usually at the heavy cost of model expressivity and capacity.

Non-convex optimization. Previous work on searching second-order stationary points can be divided into two categories. Hessian-based approach (Nesterov & Polyak, 2006; Curtis et al., 2017) relies on computing the Hession to distinguish between first- and second-order stationary points. (Nesterov & Polyak, 2006) designed a cubic regularization of Newton method, and analyzed its convergence rate to the second-order stationary points of non-convex objective. Trust region methods (Curtis et al., 2017) can reach comparable performance if the algorithm parameters are carefully chosen. These methods typically require expensive computation of the full Hessian inverse, which motivates the attempts to accelerate them by using only Hessian-vector products (Agarwal et al., 2016; Carmon & Duchi, 2016; Carmon et al., 2018). (Carmon & Duchi, 2016) applied gradient descent as a subroutine to approximate the cubic-regularized Newton step, in which a Hessian-vector product oracle is required. (Tripuraneni et al., 2018) utilized a stochastic Hessian-vector product in further accelerating the cubic-regularized Newton method.

Another line of work shows that it is possible to converge to the second-order stationary points without direct use of the Hessian. (Ge et al., 2015) showed that stochastic gradient descent can converge to a second-order stationary point in polynomial time. Levy et al. (Levy, 2016) improved the convergence rate with gradient normalization. (Jin et al., 2017) and (Guo et al., 2020) proposed to apply gradient perturbations when the second-order information indicates the potential existence of nearby saddle points. It can find a second-order stationary point in a comparable time for converging to a first-order stationary point, as long as the non-convex objective satisfies a Hessian Lipschitz property.

3. Preliminaries

We will focus on feed-forward neural networks with ReLU activation, but the methods can be used in the ReLU components in other neural architectures, such as convolution networks and ResNet, and will be shown in the experiments.

An l-layer ReLU network defines a vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$, where n and m are the input and output dimensions. We write the weight matrices across the l layers as $(W_1, W_2, ..., W_{l+1})$ and the bias vectors as $(b_1, b_2, ..., b_{l+1})$, where W_{l+1} and b_{l+1} connect the last layer to the outputs. The ReLU activation is $\sigma(x) = \max(x, 0)$, where x can be a vector and the max is taken coordinate-wise. Thus, for any input $x \in \mathbb{R}^n$, the output of the network $f(x) \in \mathbb{R}^m$ is determined by the following equations:

$$\begin{aligned} h^{(1)} &= W_1 \cdot x + b_1, \\ h^{(i+1)} &= W_{i+1} \cdot \sigma(h^{(i)}) + b_{i+1}, \quad i \in [1, l], \\ f(x) &= h^{(l+1)} \end{aligned} \tag{1}$$

For $i \in [1, l]$, we write d_i for the dimensionality of each $h^{(i)}$, i.e., the number of activation units in each layer i. With $d_0 = n$ and $d_{l+1} = m$, each W_i is of shape $d_i \times d_{i-1}$.

Given an arbitrary input x to the network, the output of each activation unit takes either zero or positive values. Activation units with zero value outputs are called *dead neurons*, and the positive ones are the *active neurons*. An *activation pattern* of the network on input x is the function

Definition 3.1 (Activation Pattern)**.** Let the network f be as in (1), then the activation pattern is a function with

$$A_f(i, j) = \text{sgn}(\sigma(h_j^{(i)}(x))) \tag{2}$$

on input x at the j^{th} neuron on the i^{th} layer.

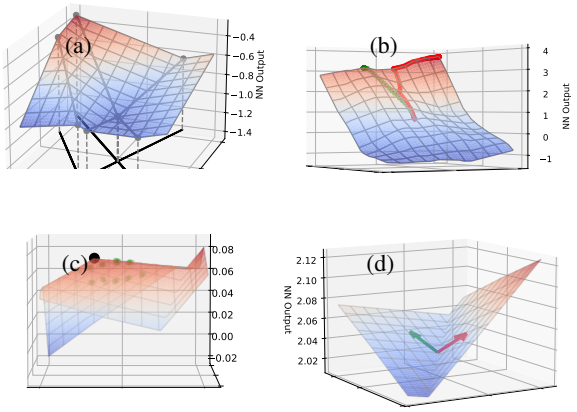


Figure 1: Value landscape of 2-D ReLU networks with scalar output space. **(a)** A small ReLU network with 3 hidden neurons. Each solid black line corresponds to one neuron, representing the set of inputs that achieve zero values. **(b)** The green and red lines demonstrate the trajectories under the optimization of vanilla GD and the proposed algorithm respectively. **(c)** The scenario where GD optimization gets stuck around a local maxima. The green dots demonstrate the GD steps, and the black dot is the local maxima. **(d)** The scenario where the input achieves zero value at one neuron. The green and red arrows are $\nabla_x^- f(x)$ and $\nabla_x^+ f(x)$ respectively.

It is similar to the definition in (Hanin & Rolnick, 2019) but with explicit indexing that will be important for defining operations on the activation patterns. For each input, the forward evaluation function has a fixed activation function, under which the network is reduced to an affine function for a small region in the input space that shares same activation pattern. Such regions are polytope, as they are defined by linear constraints on the activation units. The total number of such regions is the same as the possible activation patterns, which is exponential in the number of neurons. This combinatorially large space marks the difficulty of input optimization of neural networks, which is clearly NP-hard (Knuth, 1974; Van Leeuwen, 1991).

4. Problems with the Input Gradients

Since the input space is divided by the activation patterns into exponentially many polytope regions, the gradient of the network over the inputs $\nabla_x f(x)$ is determined by an affine function that is only valid for the small region around x. For illustrative purposes, Figure 1 (a) visualizes the activation patterns and the corresponding value landscape of a ReLU network with 3 hidden neurons and a scalar field as the output space. The input space is partitioned into 5 polyhedrons, each of which associates with one activation pattern. The network value f is convex with respect to the input for the inputs within one polyhedron. Consequently, when performing input optimization by following the input gradient direction, gradient information can produce false optimality analysis for the update steps that lead out of the polytope region corresponding to the activation pattern of x.

We re-illustrate the above insights with mathematical formulas. For an arbitrary input x, denote the input gradient as $\nabla_x f(x)$. Define α^* to be the distance from x to the closet polyhedron boundary along $\nabla_x f(x)$, that is the optimal step size for descending along $\nabla_x f(x)$ with improvement guarantee on the target objective. For all the step sizes α that is no larger than α^*, it should be the case that $f(x + \alpha^* \nabla_x f(x)) \geq f(x + \alpha \nabla_x f(x)) \geq f(x)$. However, as discussed above, $\nabla_x f(x)$ is not a reliable descent direction if the activation pattern of new input, i.e. $\tilde{x} = x + \alpha \nabla_x f(x)$, is different from that of x, in which case α must have a value strictly larger than α^*. Figure 1(c) demonstrates a scenario where applying standard gradient descent to perform input optimization converges to a local maxima. This is because the selected step size value is not large enough to escape the polytope regions that connect the local maxima. Increasing the value of step size is never a robust solution, which motivates the development of new techniques to derive better decent directions that look beyond the local regions.

Next, we discuss another typical failure scenario caused by the locality of standard gradient directions. On condition that the input variable attains zero value at one hidden neuron, that is when the input falls upon the polyhedron boundary, vanilla gradient descent rarely consider the decent directions which turn on that specific neuron. Figure 1(d) provides a scenario where a worse descent direction is selected due to the above issue even though there exists a clearly better gradient direction that leads to more improvement to the objective. Denote the gradient directions corresponding to having dead and alive neuron for the intersected hyperplane with $\nabla_x^- f(x)$ and $\nabla_x^+ f(x)$ respectively. The worse direction is selected by following input gradients if the new input led by $\nabla_x^- f(x)$, e.g. $f(x + \alpha \nabla_x^- f(x)))$, attains a lower objective value than that from $\nabla_x^- f(x)$ for a small-valued step size α. The exceptional case is when $\nabla_x^- f(x)$ induces a negative dot product with the normal vector of the hyperplane for that neuron. When optimizing for large networks, there can be an extremely large number of hyperplanes, each corresponding to one neuron, accounting for the finite input space. In that case, every step of gradient descent optimization can lead the input variable to be near the boundaries for many neurons. Therefore, this issue occurs frequently when optimizing large networks.

5. Activation-Descent Regularization

5.1. Augmenting Inputs with Activation Variables

From the analysis above, we see that the main problem of standard input gradients is that they cannot predict how the

activation patterns change even locally. Ideally, we would like to understand how the output of the network changes with respect to the change in the activation patterns, so that the optimization searches beyond the local polyhedron. Thus, the first step is to introduce new binary variables ν that can directly represent the activation patterns:

$$\nu = \{\nu^{(1)}, ..., \nu^{(l)}\} \in \{0,1\}^{\#\text{neurons}},$$
$$\nu^{(i)} = \{\nu_1^{(i)}, ..., \nu_{d_i}^{(i)}\} \in \{0,1\}^{d_i}, \;\; i \in [1, l].$$

Consider a rewriting of the network definition in Eq. (1) in the following form:

$$\begin{aligned} h^{(1)}(x,\nu) &= W_1 \cdot x + b_1, \\ h^{(i+1)}(x,\nu) &= W_{i+1} \cdot \text{diag}(\nu^{(i)}) \cdot h^{(i)}(x,\nu) + b_{i+1}, \\ & \qquad\qquad i \in [1, l], \\ \hat{f}(x,\nu) &= h^{(l+1)}(x,\nu), \end{aligned} \tag{3}$$

where $\text{diag}(\nu^{(i)}) \in \mathbb{R}^{d_i \times d_i}$ is the diagonal matrix of new variables $\nu^{(i)}$. It is clear that the function defined in (3) attains the same value as the neural network in Eq. (1) if ν variables satisfies:

$$\nu_j^{(i)} = \begin{cases} 1, & \text{if } h_j^{(i)}(x) \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

This is because when $h_j^{(i)}(x) \geq 0$,

$$\sigma(h_j^{(i)}(x)) = \nu_j^{(i)} h_j^{(i)}(x).$$

In fact, the $h^{(i)}$ vectors will have exactly the same value as the original network.

Proposition 5.1. *If x and ν satisfy constraints in (4), then $f(x) = \hat{f}(x,\nu)$.*

Now the newly parameterized function $\hat{f}(x,\nu)$ allows us to explicitly inspect how the changes in the activation patterns, captured by the ν variables, affect the changes in the output of the network. In fact we can derive the steepest descent direction in the discrete space of ν variables. This information can be obtained by the partial derivative of the output component $h_k^{(l+1)}(x,\nu)$ over the activation variable $\nu_j^{(i)}$, which assumes for the time-being that they take continuous values and can be differentiated. Then from the sign of the partial derivative, we need to check if $\nu_j^{(i)}$ can change its actual discrete value accordingly and stay within $\{0,1\}$. In other words, we project the gradient of the output component to the discrete domain of ν. Importantly, the projected direction does not need to be a feasible direction for the network, because we are treating ν as independent variables and have not taken into account of the feasibility constraints in Eq. (4), which will be handled later.

Definition 5.2 (Steepest Activation-Descent Direction)**.** Let $h_k^{(l+1)}$ be an arbitrary component of the output of the network $\hat{f}(x,v)$ defined in Eq (3). We define the projected descent direction of $h_k^{(l+1)}(x,\nu)$ on $\nu_j^{(i)}$, for non-zero $h_j^{(i)}(x,\nu)$, as follows:

$$\begin{aligned} \partial\nu_j^{(i)} = &\hat{\text{sgn}}(\nu_j^{(i)} - 0.5) \cdot \min\Big(\hat{\text{sgn}}\Big(\frac{\partial h_k^{(l+1)}(x,\nu)}{\partial \nu_j^{(i)}} \cdot \\ &\frac{1}{h_j^{(i)}(x,\nu)}\Big) \cdot \hat{\text{sgn}}(\nu_j^{(i)} - 0.5), 0\Big) \end{aligned} \tag{5}$$

where $\hat{\text{sgn}}(x) = 2 \cdot \text{sgn}(x) - 1$ applies sign operation into the set $\{-1, 1\}$. For the case $h_j^{(i)}(x,\nu) = 0$, we set $\partial\nu_j^{(i)} = 0$. Then the overall steepest descent direction in the discrete domain for ν is simply the vector consisting of all these components $\partial\nu = [\partial\nu_1^{(1)}, \cdots, \partial\nu_{d_l}^{(l)}]^T$. This direction is the steepest descent direction in the space of discrete values for ν because it is turning each component on or off based on whether they contribute positively or negatively to the output value.

However, the steepest activation-descent direction defined for $\hat{f}(x,\nu)$ is only valid if we consider ν as independent from x and ignore the constraints in Eq. (4). Indeed, it is easy to come up with networks that require large changes in x for the activation patterns ν to change, which will invalidate the local analysis based on gradients. Consequently, we want to capture the activation-descent direction in a continuous representation of the activation patterns. A natural idea is then to replace the discrete-valued activation variables ν by continuous activation functions such as the sigmoids, so that the activation patterns can be tuned continuously. However, if we make that change from the discrete values to the continuous ones, the overall output of the network will change in complex ways, because of the dependency across layers. It is thus crucial to define surrogate functions such that, even if the value of the surrogate function is different from the original function, the gradient over the new activation variables is consistent with the steepest direction in Definition 5.2. We achieve this by showing that the gradient of the surrogate function over the activation variables always forms a positive inner-product with the steepest direction in the discrete form (see Appendix B).

Definition 5.3 (Sigmoid-Surrogate Network)**.** Let $f(x)$ be a ReLU network defined in Eq. (1), and $\eta \in [0,1]^{\#\text{ neurons}}$ be the continuous representation of activation patterns. Using the same weight and bias parameters, we define its sigmoid-

surrogate network as:

$$\begin{aligned} \bar{h}^{(1)}(x,\eta) &= W_1 \cdot x + b_1 \\ \bar{h}^{(i+1)}(x,\eta) &= W_{i+1} \cdot \mathrm{diag}(s_\alpha(\eta^{(i)})) \cdot \bar{h}^{(i)}(x,\eta) \\ &\quad + b_{i+1}, \qquad i \in [1,l], \\ \bar{f}(x,\eta) &= \bar{h}^{(l+1)}(x,\eta) \end{aligned} \tag{6}$$

where $s_\alpha(x) = (1+\exp(-\alpha(x-\frac{1}{2})))^{-1}$ is the sigmoid function with an offset of $1/2$.

The continuous η variables should satisfy the corresponding feasibility constraints:

$$\eta_j^{(i)} : \begin{cases} \geq \frac{1}{2}, & \text{if } h_j^{(i)}(x) \geq 0, \\ < \frac{1}{2}, & \text{otherwise.} \end{cases} \tag{7}$$

Proposition 5.4. *For any x and η that satisfy the feasibility constraints in Eq. (7), the gradient on η defines descent directions of the original function.*

5.2. Optimization Objective

Next, we discuss the objective for optimizing x and η variables. Let $[x]_+ = \max(x,0)$ for $x \in \mathbb{R}$, to differentiate itself from the activation function σ. Given a l-layer ReLU network and an objective function J to *maximize*, we perform optimization to minimize the following objective L^*:

$$\begin{aligned} L_i(x,\eta) = \sum_{j=1}^{d_i} \Bigg(& \left[h_j^{(i)}(x)/\|P_j^{(i)}\| \cdot \left[\frac{1}{2} - \eta_j^{(i)}\right]_+ \right]_+ \\ & + \left[-h_j^{(i)}(x)/\|P_j^{(i)}\| \cdot \left[\eta_j^{(i)} - \frac{1}{2}\right]_+ \right]_+ \Bigg), \end{aligned} \tag{8}$$

$$L^*(x,\eta) = -J(\bar{h}^{(l+1)}(x,\eta)) + \beta \sum_{i=1}^{l} L_i(x,\eta). \tag{9}$$

where β is a scalar parameter, and $P_j^{(i)}$ is the j^{th} row vector of matrix $P^{(i)} \in \mathbb{R}^{d_i \times n}$ representing the hyperplane normal vectors for the neurons in the i^{th} layer of the sigmoid-surrogate network:

$$P^{(i)} = W_i \cdot \prod_{j=-(i-1)}^{-1} (\mathrm{diag}(s_\alpha(\eta^{(-j)})) \cdot W_{-j}). \tag{10}$$

The first term in (9) is the objective function applied to the output of sigmoid-surrogate network. The second term penalizes the misclassifications from η with respect to the ground-truth activation patterns. It is essential to apply normalization based on P, as the magnitude order of hidden neurons in ReLU networks varies drastically from layer to layer. Without normalization, constraint loss (8) from deeper layers dominates the second term in (9).

Next, we discuss the gradient of (9) on x in (11). The first term in (11) corresponds to the first term in (9). It approximates the input gradient from standard GD, i.e. $\partial J(f(x))/\partial x$, if the feasibility constraints in Eq. 7 are satisfied. In words, this term performs local search for better input values within the local polyhedron as vanilla GD. The second term in (11) corresponds to the constraint loss (8) in L^*, optimizing x towards the input polyhedron recognized by η. More specifically, the binary indicator functions (i.e. the sgn functions) check the inconsistency between what η captures and the activation patterns of the original network, i.e. $f(x)$ instead of $\bar{f}(x,\eta)$. For one specific neuron, at most one of the two indicators can be triggered, which indicates the inconsistency at that neuron. In that case, the gradient of (8) optimizes x along the hyperplane normal vector of that particular neuron until the input variable reaches the boundary and thus flips the activation pattern. Intuitively, this term is important in searching more global descent directions that lead x to the potentially better polyhedron regions based on the optimization of η.

$$\begin{aligned} \nabla_x L^* = & \frac{\partial J(\bar{h}^{(l+1)}(x,\eta))}{\partial \bar{h}^{(l+1)}(x,\eta)} \cdot P^{(l+1)} + \beta \sum_{i=1}^{l} \sum_{j=1}^{d_i} \Bigg[\\ & \mathrm{sgn}\Big(h_j^{(i)}(x) \cdot \left[\frac{1}{2} - \eta_j^{(i)}\right]_+\Big) \cdot \frac{\partial h_j^{(i)}(x,\eta)}{\partial x} / \|P_j^{(i)}\| \\ & - \mathrm{sgn}\Big(-h_j^{(i)}(x) \cdot \left[\eta_j^{(i)} - \frac{1}{2}\right]_+\Big) \cdot \frac{\partial h_j^{(i)}(x,\eta)}{\partial x} / \|P_j^{(i)}\| \Bigg] \end{aligned} \tag{11}$$

Lastly, we discuss the gradient of constraint loss (8) on η. Without loss of generality, we focus on one arbitrary pattern variable $\eta_j^{(i)}$:

$$\frac{\partial L_i}{\partial \eta_j^{(i)}} = \hat{\mathrm{sgn}}\Big(h_j^{(i)}(x)\Big) \cdot \mathrm{sgn}\Big(h_j^{(i)}(x) \cdot (\frac{1}{2} - \eta_j^{(i)})\Big) \cdot \frac{h_j^{(i)}(x)}{\|P_j^{(i)}\|}.$$

This term optimizes η to satisfy the feasibility constraints in (7) with respect to x. If there is detected inconsistency between η and the ground-truth activation pattern of x, this gradient optimizes η to match the correct activation pattern. The division with $\|P_j^{(i)}\|$ applies normalization to derive the normalized distance from x to the hyperplane of one arbitrary neuron. We emphasize that the gradients of the first term in (9) on η are not discarded. As these gradients capture the descent directions of the objective on η (Proposition 5.4), they can search for the promising changes to the activation patterns that locally optimize the overall objective despite introducing mismatch between the current input and the target pattern assignments (Figure 2). Optimizing the feasibility constraint (8) helps to enforce consistency between the input variable and the targeted activation patterns.

5.3. Overall Algorithm

We provide the complete procedures in Algorithm 1, in which Perturbed Gradient Descent (Perturbed GD) (Jin et al., 2017) is employed over η variables. At line 2, we initialize η to match the activation patterns of the initial input variable. Perturbations are applied to η only when the gradient norm of L^* on η is lower than the threshold δ (line 8). Intuitively,

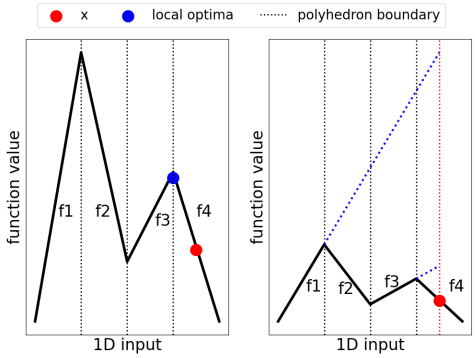


Figure 2: Toy example to illustrate the intuition behind optimizing activation pattern variables η. We consider to maximize a 1-dimensional piecewise-linear function where each linear segments (e.g. f_1-f_4 functions) corresponds to one activation pattern. **Left**: Applying vanilla GD at x quickly gets stuck at the nearby local optima. **Right**: Optimizing (9) over η identifies promising changes to η that *locally* optimizes the overall objective. In the above example, the activation pattern underneath f_1 will be targeted, as $f_1(x) > f_3(x) > f_4(x)$, despite the inconsistency between the target activation pattern and the current input. Optimizing (8) helps to correct such inconsistencies.

randomized perturbations are not helpful unless the variables are currently close to some potential saddle points. We apply perturbations at most once every T_p iterations. We do not apply perturbations to x as the input gradient of the first term in (9) is piecewise and discontinuous. Therefore, if Perturbed GD is deployed for our problem, it applies perturbations as long as the objective landscape is flat enough within the current input polyhedron, regardless of the input being around saddle points or not. At lines 15-18, we adjust the coefficient parameter β based on its gradient of L^* which essentially is the non-negative constraint loss (8). If the constraint loss has a trivial value, i.e. its norm being less than the tolerance δ_β, we linearly decay β by γ to mitigate the weighing of constraint loss.

6. Experiments

6.1. Adversarial Optimization

We evaluate the proposed method in constructing adversarial examples (Goodfellow et al., 2014) for neural image classifiers. Consider a neural image classifier $C : \mathbb{R}^n \to \mathbb{R}^m$, a classification loss $J : \mathbb{R}^m \to \mathbb{R}$, and a perturbation size ϵ. For an input image $x \in \mathbb{R}^n$, we derive the perturbation δ that leads the perturbed image to maximize the loss J within the ϵ-neighborhood around x, i.e., $\arg\max_{\|\delta\| \le \epsilon} J(C(x+\delta))$. Two types of adversarial attack are considered: *untargeted* attacks try to misguide the classifier to predict any of the incorrect classes, while *targeted* attacks aim for a particular class. J is cross-entropy loss with respect to true label for untargeted attacks, and the negation of cross-entropy loss

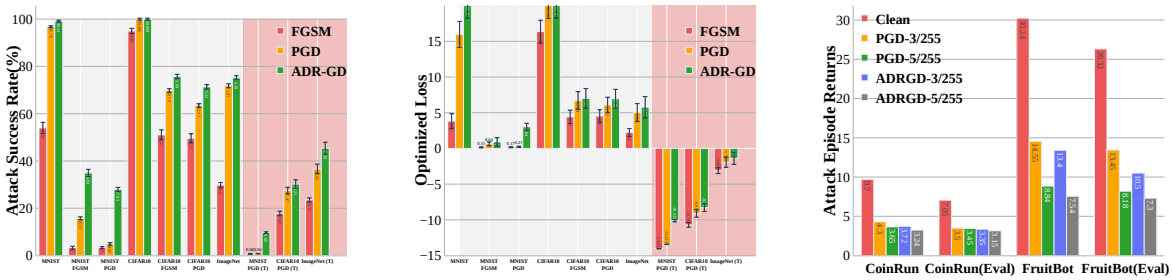
Algorithm 1 Activation-Descent Regularization GD (ADR-GD)

1: **Input** l-layer ReLU network f, total iterations T, initial coefficient β_0, step sizes (α_x, α_η, α_β), perturbation scale r, perturbation frequency T_p, coefficient decay rate γ, gradient tolerances (δ, δ_β)
2: **Initialize** x and $\eta^{(1)}, ..., \eta^{(l)}$
3: **Initialize** $\beta \leftarrow \beta_0$
4: $t_{noise,i} \leftarrow 1$ **for** i = 1, 2, ..., l
5: **for** $t = 1, 2, ..., T$ **do**
6: $\quad x \leftarrow x - \alpha_x \cdot \nabla_x L^*$
7: $\quad$ **for** i = 1, 2, ..., l **do**
8: $\quad\quad$ **if** $t - t_{noise,i} \ge T_p$ and $\|\nabla_{\eta^{(i)}} L^*)\| \le \delta$ **then**
9: $\quad\quad\quad \eta^{(i)} \leftarrow \eta^{(i)} - \alpha_\eta \cdot (\nabla_{\eta^{(i)}} L^* + r \cdot \xi_t)$
10: $\quad\quad\quad t_{noise,i} \leftarrow t$
11: $\quad\quad$ **else**
12: $\quad\quad\quad \eta^{(i)} \leftarrow \eta^{(i)} - \alpha_\eta \cdot \nabla_{\eta^{(i)}} L^*$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: $\quad$ **if** $\|\nabla_\beta L^*\| \le \delta_\beta$ **then**
16: $\quad\quad \beta \leftarrow \beta - \gamma$
17: $\quad$ **else**
18: $\quad\quad \beta \leftarrow \beta + \alpha_\beta \cdot \nabla_\beta L^*$
19: $\quad$ **end if**
20: **end for**
21: **Return** x

with respect to target label for targeted attacks. We compare the proposed method to Fast Gradient Sign Method (FGSM) (Goodfellow et al., 2014) and Projected Gradient Descent (PGD) (Madry et al., 2017), two gradient-based attack methods that leverage the signs of input gradients instead of strictly following the steepest descent directions.

We evaluate the proposed algorithm on MNIST (Deng, 2012), CIFAR10 (Krizhevsky et al., 2009), and ImageNet (Deng et al., 2009) datasets. For MNIST and CIFAR10 datasets, in addition to the standard classification model trained over clean images, we consider the robust models from adversarial training (Shafahi et al., 2019) that use FGSM or PGD methods to generate adversarial examples as supplementary training data. We employ small-sized Convolutional neural networks for MNIST, while VGG19 (Sengupta et al., 2019) for CIFAR10. For ImageNet, we use the pre-trained Wide ResNet-50-2 model (Zagoruyko & Komodakis, 2016) downloaded from (TorchVision, 2016). When dealing with large networks, we can apply the proposed method to partial networks selectively. As an example, only the pattern variables over the last layer of Wide ResNet-50-2 model are optimized, while we ignore the neurons from earlier layers just as in vanilla GD.

Figure 3 (a)-(b) summarize the experiment results. The perturbation size ϵ is set to 0.2, $8/255$, and $2/255$ for the



(a) Attack success rates (b) Optimized objective values (c) Episode returns

Figure 3: **(a)-(b)** Adversarial optimization experiments in Section 6.1 and 6.3. Grey and pink shading indicate the untargeted and targeted experiments respectively. **(c)** Attacks are constructed to worsen the performance of well-trained neural policies in FruitBot and CoinRun.

	Per-iteration runtime (sec)		Variable counts	
Dataset	PGD	ADR-GD	PGD	ADR-GD
MNIST	$6.22e-4$	$1.08e-2$	784	9180
CIFAR10	$1.07e-3$	$1.36e-2$	$3,072$	$23,552$
ImageNet	$6.58e-3$	$4.17e-2$	$150,528$	$351,232$

(a) Runtime benchmarks

Model A	[[10, 64], [64, 64], [64, 1]]
Model B	[[10, 500], [500, 500], [500, 500], [500, 1]]
Model C	[[128, 500], [500, 500], [500, 500] [500, 1]]

(b) Model architecture

	Model A	Model B	Model C
GD	40.58	3873.76	35807.10
Adam	41.17	3914.07	36531.20
Adagrad	41.72	3884.90	35513.52
Perturbed GD	40.62	3903.20	35964.02
ADR-GD (ours)	**43.95**	**4047.99**	**37672.04**
M1	43.33	3914.25	36609.58
M2	43.74	3940.10	36822.27
M3	43.90	3995.62	36064.52
M4	43.13	3882.18	36036.73

(c) Optimize objective values

Table 1: **(a)** Average *per-iteration* runtime benchmarks for experiments in Section 6.1. **(b)-(c)** Model architectures and results for ablation experiments in Section 6.4. For instance, Model A is a 3-layer ReLU network with the input dimension $d_n = 10$, the latent dimensions $d_1 = d_2 = 64$, and the output dimension $d_m = 1$.

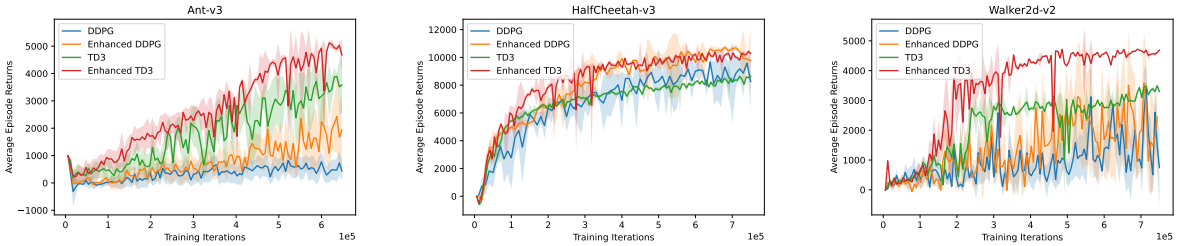
evaluations in MNIST, CIFAR10 and ImageNet respectively. When evaluating PGD, we allow it to run for a sufficient number of iterations to ensure convergence. As shown, the proposed method achieves better attack success-rates and higher optimized loss than PGD and FGSM in all cases. For all three datasets, PGD delivers comparable performance with the proposed method over the naturally trained models. Meanwhile, ADR-GD significantly outperforms PGD on robust models. For instance, the adversarial training strategy greatly improves the model robustness against PGD attacks on MNIST, reducing the attack success rate of PGD from 96.78% to 4.92%. In comparison, the proposed method achieves 27.93% attack success rate on the robust model. Moreover, we demonstrate the effectiveness of the proposed method for constructing targeted attacks in Figure 3.

Table 1 (a) provides the average runtime benchmarks for experiments on different datasets. It shows that the average per-iteration runtime from ADR-GD is slower by nearly an order of magnitude than that from PGD. This is because the proposed method optimizes more variables, i.e., activation variables η, than PGD. Despite the computation overheads from our implementations, we show that computation cost for the proposed algorithm does not grow exponentially with the increase in the network size.

6.2. Deep Reinforcement Learning

We enhance the training of Deep Reinforcement Learning (DRL) by formulating input optimization problems to compute action refinements during exploitation. We consider the DRL algorithms that are based on actor-critic structure such as DDPG (Lillicrap et al., 2015) and TD3 (Fujimoto et al., 2018). As a motivation, one well-known challenge in these algorithms are to ensure the optimal coupling between actor and critic, i.e., actor model properly captures the actions maximizing the approximated Q-values from critic model.

Next, we discuss the formulation of input optimization problem. Consider the actor model $\pi : \mathbb{R}^n \to \mathbb{R}^m$ that maps the observation states to the actions, and the critic model $V : \mathbb{R}^{n+m} \to \mathbb{R}$ that approximates the optimal Q-values from state-action pairs. We do not modify the training objective of model parameters. Instead, we compute action refinements by optimizing around the action candidates from actor to maximize the output of critic. For an arbitrary state



(a) Ant-v3 (b) HalfCheetah-v3 (c) Walker2d-v2

Figure 4: Enhanced DRL training in three mujoco environments

$s \in \mathbb{R}^n$ and a probability threshold $\epsilon \in [0, 1]$, we obtain the exploitation action a as the following:

$$a = \begin{cases} \pi(s) & \text{with probability } 1-\epsilon, \\ \pi(s) + \arg\max_{(\partial)\in\mathbb{R}^m} V(s, \pi(s)+\partial) & \text{otherwise.} \end{cases} \tag{12}$$

Figure 4 demonstrates the experiments on several mujoco environments (Todorov et al., 2012). For each testing environment, we apply the proposed enhancement procedures to both DDPG and TD3 algorithms, and use $\epsilon = 0.25$ for the enhanced algorithms. We observe robust improvements from the enhanced methods. Many prior works reported that a DDPG agent cannot be trained to exceed more than $1,000$ average episode rewards in Ant environment (Fujimoto et al., 2018; Dankwa & Zheng, 2019). In Figure 4, we show that the enhanced DDPG approaches nearly $2,000$ average episode rewards within limited training iterations.

6.3. Adversarial Optimization against Neural Policies

Adversarial attacks to neural policies can be constructed via input optimization. We consider the environments that have imagery observation space, and discrete action space as in (Oikarinen et al., 2021). We perform optimization in the observation space, i.e. the policy model's input space, so that non-optimal actions will be produced. For an arbitrary state $s\in \mathbb{R}^n$, a policy model $\pi : \mathbb{R}^n \to \mathbb{R}^m$ and a classification loss $J : \mathbb{R}^m \to \mathbb{R}$ that is minimized at the (sub-)optimal action $\pi(s)$, we derive the perturbed state within the ϵ-neighborhood of x to maximize J, i.e. $\arg\max_{\|\partial\|\leq\epsilon} J(\pi(s+\partial))$.

Figure 3(c) demonstrates the evaluations in two Procgen environments (Cobbe et al., 2019), FruitBot and CoinRun. We experiment with two attack sizes, i.e., $\epsilon = 3/255$ and $\epsilon = 5/255$. Both PGD and the proposed method can construct impactful attacks that sharply undermine the performance of well-trained policies. Nevertheless, ADR-GD robustly leads to more significant reductions in average episode rewards than PGD. Note that in CoinRun environment, the attacks from the proposed method constrained with $\epsilon = 3/255$ are as strong as the PGD attacks constrained with $\epsilon = 5/255$.

6.4. Ablation Study

We perform ablation experiment to illustrate the importance of each components in Algorithm 1.We test with randomized neural networks that produce scalar outcomes. Model parameters are uniformly sampled from range $(-1, 1)$. Denote one randomized network with $F : \mathbb{R}^n \to \mathbb{R}$. The problem objective is simply to maximize the network's scalar output, i.e., $\max_{x\in[-1,1]^n} F(x)$.

Table 1(b) provides the model architectures of testing models. To understand how each component contributes to Algorithm 1, we experiment with different settings as: **M1** - detaching the gradient of $J(\bar{h}^{(l+1)}(x, \eta))$ on η, **M2** - detaching the gradient of $\beta\sum_{i=1}^{n-1} L_i$ on x, **M3** - removing normalization, and **M4** - removing perturbation. Table 1(c) shows the average objective value $F(x^*)$ where x^* is the optimized input variable. The proposed method with all the components enabled always achieves the highest $F(x^*)$. Decays in the optimized value are observed when we detach the gradients of partial objectives (e.g. **M1** and **M2**). We note that **M3** delivers comparable performance with the proposed method for small networks. It is reasonable as for small networks, the magnitude order of hidden neurons may not differ much at different layers, and thus normalization is not necessary. The performance of Perturbed GD is limited as expected, due to the piecewise gradient landscape in deep ReLU networks. Nevertheless, it is demonstrated the improvement in the proposed algorithm led by introducing randomized perturbations to the activation pattern variables.

7. Conclusion

We present a novel optimization procedure for input optimization of ReLU Networks that explicitly takes into account the impact of changes in activation patterns on the output. We introduced new regularization terms for local descent in the input space that encourages the input change

to be aligned with the descent directions in the activation space. The overall procedures can thus achieve better local descent properties than GD and also various forms of randomly perturbed gradient methods. We observed that the proposed methods improve the state-of-the-art results from prior gradient-based optimization methods in various application problems.

Future work in this direction can involve further theoretical analysis for the convergence complexity and improving scalability to larger networks. In addition, we believe the proposed method offers useful insights into optimizing non-ReLU networks whose value landscape is piecewise-continuous, rather than piecewise-linear as in ReLU networks. Therefore, these networks also suffer from the same issues discussed in Section 4, when following the standard gradient descent directions over their non-convex landscape.

Acknowledgements

We thank the anonymous reviewers for their valuable suggestions. This material is based on work supported by NSF Career CCF 2047034, NSF CCF 2112665 TILOS AI Institute, NSF CCF DASS 2217723, and Amazon Research Award.

Impact Statement

This paper presents work aiming at solving input optimization problems for ReLU networks. The negative impact of it is the potential use that perform adversarial optimizations to intentionally attack or disturb well-trained ReLU network models for vicious purposes. Nevertheless, the development of any adversarial algorithms is to achieve a truly robust neural models that can defend any attacks. The optimization criteria of the proposed method takes into consideration the combinatorial nature underneath ReLU networks, which we believe can be related to understanding the expressivity and generalizability of ReLU networks in the future.

References

Agarwal, N., Allen-Zhu, Z., Bullins, B., Hazan, E., and Ma, T. Finding approximate local minima for nonconvex optimization in linear time. *arXiv preprint arXiv:1611.01146*, 2016.

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *International Conference on Machine Learning*, pp. 146–155. PMLR, 2017.

Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

Bojanowski, P., Joulin, A., Lopez-Paz, D., and Szlam, A. Optimizing the latent space of generative networks. *arXiv preprint arXiv:1707.05776*, 2017.

Bora, A., Jalal, A., Price, E., and Dimakis, A. G. Compressed sensing using generative models. In *International Conference on Machine Learning*, pp. 537–546. PMLR, 2017.

Bunel, R. R., Turkaslan, I., Torr, P., Kohli, P., and Mudigonda, P. K. A unified view of piecewise linear neural network verification. *Advances in Neural Information Processing Systems*, 31, 2018.

Carmon, Y. and Duchi, J. C. Gradient descent efficiently finds the cubic-regularized non-convex newton step. *arXiv preprint arXiv:1612.00547*, 2016.

Carmon, Y., Duchi, J. C., Hinder, O., and Sidford, A. Accelerated methods for nonconvex optimization. *SIAM Journal on Optimization*, 28(2):1751–1772, 2018.

Chang, Y.-C., Roohi, N., and Gao, S. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

Cohen, J., Rosenfeld, E., and Kolter, Z. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning*, pp. 1310–1320. PMLR, 2019.

Croce, F. and Hein, M. Sparse and imperceivable adversarial attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4724–4732, 2019.

Curtis, F. E., Robinson, D. P., and Samadi, M. A trust region algorithm with a worst-case iteration complexity for nonconvex optimization. *Mathematical Programming*, 162(1):1–32, 2017.

Dankwa, S. and Zheng, W. Twin-delayed ddpg: A deep reinforcement learning technique to model a continuous movement of an intelligent robot agent. In *Proceedings of the 3rd international conference on vision, image and signal processing*, pp. 1–5, 2019.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

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

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Fujimoto, S., Meger, D., and Precup, D. Off-policy deep reinforcement learning without exploration. In *International conference on machine learning*, pp. 2052–2062. PMLR, 2019.

Ge, R., Huang, F., Jin, C., and Yuan, Y. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on learning theory*, pp. 797–842. PMLR, 2015.

Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Guo, X., Han, J., and Tang, W. Perturbed gradient descent with occupation time. *arXiv preprint arXiv:2005.04507*, 2020.

Gurumurthy, S., Bai, S., Manchester, Z., and Kolter, J. Z. Joint inference and input optimization in equilibrium networks. *Advances in Neural Information Processing Systems*, 34:16818–16832, 2021.

Hanin, B. and Rolnick, D. Deep relu networks have surprisingly few activation patterns. *Advances in neural information processing systems*, 32, 2019.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., and Abbeel, P. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

Ilahi, I., Usama, M., Qadir, J., Janjua, M. U., Al-Fuqaha, A., Hoang, D. T., and Niyato, D. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *IEEE Transactions on Artificial Intelligence*, 3 (2):90–109, 2021.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32, 2019.

Jin, C., Ge, R., Netrapalli, P., Kakade, S. M., and Jordan, M. I. How to escape saddle points efficiently. In *International Conference on Machine Learning*, pp. 1724–1732. PMLR, 2017.

Knuth, D. E. Postscript about np-hard problems. *ACM SIGACT News*, 6(2):15–16, 1974.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Levy, K. Y. The power of normalization: Faster evasion of saddle points. *arXiv preprint arXiv:1611.04831*, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Makkuva, A., Taghvaei, A., Oh, S., and Lee, J. Optimal transport mapping via input convex neural networks. In *International Conference on Machine Learning*, pp. 6672–6681. PMLR, 2020.

Nesterov, Y. and Polyak, B. T. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

Oikarinen, T., Zhang, W., Megretski, A., Daniel, L., and Weng, T.-W. Robust deep reinforcement learning through adversarial loss. *Advances in Neural Information Processing Systems*, 34:26156–26167, 2021.

Ongie, G., Jalal, A., Metzler, C. A., Baraniuk, R. G., Dimakis, A. G., and Willett, R. Deep learning techniques for inverse problems in imaging. *IEEE Journal on Selected Areas in Information Theory*, 1(1):39–56, 2020.

Pattanaik, A., Tang, Z., Liu, S., Bommannan, G., and Chowdhary, G. Robust deep reinforcement learning with adversarial attacks. *arXiv preprint arXiv:1712.03632*, 2017.

Rick Chang, J., Li, C.-L., Poczos, B., Vijaya Kumar, B., and Sankaranarayanan, A. C. One network to solve them all–solving linear inverse problems using deep projection models. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5888–5897, 2017.

Sengupta, A., Ye, Y., Wang, R., Liu, C., and Roy, K. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

Shafahi, A., Najibi, M., Ghiasi, M. A., Xu, Z., Dickerson, J., Studer, C., Davis, L. S., Taylor, G., and Goldstein, T. Adversarial training for free! *Advances in Neural Information Processing Systems*, 32, 2019.

Shi, Z., Jin, Q., Zhang, H., Kolter, Z., Jana, S., and Hsieh, C.-J. Formal verification for neural networks with general nonlinearities via branch-and-bound. In *2nd Workshop*

on Formal Verification of Machine Learning (WFVML 2023), 2023.

Taylor, A., Singletary, A., Yue, Y., and Ames, A. Learning for safety-critical control with control barrier functions. In *Learning for Dynamics and Control*, pp. 708–717. PMLR, 2020.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033. IEEE, 2012.

TorchVision. Torchvision: Pytorch's computer vision library. https://github.com/pytorch/vision, 2016.

Tripuraneni, N., Stern, M., Jin, C., Regier, J., and Jordan, M. I. Stochastic cubic regularization for fast nonconvex optimization. *Advances in neural information processing systems*, 31, 2018.

Van Leeuwen, J. *Handbook of theoretical computer science (vol. A) algorithms and complexity*. Mit Press, 1991.

Wong, E. and Kolter, Z. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pp. 5286–5295. PMLR, 2018.

Xu, H., Ma, Y., Liu, H.-C., Deb, D., Liu, H., Tang, J.-L., and Jain, A. K. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17(2):151–178, 2020.

Yang, L., Dai, H., Shi, Z., Hsieh, C.-J., Tedrake, R., and Zhang, H. Lyapunov-stable neural control for state and output feedback: A novel formulation for efficient synthesis and verification. *arXiv preprint arXiv:2404.07956*, 2024.

Yang, X., Yamaguchi, T., Tran, H.-D., Hoxha, B., Johnson, T. T., and Prokhorov, D. Neural network repair with reachability analysis. In *Formal Modeling and Analysis of Timed Systems: 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13–15, 2022, Proceedings*, pp. 221–236. Springer, 2022.

Yu, H., Hirayama, C., Yu, C., Herbert, S., and Gao, S. Sequential neural barriers for scalable dynamic obstacle avoidance. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 11241–11248. IEEE, 2023.

Zadeh, A., Lim, Y.-C., Liang, P. P., and Morency, L.-P. Variational auto-decoder. 2019.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, H., Wang, S., Xu, K., Wang, Y., Jana, S., Hsieh, C.-J., and Kolter, Z. A branch and bound framework for stronger adversarial attacks of relu networks. In *International Conference on Machine Learning*, pp. 26591–26604. PMLR, 2022.

A. Approximating Piecewise Constant Function with Sigmoid

In this section, we discuss the impact of the modified sigmoid used in the sigmoid-surrogate network (6). Recall that we employ a following variant of sigmoid function with an offset of $1/2$:

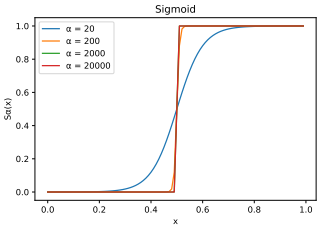
$$s_\alpha(x) = \frac{1}{1 + \exp(-\alpha(x - \frac{1}{2}))},$$

where α is the hyperparameter that determines how fast the function value grows from 0 to 1. At high level, we introduce sigmoid functions to relax the binary activation variables, i.e. ν in (3), into continuous representations, i.e. η in (6). Next, we illustrate how the approximation errors from the proposed relaxation does not worsen the optimization on input variables, as long as the value of hyperparameter α is well selected.

First, we briefly revisit the ReLU operator, and rewrite its procedures as follows:

$$\begin{aligned}\sigma(x) &= \max(0, 1) \\ &= \mathbb{1}(x > 0) \cdot x\end{aligned}$$

which shows that the ReLU operator essentially derives a binary coefficient, i.e. $\mathbb{1}(x > 0)$, whose value depends on if x is positive or not.



	Model A	Model C
GD	40.58	35807.10
Ours, $\alpha = 20$	25.84	19942.94
Ours, $\alpha = 200$	43.75	36219.05
Ours, $\alpha = 2000$	43.95	**37672.04**
Ours, $\alpha = 20000$	**44.12**	37054.31

Figure 5: **Left**: Visualization of sigmoid functions under different values of α.**Right**: The average *optimized* objective values for the models in Table 1.

Figure 5(Left) visualizes a 1-D sigmoid function within input range $[0, 1]$, using different values of α. It shows that as the value of α arises, $s_\alpha(x)$ approaches a piecewise constant function bounded between $[0, 1]$, just like the ReLU operator. An overlarge value of α may undermine our incentive to relax the ReLU operator continuously, while an oversmall α value introduces high approximation errors with respect to the ReLU operator. Figure 5(Right) demonstrates the ablation experiment with varying values of α. We evaluate on randomized neural networks that follow the same architectures with those in Table 1(a). For the experiment results reported in main paper, we employ $\alpha = 2000$.

B. Proof of Proposition 5.4

In this section, we show that the gradient of the output of sigmoid-surrogate network in (6) over the activation variables η is consistent with the steepest direction in Definition 5.2. We achieve this by showing that the gradient of the surrogate function over η always forms a positive inner-product with the steepest direction in the discrete form.

Proof. Let $\bar{h}_k^{(i+1)}(x, \eta)$ be an arbitrary component of the sigmoid-surrogate network defined in (6). We derive the gradient of $\bar{h}_k^{(i+1)}(x, \eta)$ on $\eta_j^{(i)}$ as:

$$\partial\eta_j^{(i)} = \frac{\partial\bar{h}_k^{(i+1)}(x, \eta)}{\partial s_\alpha(\eta_j^{(i)})} \cdot \frac{1}{\bar{h}_j^{(i)}(x, \eta)} \cdot s_\alpha(\eta_j^{(i)}) \cdot (1 - s_\alpha(\eta_j^{(i)})) \cdot \alpha \tag{13}$$

Next, we show that $\partial\eta_j^{(i)}$ always forms a positive product with $\partial\nu_j^{(i)}$ defined in (5.2). If $\partial\nu_j^{(i)} = 0$, then $\partial\nu_j^{(i)} \cdot \partial\eta_j^{(i)} \geq 0$

must be trivially true. If $\partial\nu_j^{(i)} \neq 0$ and $\nu_j^{(i)} = 1$, then:

$$\partial\nu_j^{(i)} = \hat{\text{sgn}}\left(\frac{\partial h_k^{(l+1)}(x,\nu)}{\partial\nu_j^{(i)}} \cdot \frac{1}{h_j^{(i)}(x,\nu)}\right)$$

As discussed in Section A, $s_\alpha(\eta_j^{(i)})$ and $\bar{h}_j^{(i)}(x,\eta)$ closely approach $\nu_j^{(i)}$ and $h_j^{((i))}(x,\nu)$ respectively, as long as $\eta_j^{(i)}$ and $\nu_j^{(i)}$ both satisfy the feasibility constraints in (4) and (7). In fact, it must hold true that:

$$\hat{\text{sgn}}(\partial\eta_j^{(i)}) = \partial\nu_j^{(i)}$$

This is because $s_\alpha(\eta_j^{(i)}) \cdot (1 - s_\alpha(\eta_j^{(i)}))$ must always yield a non-negative value for all possible values of $\eta_j^{(i)}$. Thus it must be true that $\partial\nu_j^{(i)} \cdot \partial\eta_j^{(i)} \geq 0$. The case for $\nu_j^{(i)} = 0$ can be proved similarly. □

C. Experiment Parameters

Note that we employ gradient normalization in our implementations for optimizing x and η, in case of any concerns about the high learning-rates.

	MNIST	CIFAR10	ImageNet
T	250	500	500
β_0	2.	0.5	0.5
α_x	1.	0.1	0.1
α_σ	0.25	0.5	0.5
α_β	0.001	0.001	0.001
r	0.1	0.2	0.1
δ	0.0001	0.001	0.01
δ_β	0.1	0.1	0.05
T_p	5	5	5

(a) Untargeted attacks in Section 6.1

	MNIST	CIFAR10	ImageNet
T	250	500	500
β_0	2.	1.	0.5
α_x	0.5	0.1	0.1
α_σ	0.1	0.5	0.5
α_β	0.001	0.001	0.0001
r	0.01	0.1	0.01
δ	0.0001	0.001	0.01
δ_β	0.1	0.05	0.05
T_p	5	5	5

(b) Targeted attacks in Section 6.1

	DDPG	TD3
T	100	100
β_0	0.5	0.5
α_x	0.2	0.2
α_σ	0.1	0.1
α_β	0.01	0.01
r	0.05	0.05
δ	0.001	0.001
δ_β	0.1	0.1
$T_{perturb}$	10	10

(c) Enhanced DRL training in Section 6.2

	FruitBot	CoinRun
T	100	100
β_0	0.5	0.5
α_x	0.4	0.4
α_σ	0.25	0.25
α_β	0.001	0.001
r	0.01	0.01
δ	0.0001	0.0001
δ_β	0.1	0.1
$T_{perturb}$	10	10

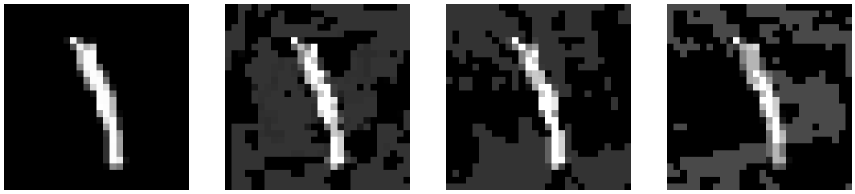
(d) Attacking DRL policies in Section 6.3

	Model A	Model B	Model C
T	3000	3000	3000
β_0	1.	1.	1.
α_x	0.5	0.5	0.5
α_σ	1.	1.	1.25
α_β	0.01	0.005	0.0001
r	0.1	0.2	0.35
δ	0.0001	0.001	0.01
δ_β	0.001	0.01	0.01
T_p	25	25	25

(e) Ablation in Section 6.4

Table 2: Training parameters

D. Visualizations of the Perturbed Images

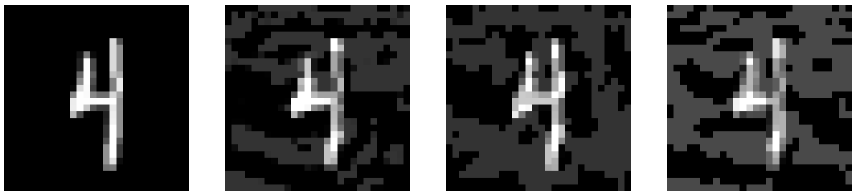


(a) Clean
Label: 1
Loss: 0.003

(b) PGD
Label: 1
Loss: 0.021

(c) FGSM
Label: 1
Loss: 0.011

(d) ADR-GD
Label: 5
Loss: 22.030



(a) Clean
Label: 4
Loss: 0.000

(b) PGD
Label: 5
Loss: 32.246

(c) FGSM
Label: 9
Loss: 7.430

(d) ADR-GD
Label: 3
Loss: 64.006

(a) Clean
Label: Horse
Loss: 0.495

(b) PGD
Label: Airplane
Loss: 2.840

(c) FGSM
Label: Dog
Loss: 1.993

(d) ADR-GD
Label: Airplane
Loss: 3.014

(a) Clean
Label: Ship
Loss: 1.008

(b) PGD
Label: Airplane
Loss: 3.422

(c) FGSM
Label: Airplane
Loss: 3.659

(d) ADR-GD
Label: Airplane
Loss: 4.776

(a) Clean
Label: Automobile
Loss: 0.000

(b) PGD
Label: Automobile
Loss: 0.188

(c) FGSM
Label: Automobile
Loss: 0.022

(d) ADR-GD
Label: Bird
Loss: 0.842

(a) Clean
Label: Scuba Diver
Loss: 0.7044

(b) PGD
Label: Scuba Diver
Loss: 1.7802

(c) FGSM
Label: Scuba Diver
Loss: 1.1360

(d) ADR-GD
Label: Snorkel
Loss: 2.3171

(a) Clean
Label: Alsatian
Loss: 0.7411

(b) PGD
Label: Alsatian
Loss: 1.9595

(c) FGSM
Label: Alsatian
Loss: 0.8340

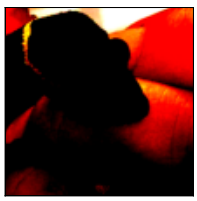
(d) ADR-GD
Label: Malinois
Loss: 2.5061

(a) Clean
Label: Llama
Loss: 1.7115

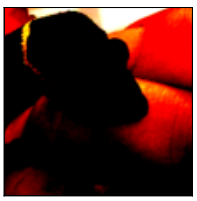
(b) PGD
Label: White Shark
Loss: 4.5178

(c) FGSM
Label: White Shark
Loss: 3.6784

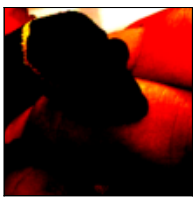
(d) ADR-GD
Label: White Shark
Loss: 6.0331



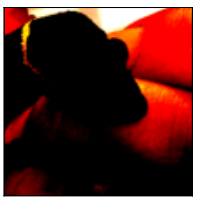
(a) Clean
Label: Banded Gecko
Loss: 0.7323



(b) PGD
Label: Schipperke
Loss: 8.5483



(c) FGSM
Label: Schipperke
Loss: 8.0851



(d) ADR-GD
Label: Schipperke
Loss: 9.0483

(a) Clean
Label: Upright Piano
Loss: 0.8907

(b) PGD
Label: Grand Piano
Loss: 2.1140

(c) FGSM
Label: Upright Piano
Loss: 1.4467

(d) ADR-GD
Label: Grand Piano
Loss: 2.4814

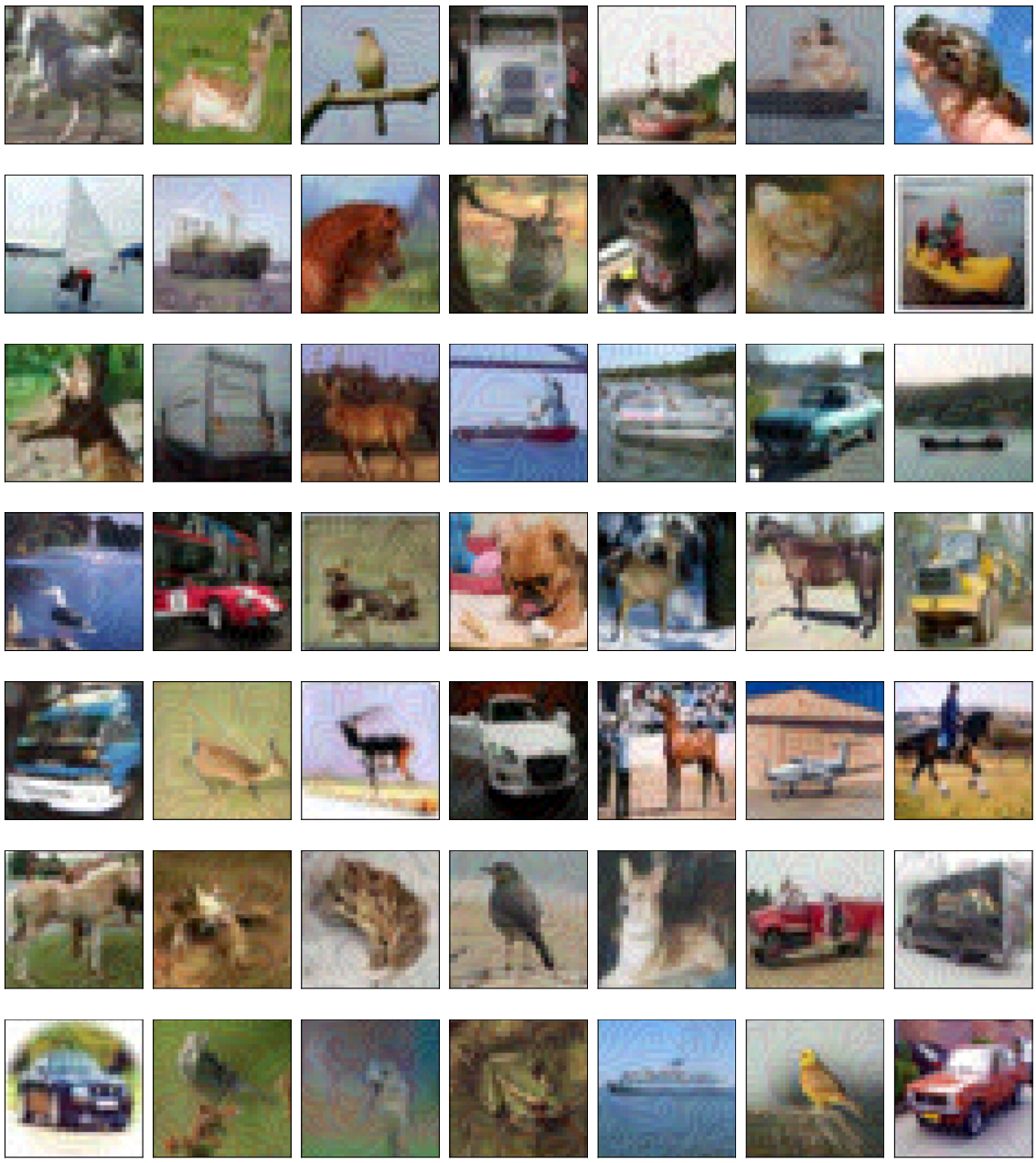


Figure 6: More adversarial examples constructed by the proposed method for CIFAR10, with $\epsilon = 8/255$.

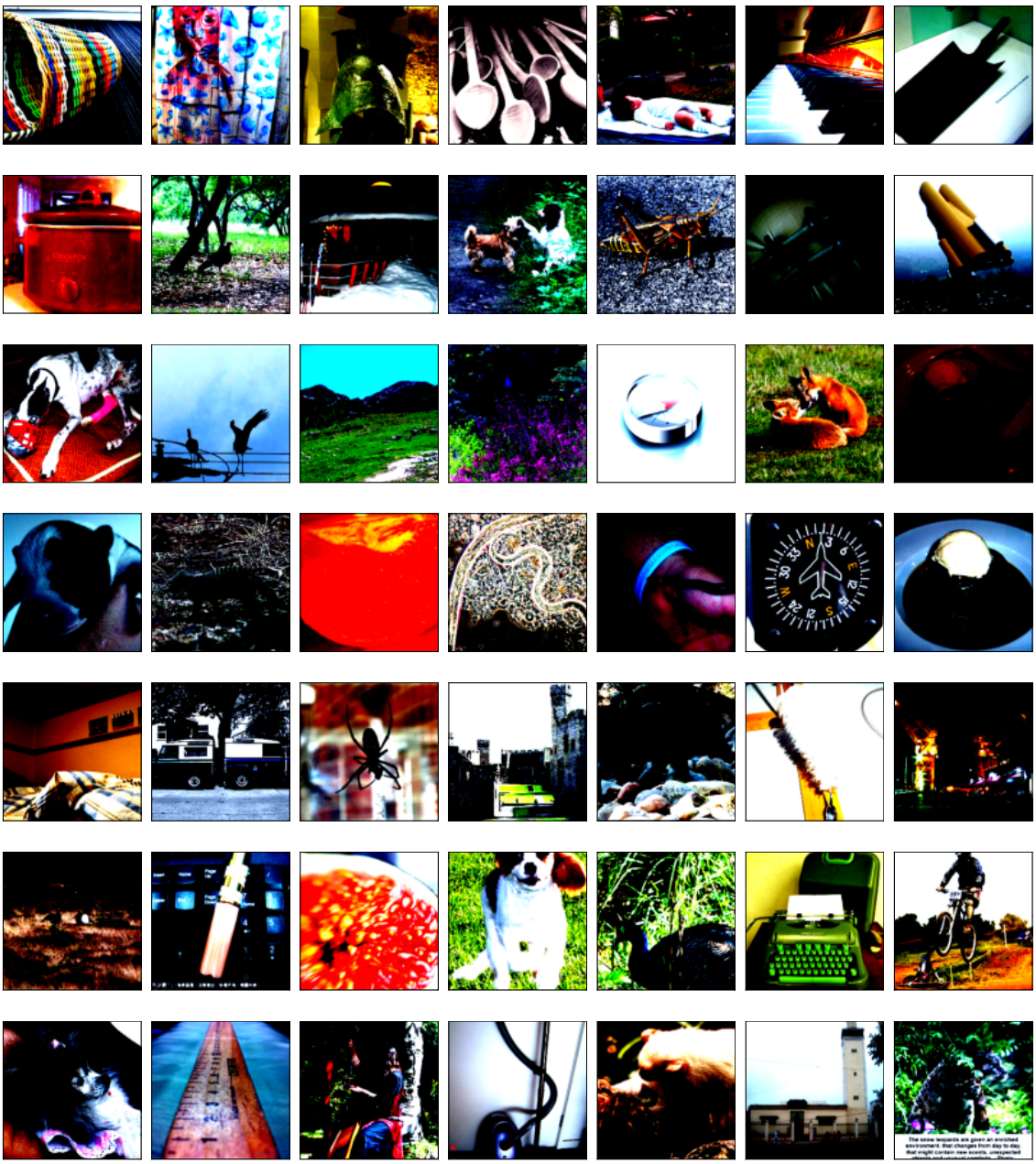


Figure 7: More adversarial examples constructed by the proposed method for ImageNet, with $\epsilon = 1/255$.

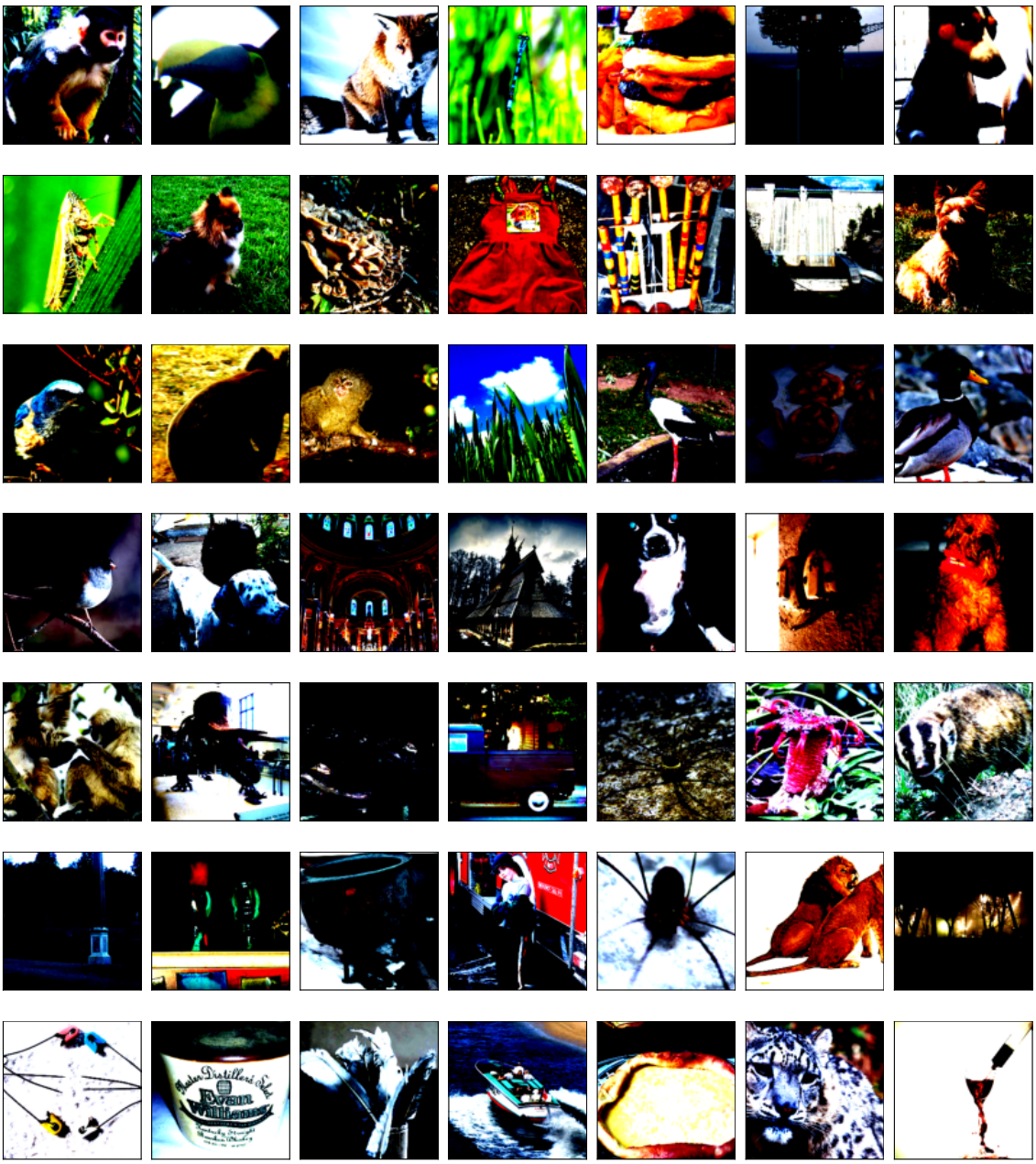


Figure 8: More adversarial examples constructed by the proposed method for ImageNet, with $\epsilon = 4/255$.